## Contents

### 1  Get Started with Visual Builder

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Oracle Visual Builder</td>
<td>1-1</td>
</tr>
<tr>
<td>How to Begin with Oracle Visual Builder Subscriptions</td>
<td>1-2</td>
</tr>
<tr>
<td>Access Oracle Visual Builder</td>
<td>1-2</td>
</tr>
<tr>
<td>Information About Cookies</td>
<td>1-3</td>
</tr>
</tbody>
</table>

### 2  Understand the Building Blocks of Visual Applications

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Building Blocks of Visual Applications</td>
<td>2-1</td>
</tr>
<tr>
<td>Understand Variables</td>
<td>2-2</td>
</tr>
<tr>
<td>Variables and Parameter Passing</td>
<td>2-3</td>
</tr>
<tr>
<td>Expressions</td>
<td>2-4</td>
</tr>
<tr>
<td>Variables and Lifecycles</td>
<td>2-5</td>
</tr>
<tr>
<td>Variables and Events</td>
<td>2-5</td>
</tr>
<tr>
<td>Understand Actions and Action Chains</td>
<td>2-5</td>
</tr>
<tr>
<td>Action Chain Context and Contract</td>
<td>2-6</td>
</tr>
<tr>
<td>Predefined Actions</td>
<td>2-6</td>
</tr>
<tr>
<td>Event Handling for Action Chains</td>
<td>2-7</td>
</tr>
<tr>
<td>Understand Page Flows and Lifecycles</td>
<td>2-7</td>
</tr>
<tr>
<td>The Lifecycle of a Page</td>
<td>2-7</td>
</tr>
<tr>
<td>Page Navigation</td>
<td>2-9</td>
</tr>
<tr>
<td>Understand UI Components</td>
<td>2-9</td>
</tr>
<tr>
<td>The Component Contract</td>
<td>2-10</td>
</tr>
<tr>
<td>Properties</td>
<td>2-10</td>
</tr>
<tr>
<td>Events</td>
<td>2-11</td>
</tr>
<tr>
<td>Child Slots</td>
<td>2-11</td>
</tr>
<tr>
<td>Methods</td>
<td>2-11</td>
</tr>
<tr>
<td>Component IDs and Styles</td>
<td>2-11</td>
</tr>
<tr>
<td>Understand Data Access Through REST</td>
<td>2-11</td>
</tr>
<tr>
<td>Data Binding</td>
<td>2-12</td>
</tr>
<tr>
<td>Mapping to and from REST</td>
<td>2-12</td>
</tr>
</tbody>
</table>
3 Managing Your Visual Applications

Typical Visual Application Development Workflow 3-1
About Your Application Management Tools 3-2
  Manage the Applications 3-2
  View the Application Status 3-4
Create Visual Applications 3-5
  Create a Visual Application 3-5
  Create a Copy of an Application 3-9
  Create a New Version of an Application 3-10
Import and Export Applications 3-11
  Export a Visual Application 3-11
  Import a Visual Application 3-13
About Classic Applications 3-15
  Import Classic Applications 3-16
Stage and Publish Visual Applications 3-18
  About Staging and Publishing Applications 3-19
  Stage a Visual Application 3-20
  Publish a Visual Application 3-23
Build Mobile Applications 3-25
  Change the Splash Screen and Icons Used by a Mobile Application 3-25
  Provide Custom Permission Strings for Mobile Applications on iOS 3-27
  Define a Build Configuration for the Android Platform 3-28
  Define a Build Configuration for the iOS Platform 3-30
  Configure Mobile Application Settings 3-33
  Build a Mobile Application 3-35
  Enable Progressive Web App Support from Mobile Applications 3-38
  Guidelines for Using PWA Support 3-42
Optimize Your Builds 3-43
  About Building Your Application With Grunt 3-43
  Build Your Application Using Developer Cloud Service 3-44
  Build Your Application Locally 3-46
  Visual Builder Grunt Tasks 3-49
  Customize Your Grunt Build Process 3-60
Specify a Custom App URL 3-65
Specify Custom JET and Runtime Versions for Web Apps 3-67
Add Team Members 3-69

4 Develop Applications

Understand an Application's Structure 4-1
Understand the Application Designer 4-3
Check Application Code 4-5
Create Web and Mobile Applications 4-6
  Create a New Web Application 4-6
  Create a New Mobile Application 4-8
  Add a Custom Plugin to Your Mobile Application 4-9
Work with Variables and Types 4-12
  About Variables and Types 4-12
  Create Variables in Artifacts 4-13
  Create a Custom Type 4-15
  Create a Custom Type From an Endpoint 4-17
Work with Flow Artifacts 4-19
  Work with Page Flows 4-20
  Set the Default Page for a Flow 4-21
  Embed a Flow Within a Page 4-22
  Navigate Between Flows in the Root Page 4-23
Work with Actions and Action Chains 4-26
  About Action Chains 4-26
  Create an Action Chain 4-28
  Start an Action Chain with an Event 4-32
  Add an Assign Variables Action 4-34
  Add a Call Action Chain Action 4-36
  Add a Call Component Method Action 4-38
  Add a Call Module Function Action 4-39
  Add a Call REST Endpoint Action 4-40
  Add a Fire Custom Event Action 4-46
  Add a Fire Notification Action 4-46
  Add a Navigation Action 4-47
Design Pages 4-50
  Use the Page Designer 4-51
    The Page Designer Toolbar 4-51
    The Page Editor 4-52
    The Components Palette 4-53
    The Page Structure View 4-54
    The Property Inspector 4-55
    Activate Role-Based Application Preview 4-55
Work with Pages and Page Layouts 4-57
  Create Pages 4-57
  Edit the App Header and Footer 4-58
  About Containers and Layout Components in Pages 4-58
  Add a Container or Layout Component to a Page 4-62
  Add Components to Pages 4-63
About Page Components 4-63
Add a Component From the Components Palette 4-65
Add a Component Using Code Completion in the Code Editor 4-66
Manage Component Visibility Using Conditions 4-66
Add Events to Components 4-69
Add an Image to a Page 4-71
Add an Icon Component to a Page 4-72
Add a Custom Style to a Component 4-74
Filter Data Displayed in a Component 4-75
Create a Filter for Component Data 4-77
Add Page Functionality Using Quick Starts 4-81
About Quick Starts 4-81
Add Data to a Table or List 4-82
Add a Create Page With a Quick Start 4-85
Add an Edit Page With the Quick Start 4-87
Add a Details Page With the Quick Start 4-90
Work with the Functions Editor 4-91
Secure the Application 4-92
About Authentication Roles and User Roles 4-92
About Anonymous Access 4-93
Allow Anonymous Access 4-95
Manage User Roles and Access 4-97
Embed a Web Application 4-99
Configure Basic Authentication for a Mobile Application 4-99
Configure Data Cache and Offline Support 4-101
Work with Application Resources 4-104
About Application Resources 4-105
Import Resources 4-106
Export Resources 4-108
Work with the Image Gallery 4-110
Work with OJET Web Components 4-112
About OJET Web Components 4-112
Work with the Component Exchange 4-114
Get Components From the Component Exchange 4-115
Update a Component from the Component Exchange 4-116
Uninstall a Component 4-117
Import a Web Component Archive 4-118
Create a Custom Component 4-119
Work with Translations 4-120
About Translation Resources 4-121
Understand the Structure of Translation Bundles 4-121
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understand Translation Keys for Display Texts</td>
<td>4-123</td>
</tr>
<tr>
<td>Generate Translation Keys for Display Texts</td>
<td>4-124</td>
</tr>
<tr>
<td>Download Bundles for Translation</td>
<td>4-125</td>
</tr>
<tr>
<td>Use Translation Strings in JavaScript Files</td>
<td>4-127</td>
</tr>
<tr>
<td>Upload Translated Files</td>
<td>4-129</td>
</tr>
<tr>
<td>Create Translation Bundles</td>
<td>4-131</td>
</tr>
<tr>
<td>Integrate Your Application With a Git Repository</td>
<td>4-133</td>
</tr>
<tr>
<td>About Git Integration</td>
<td>4-133</td>
</tr>
<tr>
<td>Add Credentials for Your Developer Cloud Service Account</td>
<td>4-134</td>
</tr>
<tr>
<td>Link Your Visual Application to a Git Repository</td>
<td>4-135</td>
</tr>
<tr>
<td>Pull Files From Your Git Repository</td>
<td>4-136</td>
</tr>
<tr>
<td>Push Your Changes to Your Git Repository</td>
<td>4-137</td>
</tr>
</tbody>
</table>

### 5 Work with Business Objects

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Working with Business Objects</td>
<td>5-1</td>
</tr>
<tr>
<td>Create and Edit Business Objects</td>
<td>5-3</td>
</tr>
<tr>
<td>Create a Business Object</td>
<td>5-3</td>
</tr>
<tr>
<td>View, Create, and Edit Business Object Relationships</td>
<td>5-4</td>
</tr>
<tr>
<td>Add Fields to Business Objects</td>
<td>5-5</td>
</tr>
<tr>
<td>Edit Fields of Business Objects</td>
<td>5-6</td>
</tr>
<tr>
<td>Add a Field for Aggregating Data</td>
<td>5-7</td>
</tr>
<tr>
<td>Add a Formula to a Field</td>
<td>5-8</td>
</tr>
<tr>
<td>How to Format a Date Field</td>
<td>5-9</td>
</tr>
<tr>
<td>Access and Secure Business Objects</td>
<td>5-9</td>
</tr>
<tr>
<td>Work with the Endpoints Tab of a Business Object</td>
<td>5-10</td>
</tr>
<tr>
<td>View the Endpoints of Business Objects</td>
<td>5-11</td>
</tr>
<tr>
<td>Secure Business Objects</td>
<td>5-12</td>
</tr>
<tr>
<td>About Allowing Access to the Catalog API</td>
<td>5-14</td>
</tr>
<tr>
<td>Get an Access Token for Authentication</td>
<td>5-16</td>
</tr>
<tr>
<td>Create Rules for Business Objects</td>
<td>5-17</td>
</tr>
<tr>
<td>About Adding Business Rules</td>
<td>5-17</td>
</tr>
<tr>
<td>About Triggers for Business Objects</td>
<td>5-19</td>
</tr>
<tr>
<td>About Object Triggers</td>
<td>5-19</td>
</tr>
<tr>
<td>About Field Triggers</td>
<td>5-25</td>
</tr>
<tr>
<td>Add an Action to Send Email Notifications</td>
<td>5-27</td>
</tr>
<tr>
<td>Convert a Trigger to Editable Code</td>
<td>5-30</td>
</tr>
<tr>
<td>Build Conditions for Triggers</td>
<td>5-31</td>
</tr>
<tr>
<td>About Object Validators for Business Objects</td>
<td>5-33</td>
</tr>
<tr>
<td>About Field Validators for Business Objects</td>
<td>5-34</td>
</tr>
<tr>
<td>About Object Functions for Business Objects</td>
<td>5-36</td>
</tr>
</tbody>
</table>
7 Work with Business Processes

- About Using Processes in Your Application 7-1
- Add a Business Process to an Application 7-2
- Modify a Business Process Alias 7-3
- Navigate to Your Process Instance 7-4
- Add a Process to a Page 7-5
- Add a Process to an Action Chain 7-8
- Start and Complete Processes from Oracle Visual Builder 7-11
- Configure the Connection to a Process Server 7-11
- Process Server Authentication Options 7-11
- Process Server Instance Options 7-12

8 Understand the Page Model

- Variables 8-1
  - Object Variables 8-1
  - Array Variables 8-2
  - Built-in Variables 8-3
  - Types 8-5
  - Built-in Types 8-5
    - Service Data Provider 8-6
    - MultiService Data Provider 8-27
    - Built-in Array Data Provider 8-46
  - Default Values 8-52
  - Expressions in Default Values 8-53
  - Input Variables 8-54
  - Persisted Variables 8-55
- Actions 8-55
  - Assign Variables 8-56
    - Metadata-Driven Variable Assignment 8-56
    - Assign Variables With a Custom Function 8-58
Call Component Method 8-59
Call Chain Action 8-60
Call Module Function 8-60
Call REST Action 8-61
Fire Data Provider Event Action 8-67
ForEach Action 8-71
Fork Action 8-72
If Action 8-73
Geolocation Action 8-74
Login Action 8-76
Logout Action 8-76
Navigate Back Action 8-77
Navigate to Page Action 8-77
Open URL Action 8-78
Return Action 8-79
Switch Action 8-80
Take Photo Action 8-81
Transform Chart Data Action 8-82
Fire Custom Event Action 8-84
Fire Notification Event Action 8-84
Action Chains 8-85
Chain Actions 8-85
Reference Variables in Action Chains 8-86
Action Chain Variables 8-87
Action Results 8-88
Components 8-88
HTML Source 8-89
Component Events 8-89
Custom Events 8-90
Import Custom Components 8-91
Security 8-91
Security Configuration 8-91
Security Provider 8-92
User Information 8-93
Error Handling 8-93
Translations 8-94
Helper Utilities 8-96
Rest Helper 8-96
Module Function Event Builder 8-98
Events 8-99
Declared Events 8-100
9 Common Use Cases

Enable Client-Side Validation for a Form 9-1
Format Row Values in a Table Conditionally 9-3
Validate Dates in Forms 9-4
Get The Label Value from a Selected Item in a List 9-5
Run Visual Builder Applications On Other Servers 9-6
Style Visual Builder Applications 9-6
Use a Check Box to Select Rows in a Table 9-7
Preface

Developing Applications with Oracle Visual Builder in Oracle Integration describes how to use a web-based visual development tool to create and publish custom web and mobile applications that can integrate business objects and Fusion Applications REST services to extend SaaS services.

Topics:

- Audience
- Related Resources
- Documentation Accessibility
- Conventions

Audience

Developing Applications with Oracle Visual Builder in Oracle Integration is intended for developers who want to create and publish modern enterprise applications using a visual development tool and still have full access to the source code of their applications.

Related Resources

For more information, see these Oracle resources:

- Oracle Public Cloud
  
  http://cloud.oracle.com

- Learn About Oracle Integration in Administering Oracle Integration

- Manage Instance Settings in Administering Oracle Visual Builder in Oracle Integration

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.

Conventions

The following text conventions are used in this document:
## Conventions

<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><strong>monospace</strong></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 with Visual Builder

To use Oracle Visual Builder you should be familiar with the tools available for building your applications.

Topics:
- About Oracle Visual Builder
- How to Begin with Oracle Visual Builder Subscriptions
- Access Oracle Visual Builder
- Information About Cookies

About Oracle Visual Builder

Oracle Visual Builder is a visual and declarative cloud environment for developing and hosting engaging mobile and web applications.

Oracle Visual Builder provides easy access to data from any REST-based service, and enables the creation of custom reusable business objects for storing and managing data. Using the cloud-based visual development tools, you can create and test responsive web applications and native mobile apps without the need to install any additional software. The visual designer enables you to quickly layout pages in your applications by dragging and dropping UI components, customizing their attributes and defining their behavior.

While Oracle Visual Builder provides a rich set of visual designers to enhance productivity, experienced developers always have access to the source code of their applications. Developers can enhance their application’s functionality using the code editors to write custom JavaScript functions and edit application metadata. Oracle Visual Builder leverages the open source Oracle JavaScript Extension Toolkit (Oracle JET) to create engaging web and mobile interfaces. The reusable Oracle JET Composite Component Architecture (CCA), built based on the Web Component standard, can be added to the application to further enhance the user experience.

Oracle Visual Builder also provides the following tools for building and publishing applications:
- Data management tools for managing business objects and importing and exporting data
- Publishing tools for staging and publishing your applications
- Advanced tools for configuring role-based security and user access management
- Collaboration tools for sharing application resources between team members
- Robust tools for describing requests and responses to Oracle SaaS REST services in the integrated catalog

The following figure shows the architecture of Oracle Visual Builder applications.
How to Begin with Oracle Visual Builder Subscriptions

Here's a summary of the key steps to help Oracle Cloud account administrators get started with Oracle Visual Builder:

1. Sign up for a free credit promotion or purchase a subscription. See Requesting and Managing Free Oracle Cloud Promotions or Buying an Oracle Cloud Subscription in Getting Started with Oracle Cloud.


3. Learn about the users and roles. See Oracle Cloud User Roles and Privileges in Getting Started with Oracle Cloud.

4. Create accounts for your users and assign them appropriate privileges and roles. See Adding Users to a Traditional Cloud Account or Adding Users to a Cloud Account with IDCS in Getting Started with Oracle Cloud. Users are automatically sent an email with the login details for their account.

Access Oracle Visual Builder

To develop applications using Oracle Visual Builder, you access the service through a web console.

To access Oracle Visual Builder:

1. Sign in to your Oracle Cloud Account using your user name and password.
   
   If you have the Welcome email you received when you signed up for Oracle Cloud, see Signing In For the First Time or Signing In to Your Traditional Cloud Account in Getting Started with Oracle Cloud.

   If you don't have access to your Welcome email, see Signing In From the Oracle Cloud Website.
2. Enter your identity domain and user credentials. Click **Sign In**.

# Information About Cookies

When a user visits a published web application, a combination of cookies are used for storing authentication and session information.

The following cookies are used to store information about sessions, visits and authentication. The information we observe about visitor behavior is stored on our servers, not in the cookie placed on the browser. The cookies we use are usually an anonymous unique identifier, which provides a means of determining whether a visitor has visited the application before but does not provide any means of identifying the visitor. None of the cookies contain personally identifiable information, although, in accordance with standard HTTP protocol, the visitor’s IP address is passed to our servers as part of the HTTP request. All cookies are secured with encryption and sent over HTTPS. The following table describes the cookies that are saved to the browser of visitors visiting a published application.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSESSIONID</td>
<td>The JSESSIONID cookie is a transient cookie used for session management. This cookie only contains a session identifier. The cookie does not contain any personal details.</td>
</tr>
<tr>
<td>OMAuthnCookie</td>
<td>The OMAuthnCookie cookie is generated by Oracle Access Manager for all clients using an Oracle Cloud service. A valid OMAuthnCookie is required for a session.</td>
</tr>
</tbody>
</table>
|                     | • Authenticated User Identity (User DN)  
|                     | • Authentication Level  
|                     | • IP Address  
|                     | • SessionID  
|                     | • Session Validity (Start Time, Refresh Time)  
|                     | • Session InActivity Timeouts (Global Inactivity, Max Inactivity)  
|                     | • Validation Hash  
<p>|                     | Removing the cookie will cause the user to be logged out. The user will need to re-authenticate the next time the user requests a protected resource.                                                     |
| X-AppBuilder-SessionId | The X-AppBuilder-SessionId cookie is a persistent cookie that expires 24 hours after it is created and contains a unique user ID (UUID) and time stamp. This cookie is only used to store visitor behavior information across sessions for billing purposes. This cookie is used for published classic applications. |
|                     | Removing the cookie might result in each new session being recorded as a new visit.                                                                                                                             |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBCS_MTRICS_&lt;app name&gt;_&lt;app version&gt;</td>
<td>This cookie is a persistent cookie that expires at midnight on the day it is created and contains a time stamp. The purpose is help count unique visits, and to ensure that multiple visits by the same user between when the cookie is created and when it expires are counted as one unique visit. If the cookie is removed before it expires, a new cookie is created on the next visit and the visit will be counted as a new unique visit. This cookie is only used for internal metrics and is not used for billing purposes. This cookie is used for staged and published visual applications.</td>
</tr>
</tbody>
</table>
Understand the Building Blocks of Visual Applications

To develop applications with Oracle Visual Builder, you need to understand a few basic concepts.

Topics:

- About the Building Blocks of Visual Applications
- Understand Variables
- Understand Actions and Action Chains
- Understand Page Flows and Lifecycles
- Understand UI Components
- Understand Data Access Through REST

About the Building Blocks of Visual Applications

The basic components of a visual application are mobile applications, web applications, service connections, business objects, and processes.

The basic building blocks of a mobile or web application are user interface (UI) components, variables, action chains, page flows and page navigation, and data access through REST endpoints.

The building blocks and their interactions can be summarized as follows.

- Variables are the mechanism used to store and manage client state. Every variable has a type and a scope.
- An action chain is composed of a set of one or more individual actions. The action chain is triggered by an event. (For example, a button click can trigger navigation to a page.) Each action represents a single asynchronous unit of work. An action chain can define input parameters and local variables that are available only in the context of that action chain, and can also access application-scoped input parameters and variables.
- Page flows and page navigation govern the transmission of information from one page to another. Each individual page has a lifecycle, as does an application. Each lifecycle event (entry or exit from a page, for example) can provide a trigger for an action chain.
- A UI component encapsulates a unit of user interface through a defined contract – specifically, the Oracle JavaScript Extension Toolkit (JET) components contract. Component attributes are bound to variables, and component events and variable changes trigger action chains.
- All data entering a mobile or web application is based on REST. This data can come from custom business objects and from business objects provided by service connections. Actions and variables control how data is sent to and from a
REST endpoint in a mobile or web application. A developer can create a type that matches the REST payload and pass the data using a variable of that type. The following figure shows the interactions among these building blocks.

Understand Variables

A variable is the basic building block for managing client state. It is of a specific type and exists in a specific scope.

A scope defines the lifecycle of a variable, and the framework automatically creates and destroys the variables depending on the semantics of the scope. The following scopes are supported.

- **Page scope**: State is accessible only within the context of the specified page. All state is initialized with default values when the page is entered, and destroyed when the page is exited.
- **Application scope**: State is accessible in all parts of the application and in all pages.
- **Flow scope**: State is accessible in all pages contained in the current flow.
- **Action chain scope**: State is accessible in the current action chain.

Variables store intermediate state on the client between the Oracle Visual Builder user interface and REST services. Components are principally bound to these variables, and the behavior of the variables is governed by actions.

A variable’s type can be a primitive, a structured type (which can consist of other types), a dynamic type, or a builtin type.
A variable value that has not yet been instantiated is undefined. A variable is guaranteed to be instantiated and its initial value set just before the vbEnter event is raised (see The Lifecycle of a Page).

The initial value of a variable is determined using the defaultValue property set on the variable, along with its type.

When its value changes, a variable emits an event. This event may trigger an action chain. You can define variables as input parameters, with their value determined by the inputs to the page or module. These inputs can be URL parameters for bookmarking, for example.

For more information on variables, see Variables.

### Variables and Parameter Passing

You can use a variable to pass a parameter between pages. You can mark a page variable as an input, specifying how it becomes part of the contract in order to navigate to that page. You can then further mark it as required, implying that it must be set in order to navigate to that page.

The following table lists the available properties for variables.

<table>
<thead>
<tr>
<th>Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Yes</td>
<td>A specific primitive type (string, boolean, number, and so on); a structured type such as an array or object, for which each field can either be a primitive or a structure; a dynamic type (any); or a built-in type, such as ServiceDataProvider or ArrayDataProvider (see Built-in Types).</td>
</tr>
<tr>
<td>input</td>
<td>No; applicable only if the property is within the page scope</td>
<td>How the variable becomes part of the page contract for incoming navigation. The value is either none (the default), fromCaller (indicating that it will be passed internally), or fromURL (indicating that it will be passed via the URL).</td>
</tr>
<tr>
<td>required</td>
<td>No</td>
<td>Whether or not the variable must be set</td>
</tr>
<tr>
<td>defaultValue</td>
<td>No</td>
<td>The default value for the variable to be initialized. If no default value is provided, the value is &quot;not set&quot; or undefined. The defaultValue can be bound to an expression, or it can be a structure that uses expressions that reference other variables.</td>
</tr>
</tbody>
</table>
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>persisted</td>
<td>No</td>
<td>Use persisted variables when you want the lifespan of the variable to be longer than the page. For example, an authorization token can be kept for the duration of a session. This ensures that even if the page is refreshed, the token will still be available throughout the session. To store a variable across sessions, use &quot;device&quot; instead of &quot;session&quot;. The variable itself still exists only in its defined scope. Can be set to the following values:</td>
</tr>
<tr>
<td>session</td>
<td>This property will be applied when entering the page, but only during the current browser session.</td>
<td></td>
</tr>
<tr>
<td>history</td>
<td>The variable value is stored on the browser history. When navigating back to a page in the browser history using the browser back button, the value of the variable is restored to its value at the time the application navigated away from this page.</td>
<td></td>
</tr>
<tr>
<td>local</td>
<td>The variable is stored in the browser local storage so persisted on the device where the application is running even if the browser is closed.</td>
<td></td>
</tr>
</tbody>
</table>

Expressions

An expression may refer to other variables, system properties, statics, and the like. For example:

$variables.total = $variables.sum * (1 + $variables.taxRate)

The Oracle Visual Builder user interface performs dependency analysis of the expressions to correctly order expression evaluation and detect cycles.

If the value of any variable referenced in an expression changes, the expression is immediately reevaluated.

An expression can be used in the default value of a variable.

You can use the following implicit objects in expressions. All are prefixed by a dollar sign ($).

<table>
<thead>
<tr>
<th>Name</th>
<th>Where Available</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$application</td>
<td>Everywhere</td>
<td>The application object</td>
</tr>
<tr>
<td>$page</td>
<td>In the current page</td>
<td>The current page instance</td>
</tr>
<tr>
<td>$variables</td>
<td>Every scope that has a variables property</td>
<td>A shortcut for $most_specific_scope.variables in the current scope. In a page, $variables is a shortcut for $page.variables.</td>
</tr>
<tr>
<td>$chains</td>
<td>Every scope that has a chains property</td>
<td>A shortcut for $most_specific_scope.chains</td>
</tr>
</tbody>
</table>
### Variables and Lifecycles

Application and page variables are constructed automatically in a specific application or page lifecycle stage.

Input parameters that are passed by means of navigation rules, or bookmarkable variables that are provided on the URL, are automatically assigned to their corresponding variables. When you modify the value of a bookmarkable variable, the URL is automatically adjusted to match that new value (that is, a new history state is pushed). In this way the page is always bookmarkable and does not require any specific user action in order to be bookmarked.

### Variables and Events

A variable triggers an onValueChanged event when it is modified. This event is triggered only when the value is actually changed; setting a variable value to the same value does not trigger an event. The variable must be explicitly changed to send the event. For example, if a variable is a complex type, modifying an inner property does not fire this event; the entire variable must be set by means of an API call. In this case, the framework can add to the payload those parts of the structure that have changed. For example, if you changed the name property of an Employee and then reset the Employee, the framework would send an event that the Employee changed, and as part of the payload indicate that the name has changed.

An onValueChanged event can trigger a user-defined action chain. The trigger has the payload of the former and new values of the variable.

For more information, see Understanding Actions and Action Chains.

### Understand Actions and Action Chains

An action chain is made up of one or more individual actions, each of which represents a single asynchronous unit of work. Action chains are triggered by events.

An action chain, like a variable, has a scope: it can be defined at the application level or the page level. You can call an application-scoped action chain from any page. You can call a page-scoped action chain only from the page on which it is defined.

<table>
<thead>
<tr>
<th>Name</th>
<th>Where Available</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$chain</td>
<td>Actions executing in an action chain</td>
<td>The chain in which the action is executing</td>
</tr>
<tr>
<td>$parameters</td>
<td>In the beforeEnter event</td>
<td>The input parameters for the page. This object is needed because page variables do not exist yet in the vbBeforeEnter event.</td>
</tr>
<tr>
<td>$listeners</td>
<td>In a flow or page</td>
<td>The event listeners of a flow or page</td>
</tr>
<tr>
<td>$event</td>
<td>Event listeners and variable onValueChange listeners</td>
<td>For an event listener on a component, $event contains the Event JavaScript object that the component passes to the listener. For an event listener on a custom event, $event contains the payload for that event. For an onValueChanged listener on a variable, $event is a structure with the properties name, oldValue, value, and diff (itself a structure).</td>
</tr>
</tbody>
</table>
To create an action chain, you can define your own actions and can also use predefined actions. Actions within a particular chain run serially, and multiple action chains can run concurrently. Action chains simplify the coordination of asynchronous activities.

A single event may simultaneously trigger multiple action chains. For example, the page enter event may trigger multiple data fetch action chains simultaneously.

An action is a specific function that performs a task. In JavaScript terms, an action is a Promise factory. An action can exist only within an action chain, not independently.

Action Chain Context and Contract

Action chains have a well-defined context and contract: an action chain orchestrates its underlying actions, coordinating state flow and the execution path. The action chain can define input parameters and local variables that are only available in that context. An example of an action chain is one that makes a REST call (first action), then takes the result of that and stores that in a variable (second action).

An action chain maintains its own context, which is accessible through an implicit object called $chain. Actions may export new state to that context, but it is only available to future actions along that same action chain. An action chain can be created in the context of a page or the application and exists within the scope of the page or the application. It has a defined interface and contract and can be called by event triggers using its ID.

The action chain contract has three parts.

<table>
<thead>
<tr>
<th>Action Chain Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String identifier for the action chain</td>
</tr>
<tr>
<td>Input parameters</td>
<td>Zero or more variables that can be passed into the action chain and added to the action chain context</td>
</tr>
<tr>
<td>Variables</td>
<td>Zero or more variables that are internal to the action chain and usable internally by actions</td>
</tr>
</tbody>
</table>

For more information, see Action Chains.

Predefined Actions

The predefined actions for an action chain include Navigate to Page and Assign Variables. An action has the following parts that the developer can define.

<table>
<thead>
<tr>
<th>Action Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String identifier for this action instance. This action part is optional, since the ID is necessary only if you wish to refer to the action's results later in the action chain.</td>
</tr>
<tr>
<td>Configuration</td>
<td>Any properties of the action that the user can configure. For example, for the Navigate to Page action, the page to navigate to and any parameters required for that navigation.</td>
</tr>
<tr>
<td>Outcomes and Results</td>
<td>An action may have multiple potential outcomes (such as success or failure, or a branch). It can also return results.</td>
</tr>
<tr>
<td>Exported State</td>
<td>An action may export state that is available to future actions within the context of the same action chain.</td>
</tr>
</tbody>
</table>
The predefined actions include conditionals and other processing instructions. For example, you can use if and switch actions that take an expression and offer multiple different chain continuations depending on the result.

For details about predefined actions, see Actions.

Event Handling for Action Chains

Action chains are defined at the application or page level and triggered by a specific event, such as onValueChange (for a variable), or vbEnter. An event may include a payload, which can then be used within the action chain. A payload may be passed into an action chain through the input parameters. The Oracle Visual Builder user interface can help you create action chains automatically (with appropriate input parameters) based on a particular event.

Understand Page Flows and Lifecycles

The page flow governs how information is transferred between pages. The page lifecycle governs the state of an individual page.

A page has a defined lifecycle that permits you to listen to certain events that are triggered as part of the lifecycle. Examples of page lifecycle events are enter and beforeExit.

One or more pages form a page flow. Within a flow, you can set up navigation from page to page.

Navigation actions can be internal or external. An internal navigation action is composed of the ID of the page to navigate to along with any parameters that are specified for that page. An external navigation action is defined by an external URL. These actions are defined in the page model.

An application also has a lifecycle and flow. An application can contain multiple page flows.

The Lifecycle of a Page

An individual page has defined lifecycle states upon entering and leaving, and each state has a trigger. For some states, you can provide action chains in response to the triggers. Other states are internal, but help illustrate what happens and when in the system.
<table>
<thead>
<tr>
<th>Event</th>
<th>Applies To</th>
<th>Can Cancel Navigation</th>
<th>Event Payload</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
</table>
| vbBefore Enter      | Page       | Yes                    | Previous route, sanitized input parameters                                      | Typed state        | Dispatched to a page before navigating to it. Oracle Visual Builder will navigate to this page, but has not yet started the navigation and has not torn down the previous page's state. A developer can cancel navigation for this event (for example, if the user does not have permission to enter this page) by returning an object with the property `cancelled` set to `true`. A developer can also redirect the user to another page instead (for example, it can take the user to a login screen).
For success cases, this can return some state that can be passed into the `vbEnter` state.
After this state is exited, the previous page's state can be torn down.
Application state is available on a read-only basis. No page state is available.
The following variable scopes are available:
- `$application`: All application variables
- `$flow`: All parent flow variables
- `$parameters`: All page input parameters from the URL |
| vbEnter             | Page or flow | No                     | State from vbBefore Enter                                                      | None               | Dispatched after all the page-scoped variables have been added and initialized to their default values, values from URL parameters, or persisted values.
The following variable scopes are available:
- `$application`: All application variables
- `$flow`: All parent flow variables
- `$page`: All page variables
This state is generally used to trigger data fetches, which may occur concurrently. |
**Page Navigation**

Every page in the application has a name, which you can specify and change. You use the page name to navigate from one page to another within a page flow.

To configure a Navigate to Page action, specify the following parameters:

- The page to navigate to, or an expression that resolves to that value
- Values for required input parameters and for any optional parameters that you use

There are two possible results:

- Navigation was successful
- Navigation was cancelled from the page we are navigating to

**Understand UI Components**

User interface (UI) components encapsulate a unit of user interface interaction through a defined contract.

The Web Component contract exposes the functionality of a component through the user interface, enabling the component to interact with other parts of the application. Oracle Visual Builder supports the Oracle JavaScript Extension Toolkit (JET) components contract, which adds data binding, component metadata, and dependencies on top of the Web Component contract. The Oracle JET components contract exposes a custom Document Object Model (DOM) HTML element with custom properties, events, and methods. The property binding added by Oracle JET supports both one-way (read-only) and two-way (read/write) binding. In general, the component properties are bound to variables, and the component events trigger action chains.
A component can have zero or more slots that can hold one or more children of that component. For example, a toolbar can contain a number of buttons.

You can add components to an application from the Component Palette. You can also use custom JET components, including those supplied by the Component Exchange. See Work with the Component Exchange for details.

For simple use cases, you can use a simple HTML component and corresponding view model implementation.

For details about Oracle JET, see http://www.oracle.com/webfolder/technetwork/jet/index.html. The Oracle JET Cookbook provides detailed information about using all the supported components at http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html

The Component Contract

A UI component, whether shipped by Oracle Visual Builder, provided by a partner, or created by you, must follow the same component contract. This contract allows Oracle Visual Builder to expose the functionality of a component declaratively through the Oracle Visual Builder user interface. If you need to add functionality to a component, you can expand the capabilities of that component, and the new functionality is then expressed in that component's interface.

The component contract has four aspects: properties, events, child slots, and methods.

Properties

A component has properties that you can bind to variables or expressions by means of the Oracle Visual Builder user interface. These properties can affect the state of a component (for example, the value of an input text field) or affect its rendering (for example, enabled or disabled). A component property has a specific type, which matches the types available for a variable, and may itself be structured or a collection. A property may also be required as part of the component interface.

In addition to a type, a component property may also have additional metadata (as defined in the JET design time metadata for properties), such as a display name or description.

There are two kinds of properties, one-way and two-way.

<table>
<thead>
<tr>
<th>Property Type</th>
<th>Description</th>
<th>Can Be Bound To</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-way (read-only)</td>
<td>The component reads the value of this property. If the expression that the property is bound to changes, the component will be notified of this change.</td>
<td>Expressions (which may contain variables)</td>
</tr>
<tr>
<td>Two-way (read/write)</td>
<td>The component can read the value of this property and can also write back to that property. If the variable is modified externally, the component will be notified of this change.</td>
<td>Variables</td>
</tr>
</tbody>
</table>
Events

A component can fire zero or more events (for example, an onClick event for a button). Each event has a payload. The Oracle Visual Builder user interface allows the developer to listen for any of these events and to expose the event payload. An action chain can then process the event.

Child Slots

A slot is a placeholder inside a web component that you can fill with your own markup. A component can have zero or more slots that can hold one or more children. Any children not assigned to a specific slot are assigned to the default slot.

Methods

A component can have zero or more methods that can be called on the component to perform an action (for example, to flip a card). These methods may have parameters that are defined as part of the component interface. The Oracle Visual Builder user interface provides an action within an action chain that allows the user to call a component method and fill in the parameters using expressions.

Component IDs and Styles

A component can have a configurable ID to allow it to be referenced from an action. In addition, you can bind component style classes to an expression.

Understand Data Access Through REST

All data access to and from a client application occurs through REpresentational State Transfer (REST) calls.

The Oracle Visual Builder user interface provides access to two basic kinds of data:

- Custom business objects internal to Oracle Visual Builder, which you can create yourself and use in applications
- Business objects accessed through service connections, which you can set up to access external web services

When you create a custom business object, a REST API is automatically created for you, with GET, POST, PATCH, and DELETE endpoints.

When you create a service connection, you can obtain REST APIs in one of the following ways:

- Select objects from a service catalog
- Provide a service specification document inSwagger or Oracle Application Developer Framework (ADF) Describe format
- Specify an endpoint URL, an HTTP method, and an action hint

Each of these mechanisms generates REST APIs for you to use. You can specify request and response payload structures in JSON format, and you can provide a subset of query parameters to expose to the Oracle Visual Builder user interface.
Parameters can have a type (but are assumed to be primitives). You can also use the provided REST helper utility to call REST endpoints.

For full details on using REST for data access in Oracle Visual Builder, see Accessing Business Objects Using REST APIs.

Data Binding

You can create variables and action chains to call REST endpoints from your applications, retrieving and sending data to and from the endpoints. Typically, the type of the variable matches the structure of the REST payload. You have the option of defining your own type that more closely matches your use case, and then mapping from the REST payload to a variable instance that uses that type. For example, for advanced cases, you could define a variable type that matches your own page design, and then map one or more REST payloads to that type. To send that data back to a service or services, you would again map the data of that variable to the REST payloads.

Components are bound to variables. These variables do not have any intrinsic knowledge of where their data is derived from or what their data is used for. To populate a variable from a REST call, you assemble an action chain from an action making that REST call and an action assigning the result to that variable. In the common case, the Oracle Visual Builder user interface automates the creation of that variable to match the payload of the REST call, enabling you to quickly bind the REST call's payload in your application pages. To handle a POST or DELETE action, you compose an action chain with the REST action, passing in the variable as the payload.

Mapping to and from REST

In more advanced cases, you may wish to define a model (through the use of a variable) that more closely matches your specific application. In other cases, the GET and POST (or equivalent methods) may be asymmetrical or may be from different services entirely. In these cases, you can map the REST payload to and from that variable.
Managing Your Visual Applications

Oracle Visual Builder provides tools for managing your visual applications, including tools for importing and exporting applications and resources, creating versions, and staging and publishing applications.

Topics:
- Typical Visual Application Development Workflow
- About Your Application Management Tools
- Create Visual Applications
- Import and Export Applications
- Stage and Publish Visual Applications
- Specify a Custom App URL
- Specify Custom JET and Runtime Versions for Web Apps
- Add Team Members

Typical Visual Application Development Workflow

Oracle Visual Builder supports all stages of the development workflow, from creating to staging and publishing the application.

Oracle Visual Builder imposes no specific order for building the artifacts in your application and defining the building blocks. How you proceed is personal preference and determined by the way you planned your application. If you already know the data sources that you will use, or the structure of objects that you have, you might want to start by defining the service connections and business objects.

The following table provides a high-level description of some of the tasks that you will typically perform when building an application.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>More Information</th>
</tr>
</thead>
</table>
| Create a new application or version | Use the wizard on the Home page to create a new application, or create a new version of an existing application | Create a Visual Application
<p>|                               |                                                                             | Create a New Version of an Application |
| Create service connections    | Create connections to external REST web services and select and configure the endpoints that you want to use in your application. | About Service Connections               |</p>
<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create business objects</td>
<td>Define your own custom REST endpoints for data in your database based on the needs of your applications.</td>
<td>Create a Business Object</td>
</tr>
<tr>
<td>Build the web or mobile app</td>
<td>Design the flow and layout of the pages in your application.</td>
<td>Understand an Application’s Structure</td>
</tr>
<tr>
<td>Secure the application</td>
<td>Create application roles and configure the permissions for business objects.</td>
<td>Secure the Application</td>
</tr>
<tr>
<td>Stage and test the application</td>
<td>Use the Application Options menu in the toolbarr of your application to stage the application when you are ready to do more thorough testing of your application. You can share the link to the staged application with people for testing.</td>
<td>Stage a Visual Application</td>
</tr>
<tr>
<td>Import real data and check the schema</td>
<td>Use the Data Manager to import data into your databases from a file or from the live database.</td>
<td>Manage Data During the Development Lifecycle</td>
</tr>
<tr>
<td>Publish the application</td>
<td>Publish the staged version and either import data or use your live database.</td>
<td>Publish a Visual Application</td>
</tr>
</tbody>
</table>

### About Your Application Management Tools

The Home page is the primary console for managing your applications and for creating new applications.

The Home page is the main entry point and the first page that you see each time that you log in to Oracle Visual Builder. On the Home page you can see the status of each version of your applications and select the application that you want to open. The Home page also contains tools for managing the publishing lifecycle of each application.

### Manage the Applications

Each row in the table on the Home page represents a version of one of your applications. The identity domain might contain many applications, but your Home page will only display the applications that you created or those where you are included as a team member.

Each row contains an Application Options menu with commands for managing the application.
The following table describes the commands that you might see in the menu.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Opens the development version of the application</td>
</tr>
<tr>
<td>Duplicate</td>
<td>Creates a clone of this version of the application, including the content of the database.</td>
</tr>
<tr>
<td>Rename</td>
<td>Opens a dialog box where you can change the name of the application.</td>
</tr>
<tr>
<td>Export</td>
<td>Creates a ZIP archive of the application that can be imported as a new application. When exporting the application, you can choose if you want the exported archive to include the data stored in your custom business objects.</td>
</tr>
<tr>
<td>Import</td>
<td>Opens a dialog that you can use to create an application by uploading an application archive (ZIP or OVB) from your local system.</td>
</tr>
<tr>
<td>New Version</td>
<td>Creates a new version of the same application. By default the new version is a development version. Version numbers are automatically increased incrementally.</td>
</tr>
<tr>
<td>Menu Item</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Settings</td>
<td>Opens an editor for configuring the application’s settings and viewing the application API URLs. Each application version has a dedicated Settings editor.</td>
</tr>
<tr>
<td>Stage</td>
<td>Opens a dialog box where you can specify the database option for the staged application. When an application is staged, a link to the staged version is displayed in the tile.</td>
</tr>
<tr>
<td>Publish</td>
<td>Opens a dialog box where you can specify the database option and publish the staged version of your application.</td>
</tr>
<tr>
<td>Lock / Unlock</td>
<td>Enables you to lock a live application to prevent any users from using the application. You would usually use this command when you are going to update the live application with a newer version. The Unlock option is displayed only when the live application is locked.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Rolls back the live version to the previous live version. This is only available for the current live version.</td>
</tr>
<tr>
<td>Process Actions</td>
<td>Opens a submenu where you can navigate to pages where you can edit your processes or manage roles. The menu also contains a command to deploy your processes.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the application from the Identity Domain. You cannot undo this action.</td>
</tr>
</tbody>
</table>

**View the Application Status**

Each list item displays the current status of the application. The default status for new applications is Development. When available, you can expand a list item to show staged and live versions of the application.

You can filter the list of applications displayed on the page by name, status and type. Click the Filter toggle at the top of the page to open the filter options drawer and select the filters you want to apply. You can use the Sort By dropdown list to organize the order that the tiles are listed on the page.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>World Population App</td>
<td>Dev</td>
<td>1.1.0</td>
</tr>
<tr>
<td>World Population App</td>
<td>Stage</td>
<td>1.0.1</td>
</tr>
<tr>
<td>World Population App</td>
<td>Live</td>
<td>1.0</td>
</tr>
<tr>
<td>Tech Support Register</td>
<td>Dev</td>
<td>1.0</td>
</tr>
<tr>
<td>Charts and Gauges</td>
<td>Dev</td>
<td>1.0</td>
</tr>
<tr>
<td>PartnerFinder4UAI1</td>
<td>Dev</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The following table describes the application status.
<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>This status is the default for all new versions of applications and is used if the version has not been staged or published. You can make changes to the development version at any time.</td>
</tr>
<tr>
<td>Staging</td>
<td>This status indicates that you recently staged this version of the application. The tile contains a link that will open the staged version in your browser.</td>
</tr>
<tr>
<td>Live</td>
<td>This status indicates that this version of the application was published and is now read-only. An application can no longer be modified after it is published. The link in the tile will open the live version in your browser.</td>
</tr>
<tr>
<td>Live Locked</td>
<td>This status indicates that this version of the application was published but is currently locked and it cannot be opened in your browser. Use the Application Options menu to lock and unlock a live application. You should lock an application when you want to export the live database prior to publishing a new version.</td>
</tr>
<tr>
<td>Pending Push Request</td>
<td>This status indicates that a request was submitted to push the staged version of the application to an associated environment and the request is pending a response. You can make changes to your application while you are waiting for a response but the request must be accepted or denied before you can stage a new version.</td>
</tr>
<tr>
<td>Obsolete</td>
<td>This status indicates that this version was published but has been superseded by a newer version. Obsolete applications are read-only.</td>
</tr>
</tbody>
</table>

## Create Visual Applications

You create new visual applications and versions of existing applications on the Oracle Visual Builder home page.

Topics:
- Create a Visual Application
- Create a Copy of an Application
- Create a New Version of an Application

## Create a Visual Application

A visual application is a collection of resources that you use to develop web and mobile applications. A visual application includes metadata in JSON files that describe data sources and business objects, and includes the HTML and JavaScript files of the web application.

To create a visual application:

1. **Select Visual Applications** in the menu on the Home page if you are not on the Visual Applications home page.

To create visual applications you must first open the home page for Visual Applications.
2. Click **New** on the Visual Applications home page.

3. Select an application template. Click **Next**.
   
   You see a page for selecting an application template only when there are multiple application templates in your instance. The default template Empty Application is selected automatically when there are no other templates. The Empty Application template does not create any artifacts, apps or other resources. Other templates might create resources or apps that already include artifacts.
4. Type the application name in the Create Application dialog box. Click **Finish**.

The Application ID field is automatically populated based on the application name that you provide. You can modify the Application ID if you wish, but the Application ID must be unique in your identity domain.

Your new visual application opens automatically when you create it and displays the Welcome screen. Your new application does not contain any artifacts, but the wizard creates the file structure of the application and some resources and files for you by default. You can use the Welcome screen to help you decide which artifacts you want to create first. Click any of the tiles in the Welcome screen to open the panel in the Navigator where you can create and manage the artifacts. You can add multiple
mobile and web apps, and all the apps can access the data sources that you expose in your visual application.

For example, if you click Web Apps, the Web Applications tab of the Navigator opens. The other tabs are Mobile Applications, Service Connections, Business Objects, Components, Processes, and Source View.
Create a Copy of an Application

You can create copies of an application using the duplication application option in the Application Options menu or by importing an application archive.

A copy of an application contains all the resources and the database schema of the source, but will have a different name, application ID and URI. You can use the copy option if the application you want to copy is visible on your Home page.

To create a copy of an application:

1. On the Home page, locate the version of the application that you want to copy.
2. Open the Application Options menu and choose Duplicate.
3. Type the application name and application ID in the Duplicate Application dialog box. Click Duplicate.

The application ID that you enter will be exposed in the URI for the new application.
Create a New Version of an Application

You can create versions of applications to enable parallel, independent development of an application.

Each version of an application is an independent branch with its own copy of the resources and database schema. Using multiple versions enables you to work on one version, for example, to fix an issue, without disrupting ongoing development on another branch. You can stage and publish any version of your application. You can have multiple versions of your application staged simultaneously, but you can only have one live version. Each staged version has a unique URI to help you identify the version, but all versions will have the same URI when published. After you publish a version of your application it is locked as “read-only”. To make any changes to your application after it is published, for example, to fix an issue, you need to create a new version and fix the issue in the new version.

If you make a change in one version, for example, to fix an issue, you will also need to manually make those changes to any other versions of your application that you want to include that change.

To create a new version of an application:

1. On the Home page, locate the version of the application you want to use as the source for the new version.
   
   You can create a new version from any version of an application, in any stage of the development lifecycle.

2. Open the Application Options menu for the version and select New Version.
3. Specify the new version number and enter a comment in the New Application Version dialog box. Click **Create**.

You can number your versions according to your versioning scheme, but each version number must be unique.

![New Application Version dialogue box](image)

When you click Create the new version is created and the application opens in the Page Designer. The new version has Development status.

## Import and Export Applications

You can export a visual application as an archive to your local system that you can then import to create a new visual application. You can use the import and export mechanism to share application sources and to move applications between instances.

### Topics:
- Export a Visual Application
- Import a Visual Application
- About Classic Applications
- Import Classic Applications

### Export a Visual Application

You can use the Export action to download an archive of a visual application and its resources to your local system. After exporting the archive, you can import it to create a copy of the application and share the archive with team members.

When you export the application you can choose if you want the archive to include the data contained in the application’s custom business objects. Some information, such as credentials for external REST end points, is removed when you export an application. This information needs to be provided after the archive is imported.
To export a visual application:

1. On the Visual Applications home page, open the Application Options menu for the application version you want to export and select Export.
   
   If there are multiple versions of an application you must use the Options menu of the version that you want to export.

   Alternatively, when a visual application is open, you can choose Export in the application’s options menu in the toolbar.

2. Select the export option for the application.
   
   You can choose to include the data stored in the application’s custom business objects when exporting the application.
When you choose to export the application with data, the archive will include a 
json file (entity.json) and spreadsheet (entity-data.csv) for each custom 
business object. The json file describe the business object and the spreadsheet 
contains the business object data. If you choose to export the application without 
data, the archive will only contain the json file describing the business objects.

The archive will always include the data for any business objects that are identified 
as containing Application Setup Data.

The visual application and its resources are exported as an archive file. The archive is 
saved to your local system in the location specified for your browser's downloads.

Import a Visual Application

Use the Import action to import an archive of a visual application, for example, to 
create a new application from a shared archive.

To use the import tool to create a new visual application, you will need an archive file 
of a visual application that was previously exported.

You can also replace all the source files of an existing visual application with the 
contents of a visual application archive using the Import command in the Options 
menu in the toolbar of the existing application. When you click Import, the Import 
Resources dialog box will open where you can import a ZIP archive. If you 
select Delete existing files and resources in the dialog box all the files and 
resources in the existing visual application will be deleted. The resources are imported 
into the root directory of your visual application. You might want to select this option 
when you want to replace all the existing files with new files to prevent duplicating files.

To import a visual application:

1. Navigate to your Visual Applications home page and click Import.
2. Click **Application from file** in the Import dialog box.

3. Drag your visual application archive file on your local system into the dialog box. Alternatively, click the upload area in the dialog box and use the file browser to locate the archive on your local system.

4. Enter a valid application name and ID in the dialog. Click **Import**.
After you import a visual application, you might need to provide additional details such as service credentials in the new the application. For example, if the application you are importing contains Process definitions, you will be prompted to specify an existing Process application that contains all the process definitions required by the application you are importing.

### About Classic Applications

Classic applications were created in earlier versions of Oracle Visual Builder and used a structure that is not compatible with the visual application structure now used in Oracle Visual Builder. There will be no additional feature development work to support classic applications. If you have any classic applications, you should migrate them to use the visual applications structure if you want to retain the business objects defined in the classic application.

Some older Oracle Visual Builder instances might still allow you to manage classic applications, but newer instances will not provide the tools for viewing and managing classic applications. If your Oracle Visual Builder instance supports classic applications, you can export your classic applications as archives and then import them as visual applications, but the UI of web and mobile apps cannot be migrated. Importing a classic application archive only preserves the details of business objects.
Import Classic Applications

You can use the import tool to create a new visual application that contains a copy of the business objects and data in an archived classic application. Importing a classic application will not recreate the pages in the application, and the credentials and settings used in the application are not imported. To import an archived classic application, the archive must be exported using the most current version of Oracle Visual Builder.

If you have an archived classic application that was exported using an earlier version of Oracle Visual Builder, you might need to import the application as a classic application and export it again as a newer archive before you can import it as a visual application. An archive of a classic application created using earlier versions of Oracle Visual Builder will not be recognized when you try to import it as a visual application. You will see an error message that an archive is “outdated” if you try to import an older archive of a classic application.

You should contact your service administrator if your Oracle Visual Builder instance does not support viewing and managing classic applications and you need to import or export a classic application.

To import an outdated archive of a classic application:
1. Open the Oracle Visual Builder Home page.

2. Open the Application Switcher menu and select **Classic Applications** to switch to the Home page for Classic Applications.

3. On the Classic Applications Home page, click **Import**.
   Use the Import Application dialog box to upload the archive of the application.

4. Upload the classic application archive from your local system.
   You can drag the archive from your local system into the dialog box or click **Upload a file** to navigate to the location of the archive.

5. Type the application name and application ID. Click **Import**.
   The Application Name and ID are automatically populated based on the archive, but you might need to modify the name and ID because they must be unique in your identity domain.

6. Open the Application Options menu of the application you imported and click **Export**.

7. Open the Application Switcher menu and select **Visual Applications** to switch back to the Home page for Visual Applications.
   You can now import the new archive as a visual application.
When you import the new archive of the classic application, the dialog displays a warning message that only the business objects and data in the archive will be imported into the new visual application.

Stage and Publish Visual Applications

You can stage and publish visual applications from the Visual Builder home page.

Topics:

- About Staging and Publishing Applications
- Stage a Visual Application
- Publish a Visual Application
- Build Mobile Applications
- Optimize Your Builds
Note:

Staging a visual application that contains a mobile application requires you to create at least one build configuration for the mobile application. Otherwise, attempts to stage the visual application will fail.

If you build a mobile application in your visual application to create the installation file that installs the mobile application on a device, Oracle Visual Builder deploys your visual application in Development to Stage. To build a mobile application, you create one or more build configurations that sign the mobile application and generate a platform-specific installation file or QR code.

About Staging and Publishing Applications

To stage and publish an app, you deploy the app's resources to the Visual Builder runtime environment that provides services used by the staged and published apps.

The Visual Builder runtime environment provides the server for delivering pages in web applications, and services your web and mobile apps might use to access data, including the database used to store data and the proxy server for managing connections to REST services. The runtime is used when you are designing apps in the Designer and for staged and published applications. The runtime also integrates Oracle Identity Cloud services (IDCS) to manage the authentication and authorization of app users.

The following steps are performed for you when you stage an app:

- The application's resources are copied to a directory on the server
- The database schema in the staging database is updated with changes from the development database
- A URL is created for accessing the staged web app or downloading the mobile app. The web and mobile apps access the services and resources provided by the staged application.

When you stage an app you can choose to copy the data from your development database to the staging database, create a database with no data, or use the data already in the database if it has already been staged.

The following steps are performed for you when you publish an app:

- The directory containing the staged application's resources becomes the live app. The staged app is not accessible after it is published.
- The database schema in the live database is updated with changes from the staging database. You can choose if and how data should be migrated from the staging database to the live database.
- A new URL is created for accessing the live web app or downloading the mobile app the first time you publish the app. For web apps, this is the permanent URL for accessing the app. For mobile apps, you use the URL to download the mobile app, and then publish it to the mobile platform's app store. The end users of your mobile app download the published app from the app store. Web and mobile apps access the services and resources provided by the published app.
If you are staging or publishing a mobile app as a progressive web app (PWA), the Visual Builder runtime serves the app’s pages when a user visits the URL of the staged or published PWA. When you publish a PWA-enabled mobile app, users can also download and install the mobile app directly from the URL in the browser and run it like a native app. This allows you to distribute a mobile app without first publishing it to an app store.

You must stage an app before you can publish it. When an app is published, the staged app becomes the live version, and the app settings defined for the staged app are applied to the published app. You should confirm that an app’s settings, for example, its security settings and credentials, are working correctly before you publish an app, because these cannot be modified after it is published without creating a new version and staging and publishing it again. For example, when you are ready to publish an app, you might need to modify the credentials and authentication mechanism you used for connecting to a service during development because they are not suitable for the published app. In this case, you will need to edit the app to specify the credentials required for the published app and stage it again.

The runtime environment also provides a proxy server that your apps can use to help with authorizing calls to services. For example, you can use the proxy server to avoid potential CORS issues when calling a service. This is convenient if you are sending requests to services in another domain and you cannot modify its whitelist. You can bypass the proxy if you choose, for example, by using the Direct authentication mechanism in your app to call services.

Stage a Visual Application

You can stage unpublished versions of your application at any time from the Home page or from the main menu.

Staging your application enables you to test each update to confirm that it behaves as you expect and that no problems have been introduced, for example, when you add new features or change your data model. You can distribute the URL of the staged application to team members who can help you test and provide feedback. The URL of a staged application is not the same as the URL of the app preview that is opened using the Run button in the Designer. You can’t share the URL of the app preview with other team members.

To stage your application:

1. On the Home page, open the Application Options menu of the application and click **Stage**.
2. In the Stage Application dialog, select a database option. Click **Stage**.

The first time that you stage your application, you need to choose to either start with a clean database for your staged application or copy the data from your development database to the staging database.
After the initial staging, each time that you stage your application you need to specify how you want to manage the data in the staging database. You can choose to keep the data, replace the data with data from the development database, or delete all data in the database and start with a clean database.

On the Home page you can see the status of each version of your application. You can continue to update and stage versions until you are ready to publish. Click **Stage** in the Status column and click the link to open the staged application in your browser.

Alternatively, when a staged visual application is open in the Designer, you can select **Open Stage Application** from the visual application's Menu in the toolbar.

Make a note of the URL of the running staged application. You can share this URL with team members.
Publish a Visual Application

You can publish a staged version of your application from the Home page or from the main menu. After you publish a version of an app it is read-only and can no longer be changed. To make changes to update the app you need to create a new version.

When you publish a staged version of your app it becomes the live version. If you are updating an earlier version of your app, the previous live version is archived and locked. The URI of your app does not change. Only one version of an app can be live at a time, but multiple versions of an app can be staged simultaneously.

The first time that you publish your app you can choose if you want to copy the data from your staging database to the live database or use a clean database. When you update your app to a new version you will be prompted to decide how you want to manage the data in your live database.

After an app is published, the resources used by the app (for example, metadata, images, stylesheets) will not change until you publish a new version. The resources of the published app are cached on the client and are fetched from the local cache instead of retrieved again from the server. An app’s cached resources are replaced when the version of the app you retrieve is newer than the cached version.

---

**Note:**

Oracle recommends that you regularly re-stage and republish applications as new versions of Visual Builder are released. This ensures that your applications use the latest bug and security fixes available in the platform. From time to time, Oracle may issue notifications reminding you of this best practice, especially if a new release contains particularly critical fixes.

---

To publish your application:

1. On the Home page, open the Options menu of the application and choose **Publish**.
2. In the Publish Application dialog box, select an option for handling the business object data.

3. Click **Publish** to move the staged version of your application to the live server.

You can run the published app in your browser by clicking **Live** in the Status column and selecting the link. You can share the URL of the running app with public users.
Build Mobile Applications

Before you can stage or publish a mobile application in your visual application, you define a build configuration that describes how to build the mobile application for the platform where the application will be installed (Android and/or iOS).

Once you have defined the build configuration for the platform and reviewed the property values in the mobile application’s settings, you can build the mobile application for the corresponding platform. At the end of the build process, Oracle Visual Builder generates a QR code and a link to an installation file. You or other users can scan the QR code to install the mobile application or, alternatively, you can download the installation file from the provided link and install it on the device.

Oracle Visual Builder provides ready-to-use image files for the Android and iOS platforms that render the application launcher icon and splash screen for the mobile application on the device. You can replace these ready-to-use image files with custom image files that conform to the look and feel you want your mobile application to render. Similarly, Oracle Visual Builder provides ready-use permission strings for the mobile applications that it builds for the iOS platform. You can also replace these permission strings with custom permission strings that you define.

- Change the Splash Screen and Icons Used by a Mobile Application
- Provide Custom Permission Strings for Mobile Applications on iOS
- Define a Build Configuration for the Android Platform
- Define a Build Configuration for the iOS Platform
- Configure Mobile Application Settings
- Build a Mobile Application

Change the Splash Screen and Icons Used by a Mobile Application

Oracle Visual Builder provides images that your mobile application uses to render a splash screen and app launcher icon when the application runs on the Android and iOS platforms. Replace the Oracle-provided images with your own images so that your mobile application uses a splash screen and icons that conform to your brand.

To replace the Oracle-provided images that your mobile application uses by default, you upload an application image archive through the dialog that is exposed in the Resources section of the mobile application’s Settings page, as shown in the following image.
Oracle Visual Builder also provides sample application image archive files that you can download to review and determine what type of images you need to provide in the archive that you upload to Oracle Visual Builder to replace the default images used by your mobile application. Each sample provides a complete list of the images that you need to upload to replace the default images provided by Oracle. The file names and, in the case of the Android sample, the directory names describe the purpose and type of image. In the iOS sample, the use of \textit{ipad} in a file name indicates that the image file is intended for use on iPad devices while \textit{iTunesArtwork.png} indicates that the image file is used to render the artwork for the mobile application’s entry in the Apple App Store.

The Android sample includes more images so that icons are provided to display properly on Android-powered devices of different sizes and resolutions that require low, medium, high, extra-high, extra-extra-high density, and extra-extra-extra-high density versions of the same images. The name of each directory and image file within the downloaded sample indicates its purpose. For example, the \texttt{drawable-land-xxhdpi-v4} directory contains images for Landscape mode with a requirement for extra-extra-high density resolution. Similarly, the \texttt{drawable-port-xxhdpi-v4} directory provides images for when the app is in Portrait mode. The \texttt{9} within file names, such as \texttt{screen.9.png} indicates that the image is a NinePatch image that Android resizes to accommodate the contents of a view, as discussed in Android's documentation.

The archive that you upload should have the same number and type of images to those provided in the Android and iOS samples that you can download from the respective links in the Resources section. If, for example, you upload an application image archive for iOS, ensure that it includes an alternative image for each of the images listed in the iOS sample, as in the following two examples, that all images reside within the root directory of the zip file that you upload, and that the image you use for the splash screen is an Xcode storyboard.

\texttt{Default@2x-ipad-anyany.png}
\texttt{...}
\texttt{iTunesArtwork.png}

Also, for iOS, ensure that you do not use the Compress option on your Mac when you create the application image archive. Creating an archive with the Compress option results in an archive with an invalid structure that Oracle Visual Builder will reject. Instead, you should do the following:
1. From the command prompt, navigate to the directory that contains the custom images you want to include in the application image archive.

2. Run the following command to create an application image archive with the correct structure.
   ```
   zip -r ../<your zip file name>.zip
   ```
   Refer to the Android and iOS documentation for guidelines on creating icons and splash screens. Apple, for example, provides guidelines for the splash screen (launch screen) and the app icon.
   
   Irrespective of the name that you give to the archive file that you upload, Oracle Visual Builder renames it to `iOSResourcesImages.zip` in the case of iOS and `AndroidResourcesImages.zip` in the case of Android.

Provide Custom Permission Strings for Mobile Applications on iOS

Mobile applications on iOS require usage descriptions (permission strings) if the application uses hardware capabilities, such as the device camera, that allow it to access end user data.

The iOS device displays these permission strings in the system dialog that requests an end user to allow access to the hardware capability and/or potentially sensitive user data, as in the following example, where the "This application allows the user to upload photos from their photo library" permission string appears in the dialog when the application requests access to the photo library.

"DeviceDemo" Would Like to Access Your Photos

This application allows the user to upload photos from their photo library.

<table>
<thead>
<tr>
<th>Don’t Allow</th>
<th>OK</th>
</tr>
</thead>
</table>

Oracle Visual Builder provides permission strings for each mobile application that it builds for iOS. Mobile applications for Android do not use these permission strings. You can provide alternative permission strings to those provided by Oracle Visual Builder by uploading a ZIP file that contains the permission strings you want your mobile application to use. Upload the ZIP file through the dialog that is exposed in the Resources section of the mobile application's Settings page. Download the sample from the Permission Strings Sample link to use as a reference when creating your ZIP file with the permission strings you want to use. The downloaded sample ZIP file contains an `ios_permissions.properties` file with a list of the permission strings that you can modify, such as the following:

```properties
NSCameraUsageDescription=The camera may be used to take pictures that may be uploaded to a server.
```
Replace the string to the right of \( = \) with the permission string you want to use and upload a ZIP file containing the modified `ios_permissions.properties` file. Irrespective of the name that you give to the ZIP file that you upload, Oracle Visual Builder renames it to `iosPermissionStrings.zip`.

For information about the Apple Cocoa keys, such as `NSCameraUsageDescription`, see Apple’s documentation.

**Define a Build Configuration for the Android Platform**

Define one or more build configurations for the Android platform to test and deploy your mobile applications.

Before you define a build configuration, make sure you have access to a keystore and its access credentials. The build configuration that you define uses the keystore to sign the mobile application when it builds it. For information about creating a keystore and using it to sign an app for the Android platform, see Sign Your App in Android Studio’s documentation.

1. Click the **Mobile Applications** tab.

2. Click the `<app name>` node and click the Settings icon (🔧).
   The General Settings tab is displayed.

3. Select the **Build Configurations** tab and click **Android** in the New Configuration list.
The Create New Android Build Configuration dialog box is displayed.

4. Enter a name for the configuration in the Configuration Name field.

5. Select the build type from the Build Type list. Possible options include Debug or Release.
   Select Debug to build a mobile application that is in development and you want to stage to share with other testers. Select Release when you have completed development of your mobile application and want to publish it for download by end users.

6. Enter a unique ID for the application in the App ID field. Each application deployed to an Android device has a unique ID, one that cannot start with a numeric value or contain spaces.

7. Specify the release version number for the application in the Version Name field. This is the release version of the application code that is displayed to the user. For example, enter 2.0 if this is the second version of your application. The value you enter appears in application information dialogs when you deploy the application to a device.

8. Enter an integer value that represents the version of the application code in the Version Code field. This version code is checked programmatically by other applications for upgrades or downgrades. The minimum and default value is 1. You can select any value and increment it by 1 for each successive release.

9. Drag and drop, or browse to and retrieve, the directory of the keystore containing the private key used for signing the application for distribution in the Keystore field.

10. Enter the password for the keystore in the Keystore Password field. This password allows access to the physical file.

11. Enter an alias for the key in the Key Alias field. This is the value set for the keytool’s -alias argument. Only the first eight characters of the alias are used.

12. Enter the password for the key in the Key Password field. This password allows access to the key (identified by the alias) within the keystore.

13. Select the Default Configuration for Stage checkbox, Default Configuration for Publish checkbox, or both to specify how to use the build configuration when you stage or publish an application.
   Oracle recommends that you define separate default build configurations for Stage and Publish.
14. Optionally, review the custom application template that is automatically selected. The field is displayed only if a custom application template is defined for the selected build type. You can define one application build template each for the Debug and Release build types.

15. Click **Create Configuration**.

The new build configuration is displayed on the Build Configurations page.

Define a Build Configuration for the iOS Platform

Define one or more build configurations for the iOS platform to test and deploy your mobile applications.

Before you define a build configuration, you must create a provisioning profile. To create a provisioning profile, you require membership of the iOS Developer Program or the iOS Developer Enterprise Program. A provisioning profile associates development certificates, devices, and an application ID. The build configuration you create uses the provisioning profile that you specify to build the mobile application for the iOS platform. For more information about iOS developer programs and provisioning profiles, review the App Distribution Guide available through the iOS Developer Library.

1. Click the **Mobile Applications** tab.

2. Click the `<app name>` node and click the Settings icon (**`).

   The General Settings tab is displayed.

3. Select the **Build Configurations** tab and click **iOS** in the New Configuration list.

   The Create New iOS Build Configuration dialog box is displayed.
4. Enter a name for the configuration in the Configuration Name field.

5. Select the build type from the Build Type list. Possible options include Release and Debug.
   Select Debug to build a mobile application that is in development and you want to stage to share with other testers. Select Release when you have completed development of your mobile application and want to publish it for download by end users.

6. Enter a bundle ID in the Bundle ID field or accept the default value generated by Oracle Visual Builder.
   The bundle ID must be unique for each application installed on an iOS device. The application ID must adhere to the format set in the iOS Provisioning Portal and cannot contain any spaces. For more information, see the documentation available through the iOS Developer Library.

7. Specify the version name in the Bundle Version Name field.
   The release version number for the application. This version number is displayed to the end users and identifies a released iteration of the application. It is a string made up of three non-negative and period-separated integers, such as 3.1.2. The string should only contain numeric (0-9) and period (.) characters.

8. Enter the version in the Bundle Version field.
   The version you enter corresponds to the build number that identifies an iteration (released or unreleased) of the application and must be incremented each time the application is uploaded to the Apple App Store. The build version is typically a string made up of three non-negative and period-separated integers, such as 3.1.2. The string should only contain numeric (0-9) and period (.) characters.
   • The first number represents the most recent major release and must be greater than zero. The integer is limited to a maximum length of four digits.
   • The second number represents the most recent significant revision and is limited to a maximum length of two digits.
   • The third number represents the most recent minor bug fix and is limited to a maximum length of two digits. If the value of the third number is 0, you can omit it and the second period.

9. Drag and drop, or browse to and retrieve a certificate in the Certificate field.
   Ensure you use the certificate associated with the provisioning profile you intend to use (Step 11) for this build configuration. The certificate file must be a P12 file (.CER format is not supported), for example, ent2_2018.p12. The iOS Development Certificate electronically associates a developer’s identity with a public key and private key. The certificate identifies you as an iOS developer and enables the signing of the application for deployment. In the iOS operating environment, all certificates are managed by the Keychain Access app.

10. In the Certificate Password field, enter the certificate password that was set to protect the provisioning profile certificate when it was exported from the Keychain Access app.

11. Drag and drop, or browse to and specify the location of the provisioning profile in the Provisioning Profile field.

12. In the Signing Identity field, enter the name of the developer or distribution certificate that identifies the originator of the code (such as a developer or a
company). Enter the entire name as it appears in the Common Name column of the Keychain Access app.

You use a signing identity to code sign your application. A certificate and its public key are stored in the Member Center, and the corresponding signing identity (the certificate with its public and private key) is stored in your keychain. You will not be able to code sign without this private key. Here is how to identify the value to provide in the Signing Identity field.

13. Select the Default Configuration for Stage checkbox, Default Configuration for Publish checkbox, or both to specify how to use the build configuration when you stage or publish an application.

Oracle recommends that you define separate default build configurations for Stage and Publish.
14. Optionally, review the custom application template that is automatically selected. The field is displayed only if a custom application template is defined for the selected build type. You can define one application build template each for the Debug and Release build types.

15. Click **Create Configuration**. The new build configuration is displayed on the Build Configurations page.

**Configure Mobile Application Settings**

After creating your mobile application, review the settings for your application to verify that it has appropriate values for properties like App Name and Lock Portrait Mode.

1. Click the **Mobile Applications** tab.

2. Click the `<app name>` node and click the Settings icon (🔧). The General Settings tab is displayed.

3. In the Application Settings tab, select the main or starting page for your mobile app from the Default Page list. This page is displayed when you open the app.

4. Select a value to specify the theme, if any, to use for the mobile app.

5. In the App Name field, enter the app name that is to be displayed when the app is installed on a mobile device.

6. Enter the name of the vendor who originated the application in the Vendor Name field.

7. Enter text that describes the application in the Description field.

8. Optionally, you can use an earlier released version of Visual Builder Runtime, JET, or both to build your application.

   For more information, see #unique_55/unique_55_Connect_42_GUID-71474A5E-AA15-4F77-BFCE-67E07282026E.

9. Enter the URL scheme for the app in the URL Scheme field.

10. Specify the package name for the app in the Package Name/Bundle ID Default field.
To avoid naming conflicts, Android and iOS use reverse package names, such as com.company.application.

11. Clear the **Lock Portrait Mode** checkbox if you want the mobile application to render in both Landscape and Portrait mode on the user's device. By default, the application renders in Portrait mode only.

12. Select the **Support iPad Deployment** checkbox if you want to deploy the mobile application to iPad.
Build a Mobile Application

Build the mobile application to generate a QR code and installation file for your application.

Before you build your mobile application, make sure you:

• Review the settings for your mobile application, as described in Configure Mobile Application Settings.

• Define the build configuration for the platform where you want to install the mobile application as described in Define a Build Configuration for the Android Platform and Define a Build Configuration for the iOS Platform.

1. Open the mobile application that you want to build.

2. Click the Run icon ( ) to run the app on a new tab in the browser.

   The mobile application is displayed in the browser. If your application is configured to run as PWA, both the Native app and PWA tabs are displayed.
3. Click the **Build my App** button in the **Native app** or **PWA** tab to build a native or PWA app, respectively.

4. In the Stage Application dialog box, select the appropriate option to determine what to do with business object and then click **Stage**.

   ![Stage Application dialog box]

   When the build is complete, the generated QR codes are displayed.

5. Perform one of the following.
   - For a native app, scan the QR code to download the installation file onto the mobile device or use the download link to download the installation file (APK or IPA) to your file system.
For a PWA app, scan the QR code to download the file onto the mobile device or click the **Launch in Browser** link to view the app in the browser.
Enable Progressive Web App Support from Mobile Applications

Mobile applications that you develop in Oracle Visual Builder can be distributed as Progressive Web Apps (PWA) if you enable PWA support for the application.

When a mobile app with PWA support enabled is deployed using Oracle Visual Builder, the application runs as a web app and not as a native mobile app. A PWA app created in Visual Builder:

- Allows you to install the application from the browser window and add the application to the Home screen
  As a developer, you don’t need to publish the app to an app store. Instead, you can distribute a URL which prompts the user with the \textit{Add <PWA appName> to Home screen} message (in Chrome on Android).

- Runs on your device like a native app
  Similar to a native app, no address bar is displayed when a PWA app runs.

- Offers the user experience of a native app
  PWA apps deliver user experiences through progressive enhancement. This refers to the ability to gradually add more features to a PWA app to make it closer to a native app.
native app (as browser support becomes available). For more information on PWAs, see the documentation.

- Works offline similar to a native app
  If you implement data caching using the offline toolkit, PWA apps work offline similar to native apps.

Here is what a PWA app looks like when running in the browser.

Here is what the same PWA app looks like when running on a device.
When developing PWA apps, select the Oracle Cloud option in the Authentication Mechanism list in the Security tab for the mobile application.

1. Click the Mobile Applications tab.

2. Click the `<app name>` node and click the Settings icon (⚙️). The General Settings tab is displayed.

3. Select the PWA tab.

4. Click Enable Progressive Web App (PWA).
The Manifest Settings, Resources, and Advanced File Caching panes are displayed.

5. Review and edit, if needed, the information displayed in the Manifest Settings pane.

These settings are included in the web manifest file (a JSON configuration file) that contains information about your application. When an application is configured as PWA, a Web Manifest is added to the application with content required by PWA. For more information on this file, review this page.

a. Specify the Application Name.

The specified name appears on the dialog that displays prompts you to install the application.

b. Review the short name of the application.

If both application name and short name are specified, the short name is used on the Home screen (below the app icon), launcher, and other places where space is limited.

c. Change the background color and theme color, if needed, by clicking the currently selected color and choosing a new color in the color picker or by entering a RGB value in the text field.

6. Customize the splash screen and icons to use for your app in the Resources pane.

By default, Oracle-provided images are used for your app. Replace these with images to use splash screen and icons that conform to your brand. Upload an application image archive either by dragging it or by using the Open dialog box.

7. Specify the files to cache on the user’s device in the Advanced File Caching pane.

By default, when launched for the first time, the PWA app caches all flows and pages on the user’s device. Use this pane to narrow down the required resources to be stored in the browser cache.

After you successfully build your PWA enabled mobile app, scan the generated QR Code for the PWA app from a QR Code scanner or camera app installed on your Android device.

Click the Launch in Browser link to open the PWA in the browser. After you launch the PWA application in a browser on a mobile device, you can add it to the Home screen.

• On the iOS platform, select the Add to Home Screen option.
• On the Android platform, when PWA applications are deployed on Android, a new install application dialog is displayed to allow users to install the application on the Home screen. The dialog displays when the application receives the new vbBeforeAppInstallPrompt lifecycle event.

When you enable PWA for an application, a sample implementation is added to the application start page for the UI and event handling. We recommend that you customize the UI and handling of the vbBeforeAppInstallPrompt event to meet your needs.

Guidelines for Using PWA Support

Here are a few things to consider when using PWA support for your applications.

• At present, you can only enable mobile applications that you created in Oracle Visual Builder as PWAs.

• Currently, we support the Chrome browser for Android and the Safari browser on iOS. We recommend that you use the latest available browser versions. For information on supported browser versions, review this page.

• Here are few limitations for PWA apps running on the iOS platform. These issues may be resolved in future iOS releases.
  – Install the PWA app by using the share icon (because the Add <PWAappName> to Home screen message is not displayed).
  – PWA app state is not saved between sessions. If a user exits a PWA, the app is restarted when the user returns.
  – Navigation between screens in an app is possible only by using the built-in navigation. This is because Apple devices do not have a Back button.
  – Inactive apps appear as a white screen (no splash screen support) in the task manager.
  – Service worker cache size is limited to 50 MiB per partition.
  – Caches for unused apps are purged frequently.
  – Web manifest is not supported on Safari.
  – Orientation lock not supported.

• Periodically delete service workers and clear cache when developing PWAs. When you repeatedly stage new versions of a PWA (in the iterative development cycle), you might run into issues with Chrome DevTools if multiple service workers are present. Here are high-level steps to do this:
  1. Click Cmd+Option+I (on Macintosh) or Ctrl+Shift+I (on Windows) to open the Chrome DevTools.
  2. Switch to the Application tab.
  3. Click Clear Storage in the left menu.
4. Click **Clear Site Data** to clear the cache and unregister service workers.

# Optimize Your Builds

You can optimize the performance of your published web apps by using node and Grunt to build them locally or to set up build jobs on Developer Cloud Service.

**Topics:**
- About Building Your Application With Grunt
- Build Your Application Using Developer Cloud Service
- Build Your Application Locally
- Visual Builder Grunt Tasks
- Customize Your Grunt Build Process

## About Building Your Application With Grunt

You can customize the build process of your applications by using Grunt instead of the Stage and Publish actions in Visual Builder.

You can use Grunt to build your application from sources stored in a Git repository on Developer Cloud Service or stored on your local system. Your visual application includes a Grunt file that you can modify to define custom tasks that you want to include in the build process.

Your application includes two resource files in the root folder of your application that are used when building the application using Grunt: **Gruntfile.js** and **package.json**.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package.json</td>
<td>This file declares the dependency and specifies the URL reference to the grunt-vb-build NPM package that provides the Grunt tasks used to build visual applications. Visual Builder automatically updates the package version and URL when Oracle publishes a new version of the package.</td>
</tr>
<tr>
<td>Gruntfile.js</td>
<td>This file contains a basic Grunt script for building the app that can be modified to add custom build tasks. The tasks are loaded by calling load-grunt-tasks.</td>
</tr>
</tbody>
</table>

The build folder contains the following artifacts that are generated during the build process:

- **optimized folder** — optimized application resources, generated by the **vb-optimize task**
- **optimized.zip** — archive of optimized resources that are uploaded by the **vb-stage task**
- **processed.zip** — archive obtained from Visual Builder server containing processed local sources, generated by the **vb-process task**
to-be-processed.zip — archive of local sources that are uploaded to Visual Builder server for processing

The build process for an application using Grunt includes the following steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process the application sources</strong></td>
<td>This step consists of several important processes. The most important is “metadata processing”, when the visual application sources are transformed into a deployable form. This includes the injection of Visual Builder runtime links and other configurations into the application’s index.html, processing other application templates, and the creation of service definition files. You run the <code>vb-process</code> task to process the sources. This task creates a ZIP archive of the local application sources (to-be-processed.zip), uploads this to the Visual Builder instance that you specify, then downloads an archive of the processed application sources (processed.zip) and expands the archive into the ./build/processed folder. The processed application sources in this folder are the input for the optimizing sources step.</td>
</tr>
<tr>
<td><strong>Optimize the processed sources</strong></td>
<td>This step consists of three parts: optimize images, optimize styles, and create require module bundles. The optimization step compresses the images in the application, minifies the stylesheets, and organizes and distributes the resources in the module bundles to optimize application performance based on the application's structure and how you want it to load. You run the <code>vb-optimize</code> task to optimize the processed sources. The <code>vb-optimize</code> task runs the <code>vb-image-minify</code>, <code>vb-css-minify</code> and <code>vb-require-bundle</code> tasks, and generates the ./build/processed folder containing the optimized sources.</td>
</tr>
<tr>
<td><strong>Stage the application artifact</strong></td>
<td>This step consists of creating the deployment package archive and deploying it to the Visual Builder instance. You run the <code>vb-stage</code> task to deploy the application artifact. If the application uses Visual Builder business objects, the staging step also involves processing the corresponding database schemas.</td>
</tr>
</tbody>
</table>

You can include custom tasks in any of the steps by modifying `Gruntfile.js` to redefine tasks.

Build Your Application Using Developer Cloud Service

When your sources are stored in a Git repository on Developer Cloud Service, you can configure a continuous integration and delivery pipeline (CI/CD) by setting up jobs in Developer Cloud Service to run build tasks for you.

**Note:**

To build your application, the build VM must have node8 or higher enabled in the software section.

To create a job in Developer Cloud Service:

1. Log in to your Developer Cloud Service instance.
2. Click **Builds** in the Navigation bar, and **New Job** in the Jobs tab.

3. In the New Job dialog box, enter the project name and description.

4. Select **Basic Build VM Template** in the Software Template dropdown list. Click **Create Job** to open the Job Configuration page.

   If Basic Build VM Template is not available in the dropdown list, you will need to find or create an instance that has basic tools and Nodejs version 8 or newer.

5. In the Source Control tab, click **Add Source Control** and choose Git in the dropdown list.

6. Select your application’s repository and choose the **master** branch.

7. Select **Automatically perform build at SCM change** if you want a build to be triggered each time you push changes in to the repository from Visual Builder.

8. Add the following parameters in the Build Parameters tab:

   In the Build Parameters tab, use the Add Build Parameter button to add and define the parameters you need for the build configuration.

   ![Configure Build Parameters](image)

   The parameters are entered using the following format: *(name/type/initial value)*

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceURL</td>
<td>string</td>
<td>Your Visual Builder instance URL</td>
</tr>
<tr>
<td>id</td>
<td>string</td>
<td>The id of your visual application</td>
</tr>
<tr>
<td>ver</td>
<td>string</td>
<td>The version of your visual application</td>
</tr>
<tr>
<td>username</td>
<td>string</td>
<td>Your username for your Visual Builder instance</td>
</tr>
<tr>
<td>password</td>
<td>password</td>
<td>Your password for your Visual Builder instance</td>
</tr>
<tr>
<td>schema</td>
<td>string</td>
<td>The data schema for processing during application stage or publish. The value can be new, dev, stage or live.</td>
</tr>
</tbody>
</table>

9. In the Steps tab, click **Add Step** and select **Unix Shell**.
You use the Unix Shell text area to enter options for the `vb-build` task and assign the build parameters storing the values for the options.

10. Type the following in the Script text area.

```bash
npm install
./node_modules/.bin/grunt vb-build \
--url=${serviceURL} \
--username=${username} \
--password=${password} \
--id=${id} \
--ver=${ver} \
--schema=${schema}
```

11. Click **Save**.

After you create and configure the build job, you can click **Build Now** when you want to manually start the job.

**Build Your Application Locally**

After you create your visual application in Visual Builder and you are ready to stage or publish it, you can download the sources to your local machine and use Grunt to build it locally, without setting up a Developer Cloud Service pipeline for building the application.

To build an application locally you need to confirm that node (npm) and the dependencies are installed and that the sources for the application are saved on your local system. You can get the sources of your visual application in the following ways:

- Cloning the Git repository containing the sources
- Exporting the visual application from Visual Builder and extracting them to your local system
- Using the `vb-export` Grunt task to retrieve the sources from Visual Builder
Get Sources From Visual Builder

You can use the `vb-export` task to retrieve the sources of your visual application from Visual Builder and save them to a folder on your local system.

If your visual application sources are not stored in a Git repository, or you cannot get your sources using Git, you can get the sources by exporting the visual application directly from Visual Builder using the Export command, or by using the `vb-export` Grunt task.

To use the `vb-export` Grunt task, you must have node installed on your local system and you must have a copy of a `package.json` file and `Gruntfile.js` in a folder on your local system. You can get a copy of those files from a Visual Builder application. To get the sources of a visual application, you must also have the details of the visual application that you want, including the URL of the Visual Builder instance, the credentials for accessing the instance, and the id and version of the visual application that you want to retrieve.

To get visual application sources from your Visual Builder instance using Grunt:

1. Open a system terminal on your local machine and type `npm -v` to confirm that node is installed.
2. In the terminal, navigate to the folder on your local system containing the `package.json` and `Gruntfile.js` files.
3. Type `npm install` in the terminal to retrieve the libraries and `grunt-vb-build` NPM package.
   
   The location of the libraries and `grunt-vb-build` package is specified in `package.json`.
4. In your terminal, run the `vb-export` task, supplying the task options.
   
   You need to supply the options required to access and specify the visual application. You will probably want to also use the `--git-sources` option to specify the location on your local system where you want to store the sources. If you do not specify a location, the sources will be downloaded to the current directory, which could overwrite files in the folder. The commands you type in the terminal to run the `vb-export` will be similar to the following:

   ```bash
   ./node_modules/.bin/grunt vb-export \
   --url=<url of visual builder instance> \ 
   --username=<username> \ 
   --password=<password> \ 
   --id=<your visual app ID> \ 
   --ver=<your visual app version> \ 
   --git-sources=<local directory for sources>
   ```

After getting the sources from Visual Builder, you can use the `vb-build` task to build the application.

Build Your Application

You can use the `vb-build` task to process, optimize and stage your application.

To build your application locally:
1. Open a system terminal on your local machine and type `npm -v` to confirm that node is installed.

2. In the terminal, navigate to the folder on your local system containing the `package.json` and `Gruntfile.js` files.

3. Type `npm install` to retrieve the node dependencies required to build the application.

   The install command retrieves the `grunt-vb-build` npm package defined in `package.json`.

4. Type the following in the terminal:

   ```
   ./node_modules/.bin/grunt vb-build \
   --url=<url of visual builder instance> \
   --username=<username> \
   --password=<password> \
   --id=<your visual app ID> \
   --ver=<your visual app version> \
   --git-sources=<local directory for sources>
   ```

   When you run `vb-build` and other tasks, you will need to include the `--git-sources` option to specify the location of the sources if they are not located in your current folder.

When the grunt build finishes your application is staged and you can test it by opening the URL of the staged application in your browser. You can quickly open the staged application from the Visual Builder Home page.

Publish the Application

After you have staged your application, you can use the `vb-publish` task to publish the application, and use the `--schema` option to define how the app's data schema is processed.

When staging or publishing an application, you use the `--schema` option to specify the database schema that will be used. The `--schema` option must be set to one of the following values: `new`, `dev`, `stage`, or `live`.

To publish an application using `vb-publish`:

1. Open a terminal on your local system and navigate to the folder containing the visual application sources.

2. Run the `vb-publish` task, supplying the details of the target Visual Builder instance.

   In addition to the instance URL, credentials and details of the app, you will want to use the `--schema` option to identify which database schema the published app should use. If you are using the schema containing the live data, the command you type in the terminal will be similar to the following:

   ```
   ./node_modules/.bin/grunt vb-publish \
   --url=<url of visual builder instance> \
   --username=<username> \
   --password=<password> \
   --id=<your visual app ID> \
   ```
Visual Builder Grunt Tasks

The build tasks exposed by the grunt-vb-build package have options and pre- and post- task hooks that you use when defining the task.

Topics:

• About Visual Builder Grunt Build Tasks
• vb-build
• vb-clean
• vb-process
• vb-stage
• vb-publish
• vb-optimize
• vb-prepare
• vb-require-bundle
• vb-css-minify
• vb-image-minify
• vb-json-minify
• vb-export
• vb-manifest
• vb-process-raw
• vb-process-raw-index-html

About Visual Builder Grunt Build Tasks

You use pre-defined Grunt tasks and options to define the tasks that are performed when building your application.

The public Grunt tasks that are used to build the application are exposed by the grunt-vb-build package. Some tasks have pre- and post- task hooks which you can use to add custom functionality.

Options that you specify for a task might be overridden by other options that have a higher priority. Task options have the following priority:

1. Command line options
2. Task:target specific options (for multitasks)
3. Task specific options
4. Generic “vb” options
vb-build

The main task that does a full application build, including deployment.

The following table describes the subtasks, hooks and inputs and outputs of the `vb-build` task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>vb-clean, vb-process, vb-optimize, vb-manifest, vb-stage</td>
</tr>
<tr>
<td>multitask</td>
<td>n/a</td>
</tr>
<tr>
<td>hooks</td>
<td>vb-pre-build, vb-post-build</td>
</tr>
<tr>
<td>input</td>
<td>n/a</td>
</tr>
<tr>
<td>output</td>
<td>build/to-be-processed.zip, build/processed/<em>, build/processed.zip, build/optimized/</em>, build/optimized.zip</td>
</tr>
</tbody>
</table>

This task does not define its own options. Any option of the subtasks is relevant.

vb-clean

This task cleans the build directory.

The following table describes the subtasks, hooks and inputs and outputs of the `vb-clean` task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>n/a</td>
</tr>
<tr>
<td>hooks</td>
<td>vb-pre-clean, vb-post-clean</td>
</tr>
<tr>
<td>input</td>
<td>build/*</td>
</tr>
<tr>
<td>output</td>
<td>n/a</td>
</tr>
</tbody>
</table>

The following table describes the options for the `vb-clean` task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build options</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
</tbody>
</table>

vb-process

This task processes the local application sources by sending them to the Visual Builder instance.

The processing operation resolves various templates and further modifies (adds and updates) several application resources. The task makes a ZIP archive of the local application sources, sends the archive to the Visual Builder processing service and obtains an archive of processed sources. This archive is then expanded on the local filesystem and used for other tasks.
To run this task you must provide the Visual Builder URL and your credentials for accessing the Visual Builder instance. You can provide the credentials directly when you run the task. Alternatively, you can use the accessToken option to specify a valid OAuth access token.

The following table describes the subtasks, hooks and inputs and outputs of the **vb-process task**.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>n/a</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td>${gitSources}</td>
</tr>
<tr>
<td>output</td>
<td>build/to-be-processed.zip, build/processed/*, build/processed.zip</td>
</tr>
</tbody>
</table>

The following table describes the options for the **vb-process task**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>yes</td>
<td>n/a</td>
<td>ID of visual application you’re going to build. The application needs to exist on the referred Visual Builder instance.</td>
</tr>
<tr>
<td>ver</td>
<td>yes</td>
<td>n/a</td>
<td>Version of the visual application.</td>
</tr>
<tr>
<td>url</td>
<td>yes</td>
<td>n/a</td>
<td>Your Visual Builder instance URL</td>
</tr>
<tr>
<td>username</td>
<td>no</td>
<td>n/a</td>
<td>The username to be used to obtain OAuth access token for further communication with Visual Builder services.</td>
</tr>
<tr>
<td>password</td>
<td>no</td>
<td>n/a</td>
<td>The password to be used to obtain OAuth access token for further communication with Visual Builder services.</td>
</tr>
<tr>
<td>accessToken</td>
<td>no</td>
<td>n/a</td>
<td>The value of an OAuth access token. If provided, username and password options are necessary.</td>
</tr>
<tr>
<td>sslCertificate</td>
<td>no</td>
<td>n/a</td>
<td>The path to the SSL certificate for the connection to Visual Builder instances provisioned with self-signed certificates.</td>
</tr>
</tbody>
</table>

**vb-stage**

This task stages (deploys) the visual application artifact to the Visual Builder instance.

The following table describes the subtasks, hooks and inputs and outputs of the **vb-stage task**.
The following table describes the options for the vb-stage task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application selection options</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>yes</td>
<td>n/a</td>
<td>ID of visual application you’re going to build. The application needs to exist on the referred VB instance.</td>
</tr>
<tr>
<td>ver</td>
<td>yes</td>
<td>n/a</td>
<td>Version of the visual application.</td>
</tr>
<tr>
<td>url</td>
<td>yes</td>
<td>n/a</td>
<td>Your Visual Builder instance URL</td>
</tr>
<tr>
<td><strong>Authentication options</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>no</td>
<td>n/a</td>
<td>The username to be used to obtain OAuth access token for further communication with Visual Builder services</td>
</tr>
<tr>
<td>password</td>
<td>no</td>
<td>n/a</td>
<td>The password to be used to obtain OAuth access token for further communication with Visual Builder services.</td>
</tr>
<tr>
<td>accessToken</td>
<td>no</td>
<td>n/a</td>
<td>The value of an OAuth access token. If provided, username and password options are necessary.</td>
</tr>
<tr>
<td>sslCertificate</td>
<td>no</td>
<td>n/a</td>
<td>The path to the SSL certificate for the connection to Visual Builder instances provisioned with self-signed certificates.</td>
</tr>
<tr>
<td><strong>Build options</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
<tr>
<td><strong>Data processing options</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>schema</td>
<td>no</td>
<td>new</td>
<td>Specifies data schema processing during application stage or publish. The value can be new, dev, stage or live.</td>
</tr>
</tbody>
</table>

This task publishes a previously staged application to Live mode.

The following table describes the subtasks, hooks and inputs and outputs of the vb-publish task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>n/a</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td>n/a</td>
</tr>
</tbody>
</table>
The following table describes the options for the `vb-publish` task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>yes</td>
<td>n/a</td>
<td>ID of visual application you’re going to build. The application needs to exist on the referred Visual Builder instance.</td>
</tr>
<tr>
<td>ver</td>
<td>yes</td>
<td>n/a</td>
<td>Version of the visual application.</td>
</tr>
<tr>
<td>url</td>
<td>yes</td>
<td>n/a</td>
<td>Your Visual Builder instance URL</td>
</tr>
</tbody>
</table>

Authentication options

| username           | no        | n/a           | The username to be used to obtain OAuth access token for further communication with Visual Builder services. |
| password           | no        | n/a           | The password to be used to obtain OAuth access token for further communication with Visual Builder services. |
| accessToken        | no        | n/a           | The value of OAuth access token. If provided, username and password options are necessary. |
| sslCertificate     | no        | n/a           | The path to the SSL certificate for the connection to Visual Builder instances provisioned with self-signed certificates. |

Data processing options

| schema             | no        | new           | Specifies data schema processing during application stage or publish. The value can be `new`, `dev`, `stage` or `live`. |

This task optimizes the application sources by doing the following: minifies images and CSS, and creates minified requirejs module bundles.

The following table describes the subtasks, hooks and inputs and outputs of the `vb-optimize` task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td><code>vb-prepare, vb-image-minify, vb-css-minify, vb-require-bundle</code></td>
</tr>
<tr>
<td>multitask</td>
<td>n/a</td>
</tr>
<tr>
<td>hooks</td>
<td><code>vb-pre-optimize, vb-post-optimize</code></td>
</tr>
<tr>
<td>input</td>
<td><code>build/processed/*</code></td>
</tr>
<tr>
<td>output</td>
<td><code>build/optimized/*</code></td>
</tr>
</tbody>
</table>

This task does not define its own options. Any option of the subtasks is relevant.
vb-prepare

This task copies either the raw application sources or the processed application sources to the build/optimized directory where the optimization takes place.

If the build/processed directory exists (as a result of the vb-process task) and no rawSourcesMode option is specified, this task will copy the build/processed directory into the build/optimized directory.

If the build/processed directory does NOT exist or rawSourcesMode option is set to "True", this task will copy the raw application sources (specified by the gitSources option, the default values of which is "/") into the build/optimized directory and invoke the vb-process-raw task.

The following table describes the subtasks, hooks and inputs and outputs of the vb-prepare task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>vb-process-raw</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td>build/processed/* or ${gitSources}</td>
</tr>
<tr>
<td>output</td>
<td>build/optimized/*</td>
</tr>
</tbody>
</table>

The following table describes the options for the vb-prepare task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
<tr>
<td>rawSources Mode</td>
<td>no</td>
<td>false</td>
<td>Specifies if the task should copy raw application sources or metadata processed sources from the build/processed directory. Note: The implicit value is defined by the existence of build/processed directory (true if the directory exist).</td>
</tr>
</tbody>
</table>

vb-require-bundle

This tasks creates minified requirejs module bundles.

The following table describes the subtasks, hooks and inputs and outputs of the vb-require-bundle task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>config generator task: vb-require-bundle-configuration</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td>build/optimized/*</td>
</tr>
<tr>
<td>output</td>
<td>build/optimized/*</td>
</tr>
</tbody>
</table>

The following table describes the options for the vb-require-bundle task.
<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
<tr>
<td>Optimization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>emptyPaths</td>
<td>no</td>
<td>n/a</td>
<td>Comma-separated list of require paths that are set &quot;empty&quot;. Requirejs optimizer will not follow and bundle matching dependencies.</td>
</tr>
<tr>
<td>requirePaths</td>
<td>no</td>
<td>n/a</td>
<td>Requirejs optimizer paths mapping. This will override any default values or values read from app-flow.json. The value needs to be in a form of quoted JSON object: <code>--requirePaths='( &quot;foo&quot;: &quot;boo&quot; )'</code></td>
</tr>
<tr>
<td>bundles</td>
<td></td>
<td></td>
<td>Defines custom require module bundles.</td>
</tr>
<tr>
<td>bundles.modules</td>
<td></td>
<td></td>
<td>Specification of the modules that are supposed to be added to the enclosing bundle element.</td>
</tr>
<tr>
<td>bundles.exclude</td>
<td></td>
<td></td>
<td>Specify modules that shouldn't be part of the enclosing modules bundle. The exclusions are applied to all bundle modules, including modules added following transitive module dependencies.</td>
</tr>
<tr>
<td>bundles.</td>
<td></td>
<td></td>
<td>List of regular expression patterns used for matching optimized application resources. Regular expressions starting with exclamation mark are considered to be negative - resources matching these patterns won't be included.</td>
</tr>
<tr>
<td>transpile</td>
<td>false</td>
<td></td>
<td>Determines whether a separate set of require module bundles transpiled to ES5 code using babel preset-env preset should be created and stored into bundles/es5 directory. When this option is set to true application's index.html is modified so it contains a code snippet that switches between the original bundles for modern browsers and the ES5 versions for IE11.</td>
</tr>
</tbody>
</table>
vb-css-minify

This task minifies the CSS resources.

The following table describes the subtasks, hooks and inputs and outputs of the vb-css-minify task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>config generator task: vb-css-minify-configuration</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td>build/optimized/*</td>
</tr>
<tr>
<td>output</td>
<td>build/optimized/*</td>
</tr>
</tbody>
</table>

The following table describes the options for the vb-css-minify task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
</tbody>
</table>

vb-image-minify

This task minifies the image resources.

The following table describes the subtasks, hooks and inputs and outputs of the vb-image-minify task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>config generator task: vb-image-minify-configuration</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
</tbody>
</table>
The following table describes the options for the `vb-image-minify` task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
</tbody>
</table>

**vb-json-minify**

This task minifies JSON resources to remove white spaces.

The following table describes the subtasks, hooks and inputs and outputs of the `vb-json-minify` task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>config generator task: vb-json-minify-configuration</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td>build/optimized/*</td>
</tr>
<tr>
<td>output</td>
<td>build/optimized/*</td>
</tr>
</tbody>
</table>

**vb-export**

This task downloads application sources from the Visual Builder instance and expands the archive on the local filesystem for further processing.

The following table describes the subtasks, hooks and inputs and outputs of the `vb-export` task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>n/a</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td>build/optimized/*</td>
</tr>
<tr>
<td>output</td>
<td>build/optimized/*</td>
</tr>
</tbody>
</table>

The following table describes the options for the `vb-export` task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application selection options</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>yes</td>
<td>n/a</td>
<td>ID of the visual application you're going to build. The application needs to exist on the referred Visual Builder instance.</td>
</tr>
<tr>
<td>ver</td>
<td>yes</td>
<td>n/a</td>
<td>Version of the visual application</td>
</tr>
<tr>
<td>url</td>
<td>yes</td>
<td>n/a</td>
<td>URL of your Visual Builder instance</td>
</tr>
<tr>
<td>Authentication options</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>no</td>
<td>n/a</td>
<td>The username to be used to obtain OAuth access token for further communication with Visual Builder services.</td>
</tr>
<tr>
<td>Name</td>
<td>Mandatory</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>password</td>
<td>no</td>
<td>n/a</td>
<td>The password to be used to obtain OAuth access token for further communication with Visual Builder services.</td>
</tr>
<tr>
<td>accessToken</td>
<td>no</td>
<td>n/a</td>
<td>The value of an OAuth access token. If provided, username and password options are necessary.</td>
</tr>
<tr>
<td>sslCertificate</td>
<td>no</td>
<td>n/a</td>
<td>The path to the SSL certificate for the connection to Visual Builder instances provisioned with self-signed certificates.</td>
</tr>
</tbody>
</table>

### Build options

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gitSources</td>
<td>no</td>
<td>./</td>
<td>The location of the sources of the visual application.</td>
</tr>
</tbody>
</table>

---

**vb-manifest**

This task creates a build manifest that contains list of application resources, requirejs bundles mapping and the name of the `version_<hash>` directory.

The manifest location is `build/optimized/webApps/<webAppName>/build-info.json`.

The following table describes the subtasks, hooks and inputs and outputs of the `vb-manifest` task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td><code>config generator task: vb-manifest-configuration</code></td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td><code>build/optimized/*</code></td>
</tr>
<tr>
<td>output</td>
<td><code>build/optimized/*</code></td>
</tr>
</tbody>
</table>

The following table describes the options for the `vb-manifest` task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
</tbody>
</table>

---

**vb-process-raw**

This task processes application sources before the optimizations take place in raw sources mode.

This involves following actions: processing of application's index.html template. <!-- visualBuilderScripts -->, <!-- vb:inject id="headContent" --> and <!-- vb:inject id="headContent" theme="resources/css/" --> markers are resolved to appropriate code. This step is done by `vb-process-raw-index-html` task. copies /services directory from root of visual application to the optimized web application updates service references going outside of the application scope to local references referring the copy of /services directory in the web application copies /dynamicLayouts directory from root of visual application to the optimized web application
The following table describes the subtasks, hooks and inputs and outputs of the `vb-process-raw` task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td><code>vb-process-raw-index-html</code></td>
</tr>
<tr>
<td>multitask</td>
<td>n/a</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td><code>build/optimized/*</code></td>
</tr>
<tr>
<td>output</td>
<td><code>build/optimized/*</code></td>
</tr>
</tbody>
</table>

The following table describes the options for the `vb-process-raw` task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
</tbody>
</table>

**vb-process-raw-index-html**

This task resolves templating marks in an application's index.html.

This task removes the following templating marks:

- `<!-- visualBuilderScripts -->`
- `<!-- vb:inject id="headContent" -->`
- `<!-- vb:inject id="headContent" theme="resources/css/" -->`

The following template variables are used in the code that replaces these markers:

- `%BASE_URL%`
- `%JET_CDN_PATH%`
- `%JET_CDN_VERSION%`
- `%VB_CDN_PATH%`
- `%VB_VERSION%`

If the `resolveTemplate` configuration object is not provided for this task, these variables will be left unresolved in the resulting index.html.

If the `resolveTemplate` configuration object is set at least to empty object, the template variables will be resolved with values taken from the application's version files (`private/custom/versions.json` or `resources/package/versions.json`), or with default values if none of these exist.

```js
grunt.initConfig({
    "vb-process-raw-index-html": {
        options: {
            resolveTemplate: {
                BASE_URL: 'http://oracle.cloud/abc',
            }
        }
    },
});
```
The configuration object may also contain values for the other template variables. In such case, the configuration value has precedence before the versions files and defaults.

The following table describes the subtasks, hooks and inputs and outputs of the `vb-process-raw-index-html` task.

<table>
<thead>
<tr>
<th>Detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td>n/a</td>
</tr>
<tr>
<td>multitask</td>
<td>config generator task: _vb-process-raw-index-configuration</td>
</tr>
<tr>
<td>hooks</td>
<td>n/a</td>
</tr>
<tr>
<td>input</td>
<td>build/optimized/*</td>
</tr>
<tr>
<td>output</td>
<td>build/optimized/*</td>
</tr>
</tbody>
</table>

The following table describes the options for the `vb-process-raw-index-html` task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>no</td>
<td>build</td>
<td>Name of build directory.</td>
</tr>
<tr>
<td>resolveTemp</td>
<td>no</td>
<td>n/a</td>
<td>Values for template variables injected into the application's index.html</td>
</tr>
</tbody>
</table>

Customize Your Grunt Build Process

You can edit the `Gruntfile.js` file included in your application to customize the build tasks that are performed.

Topics:
- Add Custom Functionality to Existing Tasks
- Override Existing Grunt Tasks
- Optimize a Specific Web Application
- Run and Configure a Multitask
- Customize Bundle Modules
- Specify Options of Non-multitasks
- Specify Options for All Tasks

Add Custom Functionality to Existing Tasks

The public tasks exposed by the `grunt-vb-build` package have pre- and post- task hooks that you can redefine to add custom functionality.

For example, you can define hook tasks in `Gruntfile.js` to add some custom application tests before staging the application (`vb-pre-stage`) and some special application processing before the optimization step (`vb-pre-optimize`) by performing the following steps:

1. Open `Gruntfile.js`.
2. Edit the file to define the hook tasks.
To define the hook tasks `vb-pre-optimize` and `vb-pre-stage`, your edited `Gruntfile.js` might be similar to the following:

```javascript
module.exports = (grunt) => {
  require('load-grunt-tasks')(grunt);
  grunt.registerTask('vb-pre-optimize', () => {
    // add custom resources or modify existing resources here
  });
  grunt.registerTask('vb-pre-stage', () => {
    // run app tests here
  });
}
```

**Override Existing Grunt Tasks**

You can override an existing task by registering your own task under the same name.

To override an existing Grunt task:

1. Open `Gruntfile.js`.
2. Edit the file to redefine the task you want to override.

   For example, if you want to do a custom deployment of the application runtime artifact, you can override the `vb-stage` task. To redefine the task `vb-stage`, your edited `Gruntfile.js` might be similar to the following:

```javascript
module.exports = (grunt) => {
  require('load-grunt-tasks')(grunt);
  grunt.registerTask('vb-stage', () => {
    // do my own deployment of built "build/processed.zip" runtime application
    archive
  });
};
```

**Optimize a Specific Web Application**

You can optimize a specific application by editing `Gruntfile.js` or your custom Jenkins shell script to specify a target for the build task.

In `Gruntfile.js`, you can redefine the `vb-build` task and specify a target for the `vb-optimize` task. Alternatively, you can create a new task that performs steps similar to the `vb-build` task and specify the target.

To specify an app as a task target in `Gruntfile.js`:

1. Open `Gruntfile.js`.
2. Edit the file to define the tasks performed for the `vb-build` task and specify a target for the `vb-optimize` task.

   The target name is the path of the web application relative to the `/webApps` directory.

```javascript
module.exports = (grunt) => {
  require('load-grunt-tasks')(grunt);
};
```
grunt.registerTask('vb-build', [
    'vb-pre-build',
    'vb-clean',
    'vb-prepare-sources',
    'vb-optimize:myWebApp',
    'vb-stage',
    'vb-post-build',
]);

You can also define the target in your Jenkins shell script, for example, by modifying it similar to the following:

grunt vb-prepare-sources --url=... --id=... --ver=... --username=... --password...grunt
  vb-optimize:myWebApp

grunt vb-stage --url=... --id=... --ver=... --username=... --password...

You can also edit the script to run only specific optimization tasks:

# vb-image-minify-configuration will create configurations of vb-image-minify multitask for all existing web application
grunt vb-image-minify-configuration vb-image-minify

Run and Configure a Multitask

Multitasks require configuration. You can create this either by using the corresponding *-configuration tasks, or by defining the configuration in Gruntfile.js.

To configure a multitask in Gruntfile.js:

1. Open Gruntfile.js.
2. Edit the file to configure the multitask.

For example, to configure the vb-require-bundle multitask, you might edit the file to be similar to the following.

```javascript
module.exports = (grunt) => {
    grunt.initConfig({
        'vb-require-bundle': {
            options: {
                "transpile": true,
                "minify": true,
            },
            myWebApp: {
                options: {
                    "transpile": true,
                    "minify": true,
                },
            },
        },
    });
};
```
require('load-grunt-tasks')(grunt);

The top level task options are applied to all web applications. If you specify a target, the options are applied only to the target application.

Customize Bundle Modules

You can define the content of the requirejs module bundles to create multiple bundles when staging a visual application.

Pages that you want to load initially can be packaged in the main bundle, and other pages and pages in other flows that are not required initially can be packaged in a separate bundle that can be loaded when needed. Customizing the module bundles can help optimize the time needed to load and run the application.

For example, this configuration example shows how the vb-require-bundle task can be configured to create the following require module bundles for a web application named "webapp1".

- A bundle for all resources that belong to the application flow dashboard. This bundle will include all files that matches the "flows/dashboard" pattern. This will include all pages, models, resources and nested flows stored in the flows/dashboard directory. This bundle will not contain the module named helpers, that is referred to in one of the included page models.

- A bundle for resources that belongs to the application flow customers. In this case, the nested flows are excluded (as they are placed into a separate bundle). This bundle also excludes the helpers module.

- A bundle of resources of flows nested into the customers flow.

- A "base" bundle of application resources (shell pages and application resources, libraries and styles). This bundle explicitly adds the helpers module.

To customize the bundle modules:

1. Open Gruntfile.js.
2. Edit the file to configure the vb-require-bundle task.

```javascript
{
  "vb-require-bundle": {
    "webapp1": {
      "options": {
        "transpile": true,
        "minify": true,
        "bundles": {
          "dashboard": {
            "modules": {
              "find": ["flows/dashboard"]
            },
            "exclude": {
              "ids": ["helpers"]
            }
          },
          "customers": {
```
Specify Options of Non-multitasks

You can specify task options in the configurations for specific tasks.

When specifying a task’s options, you want to make sure that the options are applied only to the specific task. For example, specifying `--url` and `--username` options in the command line will override options specified in the `vb-stage` configuration.

To specify task options in `Gruntfile.js`:

1. Open `Gruntfile.js`.
2. Enter the options for the task.

For example, to override the URL and credentials parameters for the `vb-stage` task in order to stage to an instance other than where the sources were processed, your edited `Gruntfile.js` might be similar to the following:

```javascript
module.exports = (grunt) => {
  grunt.initConfig({
    'vb-stage': {
      options: {
        url: 'my production instance URL',
        username: 'production instance username',
      },
    },
  })
}
```
Specify Options for All Tasks

You can use a generic vb settings object to specify the configuration options that will be applied to all vb- tasks (for example, vb-build).

If you define the vb options, you don’t need to pass any Grunt command line parameters, but can simply run grunt vb-build.

To specify task options for all tasks in Gruntfile.js:

1. Open Gruntfile.js.
2. Enter the options for the task.

Options defined in a vb object are implemented by all vb- tasks. For example, to specify the URL, credentials and id and version options that all the build tasks will use, your Gruntfile.js might be similar to the following:

```javascript
module.exports = (grunt) => {
  grunt.initConfig({
    'vb': {
      options: {
        url: 'instance URL',
        username: 'instance username',
        id: 'myVisualApp',
        ver: 1.0
      }
    },
    require('load-grunt-tasks')(grunt);
  });
```

Specify a Custom App URL

Sometimes it's not appropriate to use the default URL that Visual Builder generates for your app. For example, if you’re building an app for your customers, you can use a custom domain for the app to shield customers from the details of your server’s host and domain name.

To use a custom domain for your app, your service administrator must first configure your instance to support the custom domain. After it is configured you can map the custom domain to your visual application in the Settings editor. After the visual application is staged and published, the web application and the business object APIs can be accessed directly using the custom domain.

Only one custom domain can be mapped to a visual application, and it can only be used to access one web application in the visual application. It is recommended that your visual application only contain one web application if you are going to use a custom domain to ensure that the correct web application is loaded.

Multiple custom domains can be used in an instance, but each must be mapped to a different visual application. For example, if the visual application myvisualapp1 is mapped to the subdomain mysubdomain1, if you want to map mysubdomain2 to an
application it must be mapped to a different visual application (for example, myvisualapp2).

To map a custom domain to a visual application:

1. Open your visual application and choose Settings in the application’s Options menu in the toolbar.

Alternatively, on the Oracle Visual Builder home page, locate the visual application where you want to change the settings and choose Settings in the application’s Options menu.

2. Open the Applications tab in the Settings editor.
3. Type the URL in the Vanity URL text field. Click Close.

The URL must be the full URL that you want to use and use a valid form (for example, https://foo.example.org).

After you publish the visual application, a visitor can type the custom domain (for example, https://foo.example.org) in the browser to open the web application. The URL will not contain any additional path parameters because the app is loaded as the root domain.

Specify Custom JET and Runtime Versions for Web Apps

Oracle Visual Builder exposes input fields in the Application Settings page that allow you to specify custom JET and VB runtime versions for web and Progressive Web Apps (PWA).

Use the Custom VB Runtime Version and Custom JET version fields if you want to make a minor change and republish an existing web or PWA application without upgrading to the current Visual Builder Runtime version or JET version. We recommend you always use the latest version of Visual Builder and JET to take advantage of new features and fixes.

If you republish your application again at a later date, Oracle Visual Builder updates your application to the latest runtime version(s) unless you again specify the custom versions you want to use. That is, you must specify the custom versions you want to use each time. Otherwise, Oracle Visual Builder upgrades your app(s) to the latest versions.

To specify custom JET or runtime versions:
1. Click the **Web Applications** tab for a web app or the **Mobile Applications** tab for a PWA.

2. Click the `<app name>` node and click the Settings icon (⚙).

3. Specify a value for the Visual Builder Runtime version, JET version, or both.

   You can specify one or both values. When specifying a value, ensure that the specified version matches an earlier and released version of the software. Oracle Visual Builder uses the specified version when you preview, stage, and publish the application.

<table>
<thead>
<tr>
<th>Custom VB Runtime Version</th>
<th>Custom JET Version</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specified</td>
<td>Specified</td>
<td>Specified versions of VB Runtime and JET are used</td>
</tr>
<tr>
<td>Specified</td>
<td>Not specified</td>
<td>Specified VB Runtime version is used and the matching JET version specified in its <code>versions.json</code> is used</td>
</tr>
<tr>
<td>Not specified</td>
<td>Specified</td>
<td>Specified version of JET is used and the default VB Runtime version for the build of VB Design time is used</td>
</tr>
</tbody>
</table>
Add Team Members

Add team members to an application to enable other developers in the identity domain to contribute to developing the application.

To allow other team members to collaborate on the same application, you need to explicitly add the name of each team member in the application’s Settings editor. To add a team member to an application:

1. Open your web or mobile application and choose **Settings** in the application’s Options menu in the toolbar.

   ![Menu Options](image)

   Alternatively, on the Oracle Visual Builder home page, locate the application where you want to change the settings and choose Settings in the application’s Options menu.

2. Open the **Team** tab in the Settings editor.

   The Team tab contains a Members panel that displays a list of current team members. The tab also displays a History panel that displays the time of the last update to the application and the name of the team member who made the update.

---

**Table 3-1  (Cont.) Behavior for custom values**

<table>
<thead>
<tr>
<th>Custom VB Runtime Version</th>
<th>Custom JET Version</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not specified</td>
<td>Not specified</td>
<td>Default VB Runtime and JET versions for this build of VB Design time are used</td>
</tr>
</tbody>
</table>

---

Chapter 3

Add Team Members
3. In the Members panel, select a team member's email from the dropdown list. Click Add.

The dropdown list displays the email addresses of all the members in your identity domain who can be added to the application as developers.
Develop Applications

To develop an application you need to define its metadata and create its pages and artifacts.

Topics:

- Understand an Application’s Structure
- Understand the Application Designer
- Check Application Code
- Create Web and Mobile Applications
- Work with Variables and Types
- Work with Flow Artifacts
- Work with Actions and Action Chains
- Design Pages
- Secure the Application
- Configure Data Cache and Offline Support
- Work with Application Resources
- Work with OJET Web Components
- Work with Translations
- Integrate Your Application With a Git Repository

Understand an Application’s Structure

Your visual application can contain multiple web and mobile applications. The web applications in your visual application are displayed when you open Web Applications in the Navigator. The mobile applications are displayed when you open Mobile Applications in the Navigator.

The structure and required files are created by default when you create a web or mobile application. Open Web Applications or Mobile Applications in the Navigator to see a visual representation of the structure of your application and to navigate to the application’s artifacts. In the Navigator, the artifacts are represented as one object, but they actually represent two or three separate files that describe the artifact's behavior and properties. The properties and behavior of an artifact are built by creating and combining the building blocks described in these files. For example, when you edit a flow artifact, you will describe the artifact’s variables and action chains in the artifact’s json file, and functions are defined in the artifact’s JavaScript file. See Understand the Application Designer.

When you open an application in the Navigator, the structure of the application is displayed as nodes and subnodes representing the application's artifacts and files. You can collapse and expand nodes to hide and reveal the contents. Selecting an
artifact or file in the Navigator opens the artifact in the one of the editors in the application designer.

The following table describes the nodes and artifacts that are displayed when a web or mobile application is open in the Navigator.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>All the artifacts of an application are grouped under the application node in the Navigator. You will see an application node for each of the apps in your visual application. For example, if your visual application has three mobile apps, you will see three application nodes in the Mobile Applications pane. You can select the application node to view the application artifact in the designer. The application artifact represents the files and metadata that describe your application. The application artifact has the same name as your app. The descriptions of the application artifact are contained in the app-flow.json, app-flow.js and index.html source files. The metadata in these files defines the artifacts that can be used by every artifact in your application, for example, the variables that are application-scoped, types that describe data structures, and security settings for the application. See Understand the Application Designer.</td>
</tr>
<tr>
<td>Item</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>flows</td>
<td>All the individual flows in your app are grouped under the flows node. Your app can have multiple flows, and each flow can contain multiple pages. Depending on the type of application, one or more flows are created by default when you create the application. In mobile apps, the default flow is named &lt;flowname&gt;. In web apps, the default flow is named main. A flow consists of a flow artifact and the pages within the flow. The descriptions of the flow are contained in the &lt;flowname&gt;-flow.json and &lt;flowname&gt;-flow.js source files. See Work with Page Flows.</td>
</tr>
<tr>
<td>pages</td>
<td>Pages are grouped by flow under the flows node. Each page uses a HTML file to specify the page elements, a JavaScript file that determines the page’s functions, and a JSON file for the page’s metadata. See Design Pages.</td>
</tr>
<tr>
<td>resources</td>
<td>The resources node contains resources available to your application, such as images, style sheets (css) and translation files (strings). See Work with Application Resources.</td>
</tr>
<tr>
<td>root pages</td>
<td>The root pages node contains one or more root page artifacts where flow artifacts are embedded. The artifact describes the area containing the flow and the areas outside the page flow that can contain elements such as a header or title area, a navigation toolbar, and a footer. An application typically contains one root page artifact for each flow. By default, a root page artifact named app is created when you create a mobile app and an artifact named shell is created when you create a web application.</td>
</tr>
</tbody>
</table>

**Note:**

You can open the Source View in the Navigator to view all the source files in the visual application.

**Understand the Application Designer**

When designing your application you will use various tools in the application designer to view and edit the application’s artifacts and pages.

You use the application designer to create and edit the building blocks that prescribe the behavior of pages, application artifacts and flow artifacts in your web and mobile applications. The application designer has dedicated editors for each of the building blocks used to build your application. The editors provide an easy to use interface for editing the files containing the descriptions of artifacts and pages. For example, when designing a page, you might need to define page variables and create action chains, in addition to positioning page components in the page and specifying component properties. The application designer provides editors to help you perform these tasks so that you don’t need to edit the source code of the HTML, JavaScript and JSON files used to describe the layout and behavior. The application designer also provides a Metadata editor if you want to edit the JSON files directly.
The editors in the application designer provide forms and wizards to help you create and edit the building blocks, for example, to create action chains and to map parameters to variables. When designing pages you can use a visual editor to compose your pages, and use the Quick Starts to help you create the building blocks needed to add some of the functionality commonly used in applications.

When you open an artifact, each artifact opens in a separate tab in the application designer. You switch between editors by selecting the editor’s icon in the left margin of the application designer. You use the same editors to edit each of your artifacts, with the exception that page artifacts provide a visual Designer for designing the page’s layout, and application and flow artifacts provide a Page Flow view for displaying the page artifacts they contain.

The following table describes the editors available for editing artifacts.

<table>
<thead>
<tr>
<th>Editor Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Page Flow" /></td>
<td>The Page Flow view displays the pages in your application and the navigation flows between the pages. When you open an application artifact, the Page Flow view displays the root page artifacts in your application. When you open a flow artifact, the Page Flow view displays the page artifacts contained in the flow. You can double-click an artifact in the Page Flow view to open it in the Designer. The Page Flow view is only available with application and flow artifacts.</td>
</tr>
<tr>
<td><img src="image" alt="Designer" /></td>
<td>The Designer is a visual editor for composing the layout of a page. The Designer contains a canvas that represents the page layout and a palette with page components that you can drag onto the canvas to add to the page. The Designer has a Property Inspector that you use to specify the properties of the page’s components and to open Quick Starts. See Use the Page Designer and About Quick Starts. The Designer is only available with page artifacts.</td>
</tr>
</tbody>
</table>
Check Application Code

Use the Audits feature to check and verify the code of your application as you develop it. This feature helps you catch and fix errors before you stage and deploy your application.

When you use Audits, Visual Builder scans and runs checks on the code in your application and displays errors and warnings, if any. In addition to reporting errors related to JET components, the Audits pane displays other errors, such as syntax errors, warnings for missing dependencies, and translation-related errors. You can view details for the issues and also resolve issues from the Audits Pane.

1. Click Audits to view the Audits pane.
   The application is scanned and results are displayed in the Audits pane. The results are sorted into four categories based on severity.

2. Review the listed issues.
   For each issue, you can review severity, issue details, source and location.

3. Optionally, filter the issue list to review specific issues by using the Search field and by selecting or deselecting the severity buttons.
4. Select an issue and click the Menu icon ( ).
5. Select Open in Source Editor or Open Artifact to view the source code for the issue.
6. Select Do not report this type of defect again to ignore similar defects in future.
7. Resolve the issue, if possible, by selecting the appropriate option.

For example, for the String not externalized for translation issue, select the Add to translation bundle option to fix the issue.

Similarly, for the Element should have non empty ID attribute type issues, select the Add missing ID attribute option to fix the issue.

Create Web and Mobile Applications

Your visual application can contain multiple web and mobile applications. Each application in your visual application is independent, but they can all use the data sources defined in the visual application.

Topics:
- Create a New Web Application
- Create a New Mobile Application
- Add a Custom Plugin to Your Mobile Application

Create a New Web Application

Each web application in your visual application is independent, but they can all use the data sources defined in the visual application.

To create a new web application:
1. Click Web Applications in the Navigator to open the Web Apps pane.

Structural representations of each of the web applications in your visual application are displayed in the Web Apps pane.
If no web applications have been created, you will see a message in the pane and a + Web Application button that you can click to open the Create Web Application dialog box.

2. Click **Create Web Application** (➕) in the Web Apps pane.

The Create Web Application dialog box contains an Application Name field where you enter the name of your new web application and options for the navigation style of your app.
3. Enter the name for the new app in the dialog box.
   The name you enter is used as the display name in the Web Apps pane in the Navigator.

4. Select the None or Tabbed navigation style for the web app.
   For the Tabbed navigation style, you can specify the number of navigation items and a name for each.

5. Click **Create**.

Create a New Mobile Application

Each mobile application in your visual application is independent, but they can all use the data sources defined in the visual application.

To create a new mobile application in a visual application:

1. Click **Mobile Applications** in the Navigator to open the Mobile Apps pane.
   Structural representations of each of the mobile applications in your visual application are displayed in the Mobile Apps pane.

   ![Mobile Apps Pane](image)

   If no mobile applications have been created, you will see a message in the pane and a **+ Mobile Application** button that you can click to open the Create Mobile Application wizard.

2. Click **Create Mobile Application** in the Navigator.
   In the General Information page of the Create Mobile Application wizard, specify the application name for the mobile application you are about to create. You can also choose from three navigation styles. The Horizontal navigation style with three navigation items is configured by default. A Vertical navigation style is also
available that allows you to create a root page with a navigation drawer panel including a header with an avatar, navigation items in the middle, and a footer with the application name. A page flow, with a starter page, is generated for each navigation item. Alternatively, select None if you want to configure the navigation items for your mobile application later. If you select None, one page flow is generated for the mobile application.

3. Click Next.

The Page Template – main page of the Create Mobile Application wizard is displayed.

4. Select Custom and click Create.

Add a Custom Plugin to Your Mobile Application

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

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

Oracle Visual Builder provides a range of plugins that are ready to use in the mobile applications that you create. One example is the camera plugin that is used when you configure your mobile application to use the Take Photo action. The plugins that Oracle Visual Builder delivers provide coverage for a wide range of use cases. For those use cases where Oracle Visual Builder does not provide a plugin ready to use,
you can import a plugin into your mobile application using the Custom Plugins page of your mobile application’s Settings screen. To identify the plugins that Oracle Visual Builder delivers, run the following command from the Cordova command-line interface after you add the Android or iOS platform, as described in the readme.md file included in the Cordova project source that you’ll download as a first step to adding a plugin to your mobile application.

cordova plugin ls

To find a plugin to use in your application, go to the Plugins page at https://cordova.apache.org/plugins/ that provides a registry of core and third-party Cordova plugins. Core plugins are plugins provided by Apache Cordova while third-party plugins are those developed by independent developers. Core plugins display a blue vertical strip to the left of the card that describes the plugin, as shown in the following image, where a core plugin (cordova-plugin-dialogs) appears after a third-party plugin named com-artemisofnian-plugins-unityads2.

The Plugins page categorizes plugins according to the platform on which the plugins will run. Some plugins run on a wide variety of mobile platforms (Android, iOS, and so on), such as the cordova-plugin-dialogs shown in the image while other plugins may only support one platform. Ensure that the plugins you install in your application support the platforms where your application will run. Each entry generally provides information such as a brief description of the plugin’s functionality, the platforms it supports, and the number of days since it was updated. It then links to a web page where you can access more documentation for the plugin, such as how to install it in your application, how to use the APIs it provides, and any known issues with regard to its usage.

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

Having identified or developed the plugin you want to use in your mobile application, download the cordova-package.zip file using the Cordova Project Source link in the Custom Plugins tab of your mobile application’s Settings page. The Readme.md file that is included in the cordova-package.zip file provides step-by-step instructions that describe how to add a plugin to the Cordova project and build the application for Android and/or iOS. Once you have successfully built the application, you upload the
APK file (for Android) or the IPA file (for iOS) using the appropriate Upload Template Application dialog that you access from the **New** dropdown list in the Uploaded Templates section of the Custom Plugins tab.

You can upload one debug and one release template application for both Android and iOS. After you upload the template application, the build configuration for the platform and build type references the template application that you have uploaded. For example, if you upload a template application for Android with a built type of Release, that template application is associated with the build configuration of Release type for Android.

Having uploaded the template application that packages the Cordova plugin that you want to use in your mobile application, you can now write JavaScript functions that invokes the interfaces that the plugin provides access to. To accomplish this task, refer to the documentation that is provided with the plugin. Plugin documentation should also identify known issues with the plugin, if any exist. The following snippet of sample
code demonstrates how you might initiate a barcode scan using the phonegap-plugin-barcode scanner plugin from a mobile application that uses this plugin.

```javascript
function _showBarcodeScanner() {
    // wrap the call with a promise
    var barcodeScannerPromise = new Promise(function (resolve, reject)
    {
        window.cordova.plugins.barcodeScanner.scan
        (function (result) {
            resolve(result.text);
        },
        function (err) {
            console.error(err);
            reject(err);
        });
    });
    return barcodeScannerPromise;
}
```

Work with Variables and Types

An application artifact describes the variables, types and action chains that can be used in any page in your application, as well as other configuration settings and properties of the application. When you open an application artifact, various editors are available for examining and modifying the artifact.

Topics:
- About Variables and Types
- Create Variables in Artifacts
- Create a Custom Type
- Create a Custom Type From an Endpoint

About Variables and Types

Variables can be used in any page, and custom types can be used to define the data of any variable. Variables and types can be created in application, flow and page artifacts. Variables created in an application are application-scoped and can be used to store data that can be used anywhere within the application, both within and across flows and pages, for example, for a login name. Variables created in a flow are flow-scoped and can be used in any of the pages within that flow. Variables created in a page can only be used within that page.

You use variables to store data that can be used by actions and page components in your application. Application variables are application-scoped, so their use is not limited to a specific page and they are accessible from any of the pages of an application. Application variables are useful for storing states, input parameters, login details and other data that you want to be accessible both within and across flows and pages. For each variable you must specify a Type property to define the type of data that is stored in the variable. See Understand Variables.

You use types to define the data type and structure of a variable, and every variable must be assigned a type. Types can be application-scoped, flow-scoped or page-
scoped, and can be assigned to application, flow and page variables. There are two kinds of types that can be assigned to variables: built-in types and custom types. Standard built-in types are used to specify data that are a primitive type, a structure or a simple array or collection, for which each field can either be a primitive or a structure. There are also two special built-in types that are typically used for storing data retrieved from an endpoint and are used when some operations need to be performed on the data: Array Data Provider and Service Data Provider. For details about how to use types in the page model, see Types.

You can define custom types from an endpoint or your can define a custom object or array.

Create Variables in Artifacts

You can create variables in application, flow and page artifacts. Variables are assigned a scope that is based on where they are created, and the scope determines where they can be used. When you are deciding where to create a variable, you should consider where it might be used.

Each variable is required to have a unique name (Id) and a type. When you create a variable you must specify one of the built-in types or a custom type. After you create the variable, you can edit its properties and attributes in the Variables tab for example, to identify it as an input, or to add attributes if its type is array or object. You can also use a variable to trigger an action chain when its value changes. You can use the Events tab in the Variable Information pane to add onValueChanged event actions and specify the action chain the change will initiate.

To create a variable in an artifact:

1. Open the Variables tab in the Variables editor of the artifact.

   The Variables editor contains a Variables tab and a Types tab. The Variables tab displays a list of the variables that are already defined for the artifact. You can enter a string in the filter field to limit the list to variables containing the string. The Variable Information pane is used for editing the details and event actions for the selected variable.
You will see a message if no variables are defined.

2. Click + Variable to open the New Variable dialog box.

3. Type a name for the variable in the Id field and select a type in the Type dropdown list.
   The dropdown list displays the built-in types as well as any custom types that can be applied to the variable.
4. Click Create.
   
   You can select the Create Another checkbox to immediately create another variable.
   
   After you create a variable you can select the variable in the list and edit its properties and add event actions in the Variable Information pane.

Create a Custom Type

You create a custom type when you want a type that defines an array or an object, and you want to individually add each of the attributes that define the type’s data structure.

To create a custom type:

1. Select your application, flow or page artifact in the Navigator.
2. Open the Types tab in the artifact’s Variables editor.
   
   The Types tab displays all the types defined for the artifact.
3. Click + Type and select Custom in the menu.

The New Type dialog box appears when you select Custom in the menu.

4. Type a name for the new type and choose either Object or Array as the type. Click Create.

When you click Create, the new type is added to the list in the Types tab. You now define the structure by adding attributes.

5. In the Types tab, click Add (➕) next to the new type to add an attribute.
6. Enter the name and select a type for the new attribute. Click Create.

You can select the Create Another checkbox if you want to immediately add another attribute to the type.

You can continue to refine the data structure of the type by adding attributes.

Create a Custom Type From an Endpoint

When you create a custom type from an endpoint, you define a data structure by selecting an endpoint and then choosing from the fields available at the endpoint. Custom types can be used to define the structure of any variable in your application.

For example, when sending a request to an endpoint `getall_Contact` you might want the structure of the response to be an array with the id and a few specific fields (a string name and a string email). You can create a custom type from the endpoint and select the fields that you want in the response. All variables that are assigned this custom type will have the same data structure.

To create a custom type from an endpoint:

1. Select your application, flow or page artifact in the Navigator.
2. Open the Types tab in the artifact's Variables editor.
3. Click + Type and select From Endpoint in the menu.
4. Select an endpoint from the list. Click Next.

5. Select the endpoint attributes you want to include in the data structure. Click Finish.
If you expand your new custom type in the Types tab, you can see it is a custom object type with an array items containing the fields in the endpoint that you selected.

Work with Flow Artifacts

You use flows to group the pages of your application together.

Topics:

- Work with Page Flows
- Set the Default Page for a Flow
• Embed a Flow Within a Page
• Navigate Between Flows in the Root Page

Work with Page Flows

Each page in your application is contained within a flow. You create a flow to group one or more pages that you might want to treat as an independent unit that performs some function in your application, for example, a flow that contains the pages and artifacts used to register a new user.

Your application can have multiple flows. When you open your web or mobile application in the Navigator, you can expand the flows node to see each of the flows in the application. You can create new flows in your application by clicking the Create Flow icon (.CreateDirectory) next to the flows node. When you create a flow, a page within the flow is automatically created and set as the flow's default page.

A flow named main is created by default when you create an application. By default, the main-start page in the main flow is set as the default page in the flow, and the main flow is set as the default flow in your application.

When you expand the flows node of a web or mobile application in the Navigator, you can see that a flow consists of the following artifacts and resources.
<table>
<thead>
<tr>
<th>Artifact</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow artifact</td>
<td>Open the flow artifact in the designer to edit the flow’s metadata, such as the variables, types, chains and functions that can be used in pages in the flow. In the Settings editor for the flow you can set the default page in the flow. You can expand the flow artifact in the Navigator to see the pages contained in the flow. Click the Create Page icon ( + ) next to the flow artifact to create a new page in the flow.</td>
</tr>
<tr>
<td>Page artifact</td>
<td>Open each page artifact in the designer to edit the page’s layout and other page metadata.</td>
</tr>
<tr>
<td>Resources</td>
<td>Each flow might contain a resources folder that contains images and translation resources that can be used in pages in the flow. See Work with Application Resources.</td>
</tr>
</tbody>
</table>

Set the Default Page for a Flow

When you create a flow, a start page is automatically created and set as the entry page for the flow. You can change the flow’s default entry page in the Settings editor of the flow artifact.

To set the default page for a flow:

1. Open the flow artifact in the Settings editor of the application designer.
2. In the General Settings tab, use the Default Page dropdown list to select the page in the flow that you want to be the default page.
Embed a Flow Within a Page

Each flow in your app can contain multiple sub-flows, enabling you to embed pages within other pages. You use the Flow Container component to create a container in the page where you can then embed sub-flows. After adding the container, you set the default sub-flow displayed in the container in the General Settings tab of the page’s Settings editor.

In the page containing the embedded flow, you can only edit the page content outside the Flow Container component, and the embedded pages are not visible on the canvas. After setting the default sub-flow for the page, you can run the app to see the embedded content in the page.

To edit a page in a sub-flow, you need to open it in the Page Designer. To help you visualize the page, the canvas displays the content embedded in the Flow Container component and the content of the parent page, but you can only edit the content in the Flow Container. You set the default page for the sub-flow in the General Settings tab of the sub-flow’s Settings editor.

You can use an embedded flow to isolate content from the page containing the flow, and to allow navigation between pages in the sub-flow without leaving the page containing the sub-flow.

To embed a flow in a page:

1. In the Navigator, locate the flow containing the page where you want to embed the flow.

2. Under the flow containing the page, click Create Flow (⁺) and enter a name for the new sub-flow in the Create Flow dialog box. Click Create.
3. Open the page where you want to embed the new flow.

4. In the Page Designer, drag the Flow Container component from the Layout category in the Components palette and place it on the canvas.

5. In the page’s Settings editor, select the default sub-flow in the General Settings tab.

   The Default Flow dropdown list displays all flows within the current flow.

### Navigate Between Flows in the Root Page

You can use the `navigateToItem` event in an action chain to open the start page of a flow in your app’s root page. You typically invoke the action chain from the drawer or tab elements used to navigate the app.

To use the `navigateToItem` event in an action chain, you add the Fire Custom Event action to the chain and then select the event and assign the name of the target flow to the event’s payload. To use the `navigateToItem` event, the current page needs to have a flow container that is configured to hold the target flow. Firing the event loads the start page of the target flow into the flow container of the root page.

For example, when you create an app from a template that uses navigation elements such as a tabs or a drawer, the app will contain separate flows that can be loaded into the app’s root page. If you create a web app with two flows (for example, “item-1” and “item-2”), and you want to use tabs in your root page to select the flow that is displayed in the flow container, you can create an action chain that fires the `navigateToItem` custom event. You would create an action chain for each of the flows, and add a click event listener to each tab to trigger the action chain.
To create a new action chain that navigates to the start page in a flow:

1. In the Designer, open the root page containing the flow container and select the component on the canvas that will open the flow.
   For example, to navigate between flows displayed in the main flow container of a web app you will need to open the root page and select one of the navigation tabs.

2. In the Events tab of the Property Inspector, click **New Event** and select the Quick Start option.
   A new action chain opens in the Action Chain editor when you select the Quick Start option.

3. Type a name for the action chain in the Id field of the Property Inspector.

4. In the Action Chain editor, drag the Fire Custom Event action into your chain.

5. In the Fire Custom Event pane, select `application:navigateToItem` in the Event Name drop-down list.
   The drop-down list contains a list of custom events that can be invoked by the action. The `navigateToItem` event is prepended with "application:" in the drop-down list and can be used in any page in your app because it is application-scoped.
6. Click Assign to open the Fire Custom Event window for mapping the `navigateToItem` payload to the name of the target flow.

7. Select item in the Target column and select **Static content** in the editor pane.

8. Type the name of the target flow (for example, "item-1") in the editor pane of the window. Click **Save**.
Work with Actions and Action Chains

You create action chains to define the behavior of components in a page.

Topics:

• About Action Chains
• Create an Action Chain
• Add an Assign Variables Action
• Add a Call Action Chain Action
• Add a Call Component Method Action
• Add a Call Module Function Action
• Add a Call REST Endpoint Action
• Add a Fire Custom Event Action
• Add a Fire Notification Action
• Add a Navigation Action

About Action Chains

An action chain is a sequence of actions that you create to define the behavior of components.
Action chains are triggered by events such as click, onValueChanged and focus that you add to components and page events. Action chains have a scope property that determines where they can be used. Action chains that are created in a page’s Actions editor can only be used within the context of the page and cannot be used in other pages. Action chains created in the Actions editor for a flow can only be used within that flow. See Add Events to Components and Start an Action Chain with an Event.

![Image of Navigate to Page action chain]

For action chains that you might want to use in multiple pages of your application, such as one that navigates to the start page, you can create an action chain that is application-scoped.

You create action chains by assembling simple, individual actions into a sequence in the Actions editor. The Actions editor contains a palette of built-in actions that you drag onto the canvas where you create your sequence. After you add an action to the sequence you usually need to provide details required by the action. For example, when you add the Call REST Endpoint action to your action chain you need to specify the endpoint and other details about the response, and you are required to select a page in your application when you add the Navigate to Page action to a chain.
The built-in actions are grouped in the palette by type. See Actions for a list of the built-in actions.

Create an Action Chain

You build action chains by dragging actions from the palette onto the canvas in the Action Chain editor.

To create an action chain:

1. Open the Actions editor of your page.

   The Actions editor displays a list of the page’s action chains, or a message if no action chains are defined.
2. Click + Action Chain.

3. Enter a name for the action chain in the Id field. Click Create.

   The new action chain opens in the editor. The Action Chain editor contains the palette of built-in actions, a canvas and a property editor. The Start icon indicates
the entry point of your action chain and the Add icon ( + ) is a placeholder indicating where you can add an action to the chain. The property editor displays the properties of the action that is selected on the canvas.

4. Drag an action from the palette and place it on the Add icon ( + ) in the chain.

   Alternatively, you can click the Add icon ( + ) in the chain and select an action in the popup menu of actions.
To create a fork in your action chain, drag the action from the palette and drop it on the Add icon next to the action where you want the chain to fork. The Add icon appears next to each action in the chain when you drag an action from the palette.

The new action is added to the chain and is selected by default. The property editor displays the properties that you can specify for the action. The properties depend upon the type of action.

5. Specify the properties of the action in the property editor.
6. Repeat steps 5 and 6 until your action chain is complete. The action chain is saved automatically.

You can open your action chains at any time from the Actions editor and edit them as necessary.

Tip:
To remove an action from the chain, select the action on the canvas and click Delete (Trash) in the property editor.

When your action chain is complete you can start the action chain from an event in the page.

Start an Action Chain with an Event

You can start an action chain when an event occurs in an artifact. The types of events that are available depend on the type of artifact.

You select the type of event and the action chain in the Events editor of an artifact. For example, you can use a page event, such as entering a page, to start an action chain. To start the action chain you would open the Events editor for the page to define the event, and then select the action chain that the event should start. The action chains that can be started from the event will depend on the scope of the action chain. You can also use a Quick Start to assign events to components.

To add an event to start an action chain:

1. Open the Events editor for the artifact that you want to trigger the action chain.

The Events editor displays a list of all events defined for the artifact.
2. Click + Event Listener in the editor.

The Select Event window displays a list of events that can be used to start an action.

3. Select an event type from the list. Click Select.

4. Select the action chain that will be started by the event.

You can select any action chain that is scoped for the artifact. The window lists all action chains that can be started in the current context, grouped by scope. For example, if you are creating an event for a flow artifact, you can only call action chains defined in the flow or in the application.
5. Click **Select**.

**Add an Assign Variables Action**

You add an Assign Variables action to an action chain to map the source of some value to a variable. The variable can be used by other action chains or bound to a component.

For example, if your action chain sends a request to a GET endpoint, you can use the Assign Variables action to map the response to a page variable bound to a page component. Another example of assigning a variable is when you want to capture the ID of an item selected in a list. You could use a Selection event to start an action chain that assigns the selected item’s ID to a variable.

To add an Assign Variables action to an action chain:

1. Open the Actions editor for the page.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Assign Variables** from the Actions palette into the action chain.

You can drag the action onto the Add icon ( ) in the action chain, or between existing actions in the chain. The property editor opens when you add the action to the chain.
The Assign Variables action is badged with a warning icon when no variables have been assigned.

4. Type a name for the action in the Id field in the property editor.

5. Click Assign in the property editor to open the Assign Variables window to map the source of the value to a page variable.

6. Drag the sources of the values in the Sources pane onto targets in the Targets pane. Click Save.

   Each target can only be mapped to one source, but you can use the action to assign multiple variables. For example, you might want to map a value from the Chain in the Sources pane, such as an input variable or the result of an action, to a Page variable or to the input of another action in the Target pane. When you select the variable in the Target pane, the expression editor in the dialog box displays the expression for the source.
Add a Call Action Chain Action

You add a Call Action Chain action to an action chain to start a different action chain. The action can call other action chains defined in the same page or in the application. To add a Call Action Chain action:

1. Open the Actions editor for the page.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Call Action Chain** from the Actions palette into the action chain.

You can drag the action onto the Add icon (⊕) in the action chain, or between existing actions in the chain. The property editor opens in the editor when you add the action to the chain.
4. Click **Select Action Chain** in the property editor.

The Select Action Chain window displays a list of available action chains. Depending on where you are creating the action chain, the window might have tabs for action chains defined in the page, in the current flow and in the application. If you are creating an action chain in a flow artifact, you can only select other action chains defined in the same flow artifact or in the application artifact, and you will not see a tab for Page action chains. You can use the Filter text field to filter the action chains in the list.
5. Optional: If the action chain that is called requires input parameters, click Assign in the Input Parameter section of the property editor to map the input parameter to a variable.

The action in the action chain is badged with a warning icon when an input parameter is required but has not been assigned.

You map variables to parameters by dragging the variable for the source value in the Sources pane onto the Parameter for the input parameter in the Target pane. Click Save.

---

**Add a Call Component Method Action**

You add a Call Component Method action to an action chain to call a method on a component.

To add a Call Component Method action to an action chain:

1. Open the Actions editor for the page or application.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Call Component Method** from the Actions palette into the action chain.
You can drag the action onto the Add icon (++) in the action chain, or between existing actions in the chain. The property editor opens when you add the action to the chain.

4. Type a name for the action in the Id field in the property editor.

5. Enter the name of the component in the Component Selector text field.
   In the text field you enter the actual name of the component that you want to call, for example, $page.components.byId('myCard')

6. Enter the Method Name and map the parameters required by the method.

Add a Call Module Function Action

You add a Call Module Function action to an action chain to call a function defined for the current page, current flow or the application. You create and edit module functions in the Functions editor.

To add a Call Module Function action to an action chain:

1. Open the Actions editor for the page or application.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag Call Module Function from the Actions palette into the action chain.

You can drag the action onto the Add icon (++) in the action chain, or between existing actions in the chain. The property editor opens when you add the action to the chain.
4. Type a name for the action in the Id field in the property editor.

5. Click **Select Module Function** to open a window to select the functions available.

6. Select the function in the Select Module Function window. Click **Select**.
   You can select functions that are defined for the current page, the current flow or for the application.

7. Specify any input parameters and return type for the function in the property editor.
   You can click **Assign** to map variables to the parameters.

### Add a Call REST Endpoint Action

When you add a Call REST Endpoint action to an action chain you might need to specify input parameters for the request or create variables for the response that you can bind to page components.

When you add the Call REST Endpoint action to an action chain, the endpoint that you select will depend upon the functions that are available. Depending on the function you might also need to create some variables to map to the action’s parameters, such as input parameters and the action’s results. If you did not create the variables before creating the action chain, you can edit the action chain after creating the variables you need.

For example, an endpoint might require an ID to identify a record. In this case you will need to create a page variable that stores the ID, and that variable needs to be mapped to the action’s input parameter.

You will use the Call REST Endpoint action in action chains that perform typical functions such as creating, updating and deleting records, and any time you want to display the details of a record in a page. You can use the Quick Starts to help you create the action chains and variables for these functions.
<table>
<thead>
<tr>
<th>Type of Endpoint</th>
<th>Typical Requirements</th>
</tr>
</thead>
</table>
| POST             | When you call a POST endpoint you will typically need the following:  
|                  | • Parameters: The page variable for the data needs to be mapped to the parameters of the payload of the POST call.  
|                  | • No input parameter is required. |
| GET              | When you call a GET endpoint you will typically need the following:  
|                  | • Input parameter: The ID of the record you want to retrieve should be passed as an input variable.  
|                  | • The payload of the GET call needs to be assigned to a variable using the Assign Variable action.  
|                  | When you want to send a request to a GET endpoint to retrieve a collection, you will typically use a page variable of the type ServiceDataProvider. |
| DELETE           | When you call a DELETE endpoint you will typically need the following:  
|                  | • Input parameter: The ID of the record you want to delete should be passed as an input variable.  
|                  | • There is no payload when calling a DELETE endpoint. |
| PATCH            | When you call a PATCH endpoint you will typically need the following:  
|                  | • Input Parameter: The page variable storing the ID of the record you want to update should be mapped to the Input Parameter.  
|                  | • Parameters: The page variable for the updated data needs to be mapped to the parameters of the payload of the PATCH call. |

To add a Call REST Endpoint to an action chain:

1. Open the Actions editor for the page.
2. Click the action chain in the list to open it in the Action Chain editor.
3. Drag **Call REST Endpoint** from the Actions palette into the action chain.

You can drag the action onto the Add icon (➕) in the action chain, or between existing actions in the chain. The property editor opens when you add the Call REST Endpoint action to the action chain.
The Call Rest Endpoint action is badged with a warning icon when no endpoint has been selected.

4. Click **Select Endpoint** in the property editor.

The Select Endpoint window displays a list of the endpoints that are available in your application. Each business object and service usually exposes multiple endpoints. The endpoint that you select will depend upon the function of the action chain. The endpoint that you select will also determine the properties that you will need to specify for the action, for example, input parameters.
5. Select an endpoint from the list. Click Select.

6. Edit the action’s properties in the property editor.

   The property editor is displayed when the action is selected on the canvas.
Call REST Endpoint

Id *

callRestEndpoint1

Label

Description

Endpoint *

Select

businessObjects/get_Customer

Input Parameters

Assign

A Customer_Id *

onlyData

NOT MAPPED

Parameters

Assign

A filePath

requestTransformOptions

NOT MAPPED

Content Type

application/json

File Content Type

image/jpeg

Response Body Format

Select Format

Response Type

Select Type
7. Optional: If the REST call requires input parameters, click **Assign** in the property editor to open the window to map the variable for the input value to the action’s parameter. Click **Save**.

You map variables to parameters in the Call REST Endpoint window by dragging the variable in the Sources pane onto the parameter in the Target pane. In some cases you might need to make multiple mappings. To delete a line mapping a variable to a parameter, place your cursor on the line and then right-click to open a Delete option. You can select the parameter name to view the expression for the mapped variable.

If a suitable variable does not exist, you can create the variable in the Variables editor and edit the action later.

8. Optional: If the REST call requires other parameters, click **Assign** in the Parameters section to open the window for mapping the variables to the action’s parameters. Click **Save**.

If the structure and names of attributes match they can be automapped. The mapping can also be done individually.

9. Optional: Specify any other parameters that may be required for the action.

After adding the Call REST Endpoint action, you can continue by adding more actions to the action chain, or by invoking the action chain from an event. If the REST call has
a result, you might want to add a Fire Notification action, or add Assign Variables to the chain and map the result to a page variable.

Add a Fire Custom Event Action

You add a Fire Custom Event action to invoke a custom event that you have defined in your application.

A custom event can be defined in an application, flow or page, and can be used to perform some action, such as navigating to a page. A custom event can carry a payload that you define when you create the event. The Custom Events tab in the Events editor displays a list of the custom events available in the context.

To add a Fire Custom Event Action:

1. Open the Actions editor for the page or application.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag Fire Custom Event from the Actions palette into the action chain.
4. Type a name for the action in the Id field in the property editor.
5. Select the Custom Event in the Event Name drop-down list.
   The drop-down list displays the custom events that are available in the current context.
6. Click Assign to open the Mapper and define the event’s payload.
   The event payload depends upon how the custom event is defined. You can use the Mapper to map the payload to a source, such as a page variable, or define a specific value or expression.

Add a Fire Notification Action

You add Fire Notification actions to display notification messages in the browser window.

You add a Fire Notification action for each message that you want to add to the action chain. When you add the action you select the type of result that will cause the notification to fire by selecting a notification type. For example, if an error occurs when performing an action in the chain, the notification will appear if you select error as the notification type. If you also want a notification to appear when the action succeeds, you will need to add another Fire Notification action and set the notification type to confirmation.

To add a Fire Notification action to an action chain:

1. Open the Actions editor for the page or application.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag Fire Notification from the Actions palette into the action chain.
   You can drag the action onto the Add icon ( + ) in the action chain, or between existing actions in the chain. The property editor opens when you add the action to the chain.
4. Type a name for the action in the Id field in the property editor.

5. Type the message to display in the **Message** field.
   
   The message you enter can be a static string (The name was updated.) or can contain variables ({{ 'Could not create new Contacts: status ' + $chain.results.createContacts.payload.status }}).

6. Select a **Notification Type** to specify the type of result that will fire the notification, and a **Display Mode** to specify how the notification is displayed.

**Add a Navigation Action**

The Actions palette contains a Navigation section that provides actions for navigating to the previous page, a specific page in the application or an external URL.

When you add a Navigate action to an action chain to navigate to a specific page in your app, after you select the target page you can specify an input parameter to specify the id of a record.

To add a navigation action to an action chain:

1. Open the Actions editor for Page.
2. Click the action chain in the list to open it in the Action Chain editor.
3. Drag an action from the Navigation section of the palette into the action chain.
The Navigation section contains **Navigate**, **Navigate Back** and **Navigate to URL** actions.

You can drag the action onto the Add icon ( ) in the action chain, and typically this action will be the final action in the chain. The property editor opens when you add the Navigate to Page action to the action chain.

4. Specify the details of the page or URL in the property editor, if required. If you add the **Navigate Back** action you do not need to specify any details.. If you add the **Navigate to URL** action you need to provide the URL in the property editor. If you add the **Navigate** action you need to perform the following steps.

   a. Click **Select Target** in the property editor.

   b. Select the context of the target in the Select Target window.

The targets you can select will depend upon the context of your action chain. The target for a Navigate action can be one of the root (shell) pages of the application, another page in the current flow, or a different flow that is defined in the current page. One or more of these options might not be valid targets for your action chain. After you choose the context of your target you can choose from the available targets in that context.
c. Select a page in the list. Click **Select**.
   If the page you select requires an input variable you will need to map a page variable to the action’s Input Parameter.

d. Type a name for the action in the Id field in the property editor.
   When the action is selected on the canvas, you edit the action's properties in the property editor.
e. Optional: If the page requires input parameters, click **Assign** in the Input Parameters section to open a window to map the variable for the input value to the action's parameter. Click **Save**.

In the window, you map Sources to Targets by dragging the variable in the Sources pane onto the parameter in the Target pane. You can click the parameter name to view the expression for the mapped variable.

If a suitable variable does not exist, you can create the variable in the Variables editor and edit the action later.

### Design Pages

Each page in your application is composed of various artifacts. When you open a page, there are various editors for examining and modifying the artifacts used in the page.

**Topics:**

- Use the Page Designer
- Work with Pages and Page Layouts
- Add Components to Pages
- Filter Data Displayed in a Component
- Add Page Functionality Using Quick Starts
- Work with the Functions Editor
Use the Page Designer

The Page Designer provides a page editor, a Components palette and a Page Structure view for designing the layout of a page.

When editing a page’s layout, you use the Property Inspector in the Page Designer to modify component properties and also to open any Quick Starts that can be used with the component that is selected in the page editor.

The Page Designer Toolbar

When a page artifact is open in the Page Designer, the toolbar provides tools for modifying the Page Designer’s behavior and appearance.

<table>
<thead>
<tr>
<th>Toolbar Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🌟</td>
<td>Toggles the Component palette</td>
</tr>
<tr>
<td>🏛️</td>
<td>Toggles the Page Structure view</td>
</tr>
<tr>
<td>🔥</td>
<td>Reloads the page</td>
</tr>
<tr>
<td>🔍</td>
<td>Opens a dialog box for entering input parameters for the page.</td>
</tr>
<tr>
<td>👤</td>
<td>Opens a dialog for selecting the user roles that are used when previewing pages in Live mode.</td>
</tr>
<tr>
<td>📸</td>
<td>Opens a menu for selecting the screen size represented by the canvas.</td>
</tr>
<tr>
<td>🔍 100%</td>
<td>Opens a dialog box for changing the magnification of the canvas.</td>
</tr>
</tbody>
</table>
The Page Editor

The Page Designer contains a page editor that you use to design, view and edit your application's pages. The page editor has three modes: Live, Design and Code. You switch between the modes using the mode toggle in the Page Designer's toolbar.

When editing a page, you use the following modes of the page editor to navigate the pages in your application and edit a page's layout and code:

- **Live mode.** The Live mode displays the page as it appears when published, and the elements in the page also behave as they do when published. You use Live mode to interact with the pages in your application to navigate to different pages, create and modify business objects and confirm that your application is behaving as you expect.

- **Design mode.** The Design mode contains a canvas area that you use to place and position components in the page. You can hold down the ctrl key to temporarily switch between Live mode and Design mode.

- **Code mode.** The Code mode contains an editor that you use to edit the page's code. In Code mode, you can drag components from the Components Palette and drop them directly into valid places in the code in the editor. When you use the
Page Structure view to edit and reposition elements, the changes are automatically reflected in the code.

The Components palette, Page Structure view and Property Inspector are available and can be used the same way in each mode. When you select an item in one mode, the item remains selected when you switch to a different mode. For example, when you switch from Design to Code mode, the source code of a component selected on the canvas will be highlighted in the code editor. All the modes are synchronized, so that changes you make in the Property Inspector or Page Structure view are visible when you switch to a different mode.

The Components Palette

The Components Palette contains UI components and organizational elements that you use to build your pages.

You add an element to a page by dragging it from the palette onto the canvas. You can use the Components Palette button (.getInteger(552, 562)) in the Designer to toggle the Components Palette.
The Page Structure View

The Page Structure view provides a structural view of the components on the canvas. You can use the Page Structure view to easily see the layout components used in the page and the components that they contain, and to reposition components within the page’s structure.

You use the Page Structure button ( ) in the Designer to toggle the Page Structure view. When you select a component in the Page Structure view, the component is also selected on the canvas and its properties are displayed in the Property Inspector. You can organize and re-position components in the page by dragging them into position in the Page Structure view and by dragging them directly onto the canvas. You can select multiple components to simultaneously re-position them in the Page Structure view, for example, to move them into a new container.

You can also drag components from the Components palette or canvas into the Page Structure view to add them to the page. When you hover your cursor over a component in the Page Structure view, a tooltip appears displaying details about the component, for example, its id, classes or hints about its content. Right-clicking a component in the Page Structure view opens a popup menu with actions you can perform on the component, including deleting it, surrounding it or to selecting its parent component.
You can use the Page Structure view’s options menu to choose if the component details are also displayed in the structure view. You can also enable Show empty slots in the menu to display the location of empty slots in the Page Structure view. When the empty slots are visible, you can easily locate the slot where you want to drop a component. When they are not visible, you can still easily locate available slots by pausing over a component node when you are dragging a component into the Page Structure view. If the component node has slots, a popup menu opens that lists the available slots, and you can then drop your component into the desired slot in the popup menu.

The Property Inspector

The Property Inspector displays the properties of the component that is currently selected on the canvas or in the code editor.

Depending on the component that is selected, the Property Inspector might have additional tabs for modifying the component’s attributes or its behavior. When you add a collection component, such as a table or list to the canvas, the Quick Start tab in the Property Inspector contains a list of Quick Start wizards that can help you add some actions and components that are typically associated with the component, such as mapping the collection to data and adding Create and Detail pages. You can use the toggle button in the toolbar to hide or display the Property Inspector.

Activate Role-Based Application Preview

Application roles can be used to secure access to components and data in your application. You can preview how the role-based security settings of your application will affect the layout of pages and the behavior of your application.

Role-based security enables you to configure the data and pages that are accessible to users based on the user’s role. The security settings for your components and business objects will determine the components and data that are visible to users, how users can navigate between the pages and the layout of the pages in your application. See Allow Anonymous Access.
When viewing pages in the Page Designer, by default the pages that you see are not affected by the security settings. To see how your security settings will affect your application, click **Who am I?** in the Page Designer toolbar to open the Who am I? role selector.

The Who am I? role selector displays a list of the application roles defined in the application. By default, all the pages and data are accessible because all roles are selected and active. You use the Who am I? role selector to select the roles that are currently active.

Multiple roles can be active simultaneously, but at least one role must always be active. For a more accurate representation of how your application will look and behave for a specific role, you might want to deactivate all the roles except the one you are interested in. For example, if Anonymous User and Authenticated User are both active, you are seeing the application as it appears to users that are signed in and to users that are not signed in. By deactivating the Authenticated User role you will see and experience the application as an Anonymous User would see it. An anonymous user that was not granted rights to view data in a business object would not see any data if they visited the collection page for the object. Additionally, if View access was not granted, links in the UI to the collection page would be hidden from the anonymous user.
Work with Pages and Page Layouts

You create new pages in the Navigator and then use the Page Designer to edit the pages.

All the pages in your application are created within one of the flows in your app, with the exception of root pages. A root page is a special type of page which provides the entry point for your app, and contains the app’s header and footer elements, and a container for the app’s flows. You set the app’s default root page in the General Settings tab in the app artifact’s Settings editor, and the root page’s default flow in the General Settings tab in the root page’s Settings editor.

Create Pages

You create new pages using the Create Page dialog box that you open from the Navigator.

To create a page, you click Create Page (+) next to the flow artifact or root pages node in the Navigator.

To create a new page in a flow:

1. Open the Web Applications or Mobile Applications browser in the Navigator.
2. Expand the node of the app where you want to create the page.
3. Expand the flows node.

When you expand the flows node you see the flows in your app. Your app contains one or more flows that are created by default when you create the app. You can expand the flows node to see the pages in the flow.

4. Click the Create Page icon (+) next to the flow where you want to create the page.
5. Type the name of the new page in the Id field of the Create Page dialog box. Click Create.
After a page is created, you can change the page title in the page’s Settings editor.

Edit the App Header and Footer

You can edit the header and footer elements of your app by modifying the root page in the Page Designer.

The default entry page for your app is a root page that contains a header, a footer and a flow container where a flow is embedded. A root page named `shell` is created by default when you create the app, and the `main` flow is set as the default flow displayed in the flow container in the root page.

To edit the app’s header and footer:

1. Select the page under the `root pages` node in the Navigator to open it in the Page Designer. The default root page is named `shell`.
2. In the Design view of the Page Designer, select the component in the header or footer that you want to edit.
   You can select the component on the canvas or in the Page Structure view.
3. Edit the component’s properties in the Property Inspector.

About Containers and Layout Components in Pages

You design and edit the pages in your app in the Page Designer. Each page has a preferred default layout, and you can add other layout components to the page to help you organize the components when designing pages.

When you create a page it automatically has a preferred page layout container. This page layout can be either a Grid or a Flex layout, and you choose which one you want to use in the Property Inspector. Every component that you add to your page is placed in a row in the page’s Grid or Flex layout, or in a layout component that you have placed in the page layout.

The Grid layout has a pre-defined 12–column grid and rows for aligning elements when you position them. You can add layout containers and components to rows to help organize and layout components in the page. You can place multiple containers within a single row, or automatically create a new row by placing a container or layout component above or below an existing row. For each row, you can modify its display settings in the Property Inspector to control the layout of the components within the row. The pages in your application incorporate responsive design to resize gracefully based on the size of the display area of the device.

Containers and layout components can be nested to organize components and to position them precisely.
You add layout containers and components to a page by dragging them from the Layout category in the Components palette and placing them on the canvas or in the Page Structure view. The Layout category has various types of containers and layout components, and some are specifically designed to help you with common design styles, such as different ways to display navigation links or data.
You can use the Page Structure view in the Page Designer to see the components in each row in your page and to reposition them. When you select a component in the Page Structure view, it is also selected on the canvas and its properties are displayed in the Property Inspector. You can also drag components from the Components palette directly into the Page Structure view.
Layout components are pre-defined Oracle JET components and styles that can be applied to page elements. You can modify the properties of each component in the Property Inspector. The following table describes some of the commonly-used containers and layout components available in the Components palette. For examples of how they look and can be used, see the Layout & Nav section in the Oracle JET Developer Cookbook.

<table>
<thead>
<tr>
<th>Container Components</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flex Container</td>
<td>The flex container is a flexible container which is useful for responsive designs that optimize the use of the available space.</td>
</tr>
<tr>
<td>Grid Container</td>
<td>The grid container is a 12-column grid that is useful when you want to align components precisely according to the grid.</td>
</tr>
<tr>
<td>Bar Container</td>
<td>The bar container is a three-section layout containing a start and end section sized to its content and a middle section that stretches.</td>
</tr>
<tr>
<td>Form Layout</td>
<td>The form layout is optimized to display the label and input pairs commonly used in forms.</td>
</tr>
<tr>
<td>Masonry Layout</td>
<td>The masonry layout is a responsive grid of tiles containing arbitrary content. You can specify the size of each tile in the Property Inspector.</td>
</tr>
</tbody>
</table>
Add a Container or Layout Component to a Page

You add a layout to a page by dragging it from the Layout category in the Components palette and placing it on the canvas or into the Page Structure view.

You use the Property Inspector to select the type of layout used to position components on the page and within page rows. Pages can have a Flex layout or a Grid layout. You can combine layout types in your page by creating new rows in the page, placing multiple layouts within rows and by nesting layouts. Each row in a page can have a Flex, Grid or Bar layout. When you drag an element onto the canvas, some elements will expand to fit all the available columns in the row. Other elements have a default column span that you can adjust.

To add a container or layout component to a page:

1. Drag the container or layout component from the Components palette and place it on the canvas.

   When a container is dragged onto the canvas, the locations where the component can be placed are highlighted on the canvas.

   If you do not place the component in an existing row, a new row containing the component is created when you place it on the page.

2. Select the component and modify its properties in the Property Inspector.

   You can select components on the canvas or in the Page Structure view.
You can drag additional components into the container, or place them above or below an existing row to create new rows.

Add Components to Pages

You build pages in the Designer by dragging components from the Components palette and positioning them on the canvas. After you position a component on the canvas, you use the Property Inspector to edit its properties.

Topics:
- About Page Components
- Add a Component From the Components Palette
- Add a Component Using Code Completion in the Code Editor
- Manage Component Visibility Using Conditions
- Add Events to Components
- Add an Image to a Page
- Add an Icon Component to a Page
- Add a Custom Style to a Component

About Page Components

You use page components to build the layout of your pages and to add elements that can be used to display content or accept input from a user.

The elements in the Components palette are organized by type. For each type, there are some rules that determine where they can be used on the canvas, as well as the types of pages where they can be placed. An error message is displayed when you try to place a component on the canvas where the component is not allowed.

The following table describes the types of components in the palette and provides some rules governing their placement.

<table>
<thead>
<tr>
<th>Component Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>The components in the Common category are basic page elements for displaying content and providing navigation. You can use these components for many uses, including to display static content or values from variables, and trigger action chains.</td>
</tr>
<tr>
<td>Field</td>
<td>The components in the Field category are used for building forms, for example, a page for editing a business object. Field elements must be associated with a business object and positioned within a form page. When you add a field to a form you can choose to create a new field in the associated business object or you can reference an existing field in the business object.</td>
</tr>
</tbody>
</table>
### Component Type

<table>
<thead>
<tr>
<th>Component Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>The components in the Collection category are used to display data from business objects, for example, a table with a row for each business object. To display data in a collection component you need to bind the collection to an endpoint.</td>
</tr>
<tr>
<td>Layout</td>
<td>The components in the Layout category are containers for organizing elements on the page, for example, to create panels and to align components. Layout components can be placed on any type of page.</td>
</tr>
<tr>
<td>Chart</td>
<td>The components in the Chart category are used to generate charts that visually represent business object data. Charts can be added to any page. When you add a chart to a page, you select the business object that you want the chart to represent. You can choose any of your business objects. The charts are independent of the business object associated with a page.</td>
</tr>
<tr>
<td>Gauge</td>
<td>The components in the Gauge category are used to add a visual representation of a single value in a business object. You can add a gauge to any page. Gauge components can be interactive and can be used in an Edit or Create page to update the data for a field.</td>
</tr>
<tr>
<td>Picker</td>
<td>The components in the Picker category are used to add UI elements that can be used to select dates, times and for choosing colors.</td>
</tr>
<tr>
<td>Custom</td>
<td>The Custom category contains web components that you have imported into your application. When you drag the component onto the canvas, you can use the Property Inspector to specify the component's properties. Web components might have properties that are specific to the component. If your instance is associated with the Component Exchange, you can locate components in the exchange and add them to the palette.</td>
</tr>
</tbody>
</table>

Use the Property Inspector to configure properties when a component is selected. Component properties are organized in tabs in the Property Inspector. The properties depend on the type of component.

### Component Properties

<table>
<thead>
<tr>
<th>Component Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>The General tab is used to edit the most important properties of the selected component, such as layout properties. The properties vary according to the component.</td>
</tr>
</tbody>
</table>
### Component Properties

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
</tr>
<tr>
<td>The Data tab is used to edit the properties which are expected to be bound to data. The General tab and All tab also contain properties that can be bound to variables and expressions.</td>
</tr>
<tr>
<td><strong>Events</strong></td>
</tr>
<tr>
<td>The Events tab is used to bind a component’s events to trigger action chains</td>
</tr>
<tr>
<td><strong>All</strong></td>
</tr>
<tr>
<td>The All tab is used to edit more advanced component properties.</td>
</tr>
<tr>
<td><strong>Quick Starts</strong></td>
</tr>
<tr>
<td>The Quick Starts tab displays a list of Quick Start wizards available for the component.</td>
</tr>
</tbody>
</table>

### Add a Component From the Components Palette

You add a component to a page by dragging it from the Components palette and dropping it onto the canvas or into the Page Structure view.

The Components palette in the Page Designer contains many of the Oracle JET components that you can add to your page. After adding a component you can define the component’s behavior by editing its properties. The properties will vary according to the type of component.

To add a component from the Components palette:

1. Open the page in the Page Designer and confirm you are in Design mode.
   - In Design mode, the window contains a canvas displaying the page’s layout, a Components palette containing a list of components, and a Page Structure view that displays a structural view of the page’s components.

2. Drag the component from the Components palette and drop it into position on the canvas or in the Page Structure view.

![Components Palette](image)

After you add the component to the page, the component name and properties are added to the page’s HTML file. You can edit its properties in the Property Inspector.
when the component is selected, or in the HTML file in Code mode. It is sometimes easier to locate, select and position components in the Page Structure view.

Add a Component Using Code Completion in the Code Editor

For components not contained in the palette, you can open the page in Code mode and use code completion to select an Oracle JET component and define its properties.

If you want to add an Oracle JET component that is not in the Components palette, you can use the Code mode in the Page Designer to edit the page’s HTML source to add the component to your page. The code editor provides code completion that can help you locate JET components and view details about each component’s properties.

To add a component to a page in the Code editor:

1. Open the page in the Page Designer.
2. Click Code to open the page in the code editor.
3. Insert your cursor in the code where you want to add the component.
4. Start typing the tag for the component you want to add and use the editor’s code completion to help you add the tag for the component.

For example, when you start typing `<oj-a` in the editor, the code completion window appears with a list of component tags that match the text you type. You can select a tag in the window to add it to the HTML file.

5. Select the component in the list. Press Enter on your keyboard to add the tag.

After the tag for the component is added to the HTML code, you can define the properties of the component using the code completion in the editor or use the Property Inspector.

Manage Component Visibility Using Conditions

You can use an `oj-bind-if` component to surround other components and set test conditions to control if the components are displayed on the page.

You can add an `oj-bind-if` component to your page by dragging it from the Components palette or by using the popup menu in the canvas or Page Structure view. After adding the component, you set the conditions for the component by entering an expression in its Test property in the Property Inspector. In the Page
Structure view and in Code mode you can see the components that are surrounded by a `oj-bind-if` component.

The content surrounded by `oj-bind-if` is displayed in the page if the test condition is true. For example, you can use an expression that evaluates if the value of a page variable does not equal a predefined value. The surrounded content is displayed if the values are not equal (the expression is true), and hidden if the values are equal.

You can use the Temporary Override property of `oj-bind-if` to temporarily set the result of the test condition to True or False. For example, when designing your page, if some content is hidden on the canvas because the test condition result is False, you can use Temporary Override to set it to True so that the content is visible on the canvas, or select Off to temporarily disable the test. The setting is temporary and will revert to Off when you reload the page. You can set the Temporary Override in the Property Inspector or in the component's popup menu that you can open from the Page Structure view or canvas.

You can use `oj-bind-if` to dynamically control what components a user sees based on the user's role, for example, to hide buttons or navigational elements, by using `application.user.roles.role_name` in the expression. You can set restrictions on entire pages, or just on certain components in a page.

The visibility of components based on roles is only enforced on the client, and the value of the role could be changed in the client. For this reason, in addition to hiding components, you should also use role-based security to secure the application and the data in a page.
To use an `oj-bind-if` component to control when a component is displayed in a page:

1. In Design mode of the Designer, locate the component that you want to control dynamically.

2. Right-click the component on the canvas or in the Page Structure view and select Surround > If in the popup menu.

3. Select the `oj-bind-if` component on the canvas or in the Page Structure view and open the Property Inspector.
   
   The component has the Test property that you use to set the condition. The default expression is `[[True]]`.

4. Enter the condition for the component in the Test field.
   
   You can type the condition in the Test field, or use the Expression Editor to help build an expression using available variables.
Add Events to Components

You define component behavior by adding types of events to the component and then selecting action chains that are triggered by the event.

The type of events that can be assigned depend on the type of component. Multiple events can be assigned to a component, and each event can trigger multiple action chains. When you add the event you can select an existing action chain or create a new one.

To add an event and action chain to a component:

1. Select the component.
   Typically you want to assign events to elements such as buttons, menus and fields in form components. You can select the component on the canvas, in the Page Structure view or in Code view.

2. Open the Events tab in the Property Inspector.
   The Events tab lists any events that are already assigned to the component.

3. Click + New Event and select the Quick Start option to use the suggested event or New Custom Event to select an event to start the action chain.
   The Select Event dialog box displays a grouped list of types of events that can be assigned to the component. You can select an event type in the dialog box to display its description.
4. Select the type of event you want to trigger the action chain. Click Select.

After selecting a type of event you need to select the action chain that the event will trigger.

5. Select an action chain from the list. Click Select.

Alternatively, you can click New Action Chain to create a new action chain.

After an event is assigned to a component you can open the Events tab to edit the type of event, the action chain and any input variables.

To modify an existing component event, located the event in the Events tab and click Select to change the type of event assigned to a component, or click + New Event to add a new type of event. To select a different action chain, click Select in the Events tab, or click the name of the action chain to open it in the editor.
Add an Image to a Page

To add an image to a page, you position an image component on the canvas and then specify the path to the image you want to add in the Property Inspector. You can select an existing image in the Image Gallery or import a new image from your local system. You use the tabs in the Property Inspector to specify the image’s display properties, the path to the image source and any component events for triggering action chains.

The images used in pages in your app are stored in an images resource folder. The app contains a default images resource folder, and each flow in your app can also contain an images resource folder. When adding an image to a page, you can use the Image Gallery to select an image that was already imported, or add a new image to the Image Gallery directly from the component's Data tab in the Property Inspector. When you add an image to the Image Gallery, you can choose to import an image as an application resource or a flow resource. If you want to select an image that was already imported into the app, you can click the Image Gallery button in the Data tab and use the Image Gallery dialog box to locate and select the image. When you select images from the Image Gallery, you can select application resources or flow resources.

When you drag an image into the drop target area in the Data tab, the image is imported into the images folder for the flow, and the path to the image location is added to the Source URL field. For example, the path to an image stored in a flow’s images folder will be similar to the following: `{{ $flow.path + 'resources/images/myimage.png' }}`.

To ensure that the relative path to the image resource is built correctly when the app is staged and published, the path to the image in the Source URL field needs to include the builtin variable $flow.path or $application.path to identify the location of the resource folder.

To add an image to a page:

1. Open the page in the Page Designer and drag an image component from the Components palette onto the canvas.
2. Open the component’s General tab in the Property Inspector and specify the height, width and alt text for the image.
3. Open the component’s Data tab in the Property Inspector.

   The Data tab contains a Source URL field that contains the path to the image. You can use a string or a variable to specify the path to the image source.
4. Drag your image into the drop target area in the Data tab.
5. Open the component’s All tab in the Property Inspector to view and edit all of the component’s attributes.

Add an Icon Component to a Page

Oracle Visual Builder includes a set of icons that you can add to your pages using the Icon component in the Components palette.

After adding the Icon component to the canvas, you use the Property Inspector to select an icon from the Icon Gallery, specify the display properties of the icon and configure any component events for triggering action chains.

To set the display size of the icon, you apply inline CSS styles to the component by specifying values in the component’s style field in the Property Inspector. To set the size you must explicitly set the width, height and font-size to the same pixel value (for example, width:50px; height:50px; font-size:50px).

To add an Icon component to a page:

1. Open the page in the Page Designer and drag an Icon component from the Components palette onto the canvas.
2. Select the component on the canvas and click the Image button ( Image ) in the General tab of the Property Inspector.
3. Select the icon in the Icon Gallery window. Click Select.
4. Specify the size of the icon in the **style** property in the **All** tab of the Property Inspector.
5. Specify additional properties or component events in the Property Inspector.

Add a Custom Style to a Component

You can add a style to a component and then define the style in your app's stylesheet.

You can apply style classes to page components to control how they are displayed. Some classes are already pre-defined in the app and are automatically applied to components when you add them to a page. Specific pre-defined style classes are applied to many Oracle JET components to ensure they display correctly and consistently. For example, if you look at the HTML for a Header component in a page's Code editor, you might see the following style classes applied to an `h1` element: `oj-flex-item oj-sm-12 oj-md-12`. Pre-defined style classes used by Oracle JET components are prepended with `oj-`. As a general rule, you should not modify the pre-defined classes or remove them from components.

When defining and adding a custom class to a component, you should exercise caution to ensure that your class does not conflict with the pre-defined classes already applied to the component.

You can define your custom style classes in the `app.css` stylesheet of your app. An empty `app.css` stylesheet was created in your app by default and the link included in the header of the app's pages. You can apply classes to components in the Property Inspector in the Design view of the Page Designer or in the page's Code editor.

To add a custom style to a component:

1. Open the page in the Page Designer and locate the component that you want to modify with a custom class.
2. Type the name of the custom class to apply it to the component.

When you select the component in the Design view of the Page Designer, you can add the name of the custom class in the **class** property field, which is usually located under the General Attributes category in the All tab of the Property Inspector. You can also add the name of the class to a component directly in the page's Code editor.

3. In the Navigator, expand the **css** node in your app's **resources** folder and click **app.css** to open the stylesheet in the editor.

4. Define the class in **app.css**.

```css
.myredlabel {
  color:red;
}
```

Reload the page in the Page Designer to see the class applied to the component.

**Filter Data Displayed in a Component**

When you bind a component to an endpoint, you can filter the data displayed in the component by defining filter expressions in the Service Data Provider used to retrieve the data. You can use expressions and static content to set the filter criteria values and Oracle JET operators to define the logic.
To display data in a collection component such as a list or table, you usually bind the component to an endpoint using a variable that is assigned the built-in Service Data Provider (SDP) type. This variable is created for you when you use the Add Data Quick Start to bind the component to an endpoint. The SDP type manages requesting and receiving data from an endpoint, and supports a filterCriterion property that can be configured to filter the data stored in the variable and displayed in the component. The filterCriterion structure can be used to express many of the filter expressions you might want to use when retrieving data. For more details, see Service Data Provider.

Note:

For more advanced filtering you can write JavaScript filtering functions that you can call from an action chain. See Work with the Functions Editor and Add a Call Module Function Action.

You build a filter expression by defining the properties of the three filterCriterion attributes (attribute, op, value). The filter expression is converted into an appropriate \"q\" query string when sent to the endpoint. You can make complex filters by combining multiple filter expressions. You can create a filter expression using the Assign Variables window of an action, or you can edit the JSON file where the action is defined (for example, main-start-page.json). The following table describes the filterCriterion attributes that you define in a filter expression.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>Name of the attribute. Typically this is the name of the field that the filter will process.</td>
</tr>
<tr>
<td>op</td>
<td>Supported Oracle JET operator. Common operators are, for example, $co (The entire operator value must be a substring of the attribute value for a match.), $eq (The attribute and operator values must be identical for a match.) and $ne (The attribute and operator values are not identical.). The operator $regex is not supported. For a list of Oracle JET operators, see <a href="https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj.AttributeFilterOperator.html">https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj.AttributeFilterOperator.html</a>.</td>
</tr>
<tr>
<td>value</td>
<td>Value of the attribute. This is the value that is used to filter the request. This value can be mapped to an expression (for example, a page variable) or a static value (for example, a string or number).</td>
</tr>
</tbody>
</table>

You can define filterCriterion attributes by editing the SDP properties in the Variables editor, or you can build a filter function in the page using variables, components and action chains. For example, you can create a filter for a collection such as a table using filterCriterion and use a page variable to store a string that a user enters in an input field. When the SDP sends a request to the endpoint, the filter processes the request and only the records that meet the filter criteria are returned and displayed.
Create a Filter for Component Data

You filter the data displayed in a component by defining the `filterCriterion` property used when calling the Service Data Provider to retrieve the data. You can use the Filter Builder to help define the filter criteria values and Oracle JET operators used to define the logic of the filter.

When you use the Add Data Quick Start to bind a collection component to a data source, you can use the Filter Builder in the Define Query step to filter data that you do not need to retrieve. For example, you can build a filter to only retrieve the business objects where the value of a field named "Active" equals "true", or equals some page variable.

To assign a page variable as a filter criterion in an action chain:

1. Create a page variable (for example, `filterVar`) to store the filter string.

   The value of the page variable can be pre-defined (for example, an input parameter), or you can bind it to a page component such as an Input Text or Combobox component to allow users to enter text strings, numbers or select from a list.

2. Create an action chain or open an existing action chain.

   The action chain will use the Assign Variables action to map a page variable to the `value` attribute in the filter expression. The action chain can be triggered by a component event or a page event.
Tip:

To create a filter using an Input Text component, you could use the Quick Start in the component's Events tab to add a Property Change event to the component. The Quick Start will automatically create a new action chain that is triggered by the event.

3. Add the Assign Variables action to the chain.

You configure the Assign Variables action to assign values to each of the attributes of the filter expression.
4. Click Assign in the action property editor to open the Assign Variables window.

5. Expand the Service Data Provider in the Target pane and select filterCriterion to open the Filter Builder.

The Filter Builder contains a Builder tab where you build expressions with the help of drop-down lists, and a Code tab where you build or edit them in a code editor.
Alternatively, you can expand the filterCriterion node in the Target pane and build your filter by specifying values for `attribute`, `op` and `value` individually.

6. In the Filter Builder, create a condition by specifying the Attribute, Operator and Value.

When you create an filter in the Builder tab, you create one or more filter conditions by selecting items in the Attribute, Operator and Value drop-down lists. You can add more conditions to build more complex filters.

You can use the Code tab to view and edit the filter. After defining a condition in the Builder tab, in the Code tab you will see that it contains an `attribute`, `op` and
value. For example, your definition of `filterCriterion` might have two conditions and be similar to the following:

```json
{
  "op": "$and",
  "criteria": [
    {
      "op": "$eq",
      "attribute": "name",
      "value": "{{ $page.variables.filterVar }}"
    },
    {
      "op": "$eq",
      "attribute": "id",
      "value": "{{ $page.variables.idVar }}"
    }
  ]
}
```

In this example, the `value` attribute (`$page.variables.customerListSDP.filterCriterion.criteria[0].value`) is mapped to a page variable expression (`$page.variables.filterVar`), the Oracle JET operator is "$eq" (it must include the dollar sign ($$)), and the `attribute` value is the name of the field that you want to filter on.

### Add Page Functionality Using Quick Starts

The behavior of pages is determined by the actions chains and events associated with pages and page elements. For common use cases, Quick Starts can help you create the artifacts, such as variables and action chains that are used to build functionality in pages.

**Topics:**

- About Quick Starts
- Add Data to a Table or List
- Add a Create Page With a Quick Start
- Add an Edit Page With the Quick Start
- Add a Details Page With the Quick Start

**About Quick Starts**

A Quick Start is a tool to help you create artifacts and action chains needed to add runtime functionality to an application.

A variety of Quick Starts are available in the Page Designer to help you perform some of the tasks required to add common application functions and behavior, for example, mapping a list to a data source, or adding a page for creating new records. To add functionality you need to create the artifacts that are used to perform the function. Depending on how complex the behavior is, adding the functionality might involve creating several variables, types, action chains and page events. If there is a Quick Start for your task, you can use it to quickly create many of the artifacts for you.
When you are designing pages in the Designer, the Quick Start tab in the Property Inspector will display a list of Quick Start tasks that are typically used to add functionality or behavior to the selected component and the type of application that you are creating. The Quick Start tasks are based on common tasks that developers need to perform when creating applications.

### Add Data to a Table or List

Use the Add Data Quick Start to populate a table or list in a page with data from a business object.

When a page with a collection component loads, a request to get data is automatically sent to an endpoint, and the response is mapped to the fields in the collection component. You will typically choose a data source that provides a **GET MANY** endpoint.

The Add Data Quick Start does the following for you.

- The Quick Start automatically modifies the collection component to add the fields necessary to display the fields in the data source that you selected. Each field is mapped to the corresponding attribute of the variable bound to the component.
The Quick Start creates a variable that is bound to the collection component. For the business object Contact, a new page-scoped variable named contactListServiceDataProvider that stores details about the endpoint, the request and the response type. When the collection is selected on the canvas, you can see the variable bound to the component in the Data field of the Data tab of the Property Inspector.

You can see the details of the new variable if you open the Variables editor of the page. In the General tab of the Variable Information pane, you can see the ID of the variable, the type and the endpoint that is called. The variable's type is Service Data Provider, a specific type that is designed for variables that are used to send a request to an endpoint.
The Quick Start creates a page-scoped type that describes the data structure of the response. The fields in the response are mapped to the field in the component. When you select the new variable in the Variables editor, in the Variable Information pane you can see that the type for the response is a new custom type named \textit{getallContactResponse}. The data structure defined by the type is based on the fields in the endpoint that you selected in the Quick Start.

The new custom type is added to the list of types available in the page. You can see the details of the new type in the Types tab of the page’s Variables editor.

Use the Add Data Quick Start

To use the Quick Start you must first add an endpoint for a service connection or business object to your visual application. After adding the endpoint you can step through the Add Data Quick Start to quickly create the artifacts needed to bind a table or list to the endpoint. The Quick Start will create a page variable for storing the data and a custom type that defines the data structure of the response to the request.

To bind an endpoint to a collection component:

1. Drag a table or list component from the palette onto the canvas.
2. Select the component and click \textbf{Add Data} in the list of Quick Starts.
3. Select the data source that you want to bind to the collection. Click \textbf{Next}.
4. Drag the fields you want to retrieve from the Endpoint Structure into the columns or template fields in your collection component.
   
   If you are binding data to a List component, you select a list template before binding the data from the endpoint to the fields.
5. Select the field to use as the Primary Key. Typically this is the \textit{Id} field. Click \textbf{Next}.  

Oracle
6. Define the parameters for querying the endpoint. Click **Finish**.

The collection is now bound to the endpoint you selected.

**Add a Create Page With a Quick Start**

Use the Add Create Page Quick Start to create a new page with a form that interacts with an endpoint to create a new object.

The Add Create Page Quick Start adds a Create button to the page with the collection. Clicking the Create button starts an action chain that navigates to a Create page containing a form for adding data. Clicking the Save button in the Create page starts an action chain that sends a request to the CREATE endpoint of the data source. The data in the page’s fields are stored in a variable that is mapped to the parameters of the request. If the request is successful, the user is navigated back to the page with the collection.

In the page with the collection component, the Quick Start does the following:

- Creates an action chain for navigating to a page.
- Adds a button component with an `ojAction` event that starts the action chain.

In the new Create page, the Quick Start does the following:

- Creates a page variable for storing the data for the new business object. The variable’s attributes are mapped to the parameters that are included in the request to the endpoint.
- Creates a variable type for defining the data structure of the variable.
- Adds a form with field components for the fields in the new business object. The fields are bound to fields in the new variable.
- Adds a Save button and Cancel button with `ojAction` events that start action chains.
- Creates an action chain to create the new business object. The action chain is started when the Save button is clicked.

The action chain sends a request to the CREATE endpoint of the business object. The data stored in the page variable is mapped to parameters that are sent as a request to the endpoint. The action chain includes actions that navigate to the previous page if the request is success or displays a warning if the request fails.
• Creates an action chain to navigate back to the previous page when the Cancel button is clicked.

Use the Create Page Quick Start

To use the Add Create Page Quick Start, you start from a page where a table or list is already bound to an endpoint. As you step through the Quick Start you select the fields that you want to include in the Create page when you create the new object. The Quick Start will add a button to navigate to a Create page with a form for adding data to create a new object.

When you create a new object, you will use a data source endpoint with a POST method. The source will already be pre-selected in the Quick Start based on the source bound to the collection.

To add a page to create a new business object:
1. In the page containing the collection component, select the component on the canvas and open the Quick Start tab in the Property Inspector.

2. Click **Add Create Page**.

3. Select the data source where you want to create a row. Click **Next**.

4. Drag the fields you want to add from the Endpoint Structure into the columns or template fields in your collection component. The Create page will include these fields in the form.

5. Modify the label for the button, the page title and the page name, if desired. Click **Finish**.

A new page is created with a form for creating a new business object.

**Add an Edit Page With the Quick Start**

Use the Add Edit Page Quick Start to create a page for editing the details of an object. Selecting an object in the component triggers a component event that stores the id of the selected object in a page variable. Clicking the Edit button triggers an action chain that navigates to an Edit page, and the id value stored in the variable is passed as an input parameter to the page. When the Edit page is loaded, a page event triggers an action chain that sends a request to the endpoint to get the data from the source, and the input parameter passed to the page is mapped to the input parameter required by the request. The response is mapped to a variable that is bound to the components in the page for editing the data.

Clicking the Save button in the Edit page starts an action chain that sends a request to the Update endpoint of the data source. The data in the page’s fields is stored in a variable and mapped to the parameters of the request sent to the Update endpoint.

In addition to creating the Edit page with a form containing the fields, the Quick Start creates various variables for the data and action chains to navigate to the page and call endpoints.

The Quick Start does the following in the page containing the collection:

- Adds a button to the page. An `ojAction` event is added to trigger an action chain.
- Adds a select event to the collection component that triggers an action chain.
- Creates a variable to store the id of the selected object.
- Creates an action chain that assigns the id of the selected object to a variable.
- Creates an action chain to navigate to the Edit page. The action chain passes the object id as an input parameter. Creates an action chain for navigating to a page.

The Quick Start does the following in the Edit page:

- Adds a form with fields bound to a variable.
- Adds a Back button. A click event is added to trigger an action chain.
- Creates an action chain that navigates back to the previous page triggered by an event on the Back button.
- Creates a page variable to store the object id as an input parameter.
- Creates a page variable to store the response from the endpoint. The Quick Start also creates a new Type that defines the structure of the variable.
- Creates an action chain that calls an endpoint when the page is loading and assign the response to a page variable. The action chain has an input parameter mapped to a page variable.

- Creates an action chain to update the business object. The action chain is started when the Save button is clicked.

The action chain sends a request to the Update endpoint of the data source. The action chain includes actions that displays a message if the request succeeds or a warning if the request fails.

The data stored in the page variable is mapped to parameters that are sent as a request to the endpoint.
Use the Add Edit Page Quick Start

When you have a page with a table or list component, you can use the Add Edit Page Quick Start to add an Edit button to the page that is enabled when you select an object in the table or list. Clicking the Edit button opens a page that displays a form with fields for editing the selected object.

You can open the Add Edit Page Quick Start from pages that use a table or list component to display a collection. The Quick Start is listed in the Quick Start tab in the Property Inspector when you select the component on the canvas. As you step through the Quick Start, you select the endpoint with the `GET` method (Get One) to request the data to display in the Edit page, and then the endpoint with the `PATCH` method (Update) where the request to update the data is sent. You also select the fields that you want to display in the Edit page.

To create an Edit page:

1. Select the table or list on the canvas.
2. Open the **Quick Start** tab in the Property Inspector, if not already open.
3. Click **Add Edit Page**.
4. Select the data source containing the GET endpoint to get the data you want to edit. Click **Next**.
5. Select the data source containing the UPDATE endpoint to send a request to update the selected record.
6. Drag the fields you want to include in the Edit page from the Endpoint Structure into the columns or template fields.
7. Specify the name of the button that will open the Edit page, and the title and name of the new page. Click **Finish**.
The page now has a new button that will navigate to a page that contains a form for editing the data of the object that is selected in the collection.

Add a Details Page With the Quick Start

Use the Add Detail Page Quick Start to create a page that displays the details of an object selected in a table or list.

After you use the Quick Start to add the Detail page, clicking the Details button opens a page that displays details of the selected object. Selecting an object in the component triggers a component event that stores the id of the selected object in a page variable. Clicking the Details button triggers an action chain that navigates to the Detail page, and the id value stored in the variable is passed as an input parameter to the page.

When the Detail page is loaded, a page event triggers an action chain that sends a request to the endpoint to get the data, and the input parameter passed to the page is mapped to the input parameter required by the request. The response from calling the endpoint is mapped to a variable that is bound to the components in the page that display the data.

In addition to creating the details page with a form containing the fields, the Quick Start creates various variables for the data and action chains to navigate to the page and call the endpoint.

The Quick Start does the following in the page containing the collection.

• Adds a button to the page. An ojAction event is added to trigger an action chain.
• Adds a select event to the collection component that triggers an action chain.
• Creates a variable to store the id of the selected object.
• Creates an action chain that saves the id of the selected object in a variable.
• Creates an action chain to navigate to the Detail page. The action chain passes the object id as an input parameter.

The Quick Start does the following in the Detail page.

• Adds a form with fields bound to a variable.
• Adds a Back button. An ojAction event is added to trigger an action chain.
• Creates an action chain that navigates back to the previous page triggered by an event on the Back button.
• Creates a page variable to store the object id as an input parameter.
• Creates a page variable to store the response from the endpoint. The Quick Start also creates a new Type that defines the structure of the variable.
• Creates an action chain that calls an endpoint when the page is loading and assigns the response to a page variable. The action chain has an input parameter mapped to a page variable.
Use the Add Detail Page Quick Start

When you have a page with a table or list component, you can use the Add Detail Page Quick Start to create a Detail page for a record and add a Details button to open the new page. The button is enabled when you select an object in the table or list.

You can open the Add Details Page Quick Start from pages that use a table or list component to display a collection. The Quick Start is listed in the Quick Start tab in the Property Inspector when you select the component on the canvas. As you step through the Quick Start, you select the endpoint for the business object and the business object’s fields that you want to display in the details page.

To create a Detail page:

1. Select the table or list on the canvas.
2. Open the Quick Start tab in the Property Inspector, if not already open.
3. Click Add Detail Page.
4. Select the data source containing the Get One endpoint to get the data you want to display in the Detail page. Click Next.
5. Drag the fields you want to include in the Detail page from the Endpoint Structure into the columns or template fields.
6. Specify the name of the button that will open the Detail page, and the title and name of the new page. Click Finish.

The page now has a new button that will navigate to a page that shows details of the selected object.

Work with the Functions Editor

You use the Functions editor to add custom JavaScript functions that can be called from within your page, flow or application.

Any JavaScript code that you add will have a defined scope based on the editor where you write the code. If your code will only be called from within a specific page, for example, to load some data when the page loads, you can write your code in the Functions editor for the page. If you want to use some JavaScript functions that will be used on multiple pages, for example, loading libraries for customizing navigation elements or custom web components, then you will probably want to open the Functions editor for a flow or the application. You use the Call Module Function action in an action chain to call functions. For details, see Add a Call Module Function Action.

The Functions editor for pages, flows and application opens the JavaScript file of an artifact. Each app, flow and page has a separate JavaScript file. For example, the main flow uses the files main-flow.js and main-flow.json to describe the artifact. You can open the Functions editor by opening the artifact in the Designer and clicking the Functions icon, or by locating the JavaScript file in the Source view in the Navigator.
The editor provides code validation and identifies the lines that contain syntax warnings and errors in the right margin of the editor. A lightbulb in the left margin indicates a hint for correcting invalid JavaScript code.

**Note:**

The auto-save function will not save a JavaScript file that has invalid code.

---

**Secure the Application**

You can secure access to your application with user credentials, and also create user roles to secure data at the level of the business object.

**Topics**

- About Authentication Roles and User Roles
- Allow Anonymous Access
- Manage User Roles and Access
- Embed a Web Application
- Configure Basic Authentication for a Mobile Application

**About Authentication Roles and User Roles**

You use authentication to manage access to the pages and data in your application. In addition to the default authentication roles, you can fine tune access to your application resources by creating user roles and assigning authenticated end users to them.

All app users are automatically assigned either the Anonymous User or Authenticated User authentication role, or both. If access to the app requires authentication, all users are automatically granted the role Authenticated User when they sign in. If anonymous access to the app is also allowed, users that sign in are granted the Authenticated User role AND the Anonymous User role, and users who are not signed in are only granted the Anonymous User role. You can use these roles when granting
permissions to operations on business objects when role-based security is enabled. The following table describes the two authentication roles.

<table>
<thead>
<tr>
<th>Authentication Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous User</td>
<td>All users who access Oracle Visual Builder applications are assigned this role when anonymous access to the application is enabled.</td>
</tr>
<tr>
<td>Authenticated User</td>
<td>All users who access Oracle Visual Builder applications are assigned this role after they sign in. An authenticated user can see all components and manage business objects unless access to the object is explicitly disabled for the Authenticated User role. All developers are assigned this role by default.</td>
</tr>
</tbody>
</table>

You use user roles to secure access to business objects and data in your application. The application's user roles ensure that users assigned the same role or group in the Oracle PaaS identity provider are granted equal access in your application. You define the user roles for the visual application in the User Roles tab of the visual application's Settings editor. The user roles defined for your application are stored in `user-roles.json` in the visual application's settings folder. See Manage User Roles and Access.

You assign users or groups in the identity domain to a user role in your visual application, but only identity domain administrators can add users to the identity domain, and it is the responsibility of the identity domain administrator to add users to groups and maintain them in the identity provider. Administrators manage groups using Oracle Identity Cloud Service (IDCS), or use Oracle Shared Identity Manager (SIM) to manage roles for services using a Traditional Cloud Account. All user authentication is delegated to the identity provider.

When a user attempts to access data in a business object secured by a user role, the roles assigned to the user are authenticated in the identity provider. The user is granted access if one of the user roles securing the business object is mapped to one of the roles or groups the user has been assigned to in the identity provider. Security based on roles is disabled by default. You can set role-based security and privileges for viewing, creating, updating and deleting objects in the Security tab of the business object editor. See Secure Business Objects.

Note:

By default, Authenticated Users can access all objects and components in your application. To thoroughly enable role-based security you must explicitly specify authentication or visibility for an object to a user role and disable access for the Authenticated User authentication role.

About Anonymous Access

You can enable anonymous access to allow users to access your app without signing in. If anonymous access is not enabled, all users must sign in with the credentials of their Oracle Cloud Account.
All users accessing your app are assigned roles that can be used to secure access to the data and services in the application. By default, your application requires authentication, but you can use the Settings editor of the app artifact to allow anonymous access to the app. When anonymous access is enabled, users that do not sign in are assigned the **Anonymous User** authentication role. By default, users assigned this role cannot access the data stored in your visual application’s business objects and data retrieved from services. You must explicitly allow anonymous users access to the data by configuring the security settings of business objects and services. See [Access and Secure Business Objects](../topics/secure-business-objects) and [About Service Connection Authentication](../topics/service-connection-authentication).

Changes that you make to authentication and security settings are only applied when you stage or publish the application. The versions of your application that are currently staged or published are unaffected. For example, if your application is already published, you must create a new version of the application before you can change the security settings. You then must stage and publish the application when you want the new security settings to take effect.

Note:
The service administrator must enable anonymous access in the instance’s Tenant settings. You will not be able to enable anonymous access for any of your visual applications if anonymous access to applications is not permitted for the instance.

The following table describes the options for allowing anonymous access to your application and artifacts.

<table>
<thead>
<tr>
<th>Allow Anonymous Access Options</th>
<th>Description</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web and mobile applications</td>
<td>You enable the Allow Anonymous Access option for a web or mobile app in the Security tab of the Settings editor for the application artifact. Settings for anonymous access must also be set explicitly for business objects and service connections. This option must be enabled to allow anonymous access to services and business objects.</td>
<td>When enabled, users are not required to sign in. Users that are not signed in are assigned the role <strong>Anonymous User</strong>. Signed in users are assigned the <strong>Anonymous User</strong> AND <strong>Authenticated User</strong>.</td>
</tr>
<tr>
<td>Business objects</td>
<td>You enable anonymous access to business objects by enabling role-based security in the business object’s Security tab and specifying the operations that the <strong>Anonymous User</strong> authentication role can perform. See <a href="../topics/secure-business-objects">Secure Business Objects</a>.</td>
<td>When enabled, anonymous users can perform operations on custom business objects based on the permissions granted to the <strong>Anonymous User</strong> authentication role.</td>
</tr>
</tbody>
</table>
### Allow Anonymous Access Options

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service connections</td>
<td>You enable anonymous access to service connection data by enabling and specifying the authentication mechanism for anonymous access in the service’s Authentication tab. See About Service Connection Authentication.</td>
<td>When enabled, anonymous users can access data from the service connections that are configured to allow anonymous access.</td>
</tr>
</tbody>
</table>

### Allow Anonymous Access

To allow anonymous access to an app and its data, you edit the security settings of each app artifact, business object and service connection. If your application is already published, you must create a new version and then change the security settings and publish the application.

Enabling anonymous access in the app artifact’s Security tab is required to enable users to access the app without signing in. When anonymous access is enabled for an app, the following property is defined in `app-flow.json`:

```json
"security":{
    "access":{
        "requiresAuthentication": false
    }
}
```

To allow anonymous access to an app and data in its business objects and from services:

1. Open your web or mobile application in the Navigator.
2. Open the application artifact and select the Settings editor in the designer.
3. Open the Security tab and select **Allow anonymous access** in the Access pane.
4. Open the **Security** tab of the business object.

5. Enable Role-based Security (if not enabled).

6. Configure the rights granted to users assigned the Anonymous User authentication role.

7. Open the **Authentication** tab of the service connection.

8. Select **Allow anonymous access**.
If the Allow anonymous access option is not visible, you might need to select **Enable authentication / proxy** and also confirm that anonymous access is an option for the authentication mechanism used to connect to the service.

9. Select the Anonymous Authentication Mechanism from the dropdown list.

If you want to allow external clients anonymous access to the Describe endpoint for business objects in your visual application, select **Allow anonymous access to business objects Describe endpoint** in the Business Objects tab of the visual application's Settings editor. If you choose to allow anonymous access, access to an endpoint will still require adding the header "Authorization: Public" to the request. This header is injected automatically for requests sent from your visual applications.

Manage User Roles and Access

You can create, edit and remove the user roles used to secure access to business objects in your application.

In addition to the Authenticated User role, when a user signs in to your application, they can be assigned an application user role based on their user credentials and the roles or groups they have been assigned to in the identity provider. You use the User Roles tab in the visual application's Settings editor to create user roles and assign roles and groups in the Oracle PaaS identity provider (IDCS) to the user roles. When you create a user role, the role and any groups or users assigned to it are automatically added to the application in IDCS.

After creating a role, you can secure access to business objects by specifying the application user roles that can access the object and setting the access privileges for the role in the business object's Security tab.

To create a user role in your visual application:

1. On the Oracle Visual Builder home page, locate the application where you want to change the settings and choose Settings in the application's Options menu.
Alternatively, open your web or mobile app and choose **Settings** in the application's Options menu in the toolbar.

2. Open the **User Roles** tab in the Settings editor.

   The User Roles tab displays a tile for each user role in your application, and the groups and users that have been assigned to it.

   ![User Roles Tab](image)

3. Click **Create Role** in the User Roles tab, or **Create Role** if you haven't created any yet.

4. Type the name in the Create Role dialog box. Click **Create**.

   This role name is displayed when designing your application. It is not exposed to end users.

5. Click **Assign groups or users** in the tile if no users or group have been assigned.

   If you want to edit a user role and some groups or users have already been assigned to it, click the Edit icon in the tile.

6. In the Change Assignments dialog box, click the Add icon for each group or user that you want to assign to the role. Click **Save Changes**.

   ![Change Assignments Dialog](image)

You can assign multiple groups and roles to your user role. The list of groups and users is defined in the identity provider and managed by the identity domain.
administrator. When you save your changes, the user roles for your application in IDCS is automatically updated.

Embed a Web Application

Your web application can be embedded in sites in domains associated with your Identity Domain as well as external sites.

You must explicitly allow embedding in your web application’s settings if you want to allow other applications to embed your application. For example, if you know that another site wants to use pages and data from your web app in their site, and they don’t want to or can’t link to your app, you can allow your app to be embedded in their app.

For security reasons, all embedding is denied by default. You can use the Settings editor in the application designer to edit the metadata of the application artifact. The web application’s security settings are stored in the configuration.json file, which is located in the application’s settings folder when you browse the application’s sources.

To allow your web app to be embedded in another app:

1. Open the web application in the Navigator.
2. Select the application artifact to open it in the application designer.
3. Click Settings in the application designer and open the Security tab in the Settings editor.
4. Enable Allow embedding in any application domain.

When your app is embedded within another app, the preferred method is for the other app to only embed the content of the page and not display the elements that wrap the content. For example, you might want to prevent a user from opening your app’s user menu and logging out when it is embedded in another app. You can edit the shell template page to remove content such as the header and footer elements that you don’t want to appear when the page is embedded.

Configure Basic Authentication for a Mobile Application

Mobile applications that you develop with Oracle Visual Builder can use basic authentication rather than the default Oracle Cloud authentication mechanism.
To use basic authentication, you specify a login endpoint URL and logout endpoint URL in the input fields that appear when you select Basic as the authentication mechanism in the Security tab of the mobile application's settings. At runtime, the mobile application presents a login screen where the user enters their user name and password (user credentials). These user credentials are converted into a basic authentication header and the login URL is invoked to pass the basic authentication header to the authentication service. On successful authentication, the user is navigated to the home page of the mobile application. If you want to pass additional HTTP headers to the service that you are connecting to through basic authentication, click New Header to display input fields where you enter the HTTP header(s) you want to pass.

The login and logout endpoint URLs can be any third party or Oracle Cloud service URL that supports basic authentication. This contrasts to the Oracle Cloud authentication mechanism which requires the user to have a valid account in the Oracle Identity Cloud Service that is associated with Oracle Visual Builder.

There are a number of things you need to know if you configure basic authentication for your mobile application. User roles, which secure access to individual page flows or business objects in your application based on the user's role, cannot be used if you configure basic authentication for your mobile application. The built-in variable, user, which accesses information about the current user when the mobile app uses the Oracle Cloud authentication mechanism, does not return information when the mobile app uses basic authentication.

Propagate Current User Identity and Direct (Bypass Authentication Proxy) are the two authentication mechanisms for REST service connections that you can use if your mobile application uses basic authentication. We recommend that you use Propagate Current User Identity to pass the user credentials entered by the mobile application user to the REST service. Note though that you cannot use Propagate Current User Identity while you develop and test the mobile application in Oracle Visual Builder using the Page Designer’s Live and Design modes or run it in a separate browser tab. For these scenarios, temporarily use one of the following authentication mechanisms for the REST service connection:

- Basic
- Client Credentials OAuth 2.0
- Resource Owner OAuth 2.0

Once you complete testing the mobile application in Oracle Visual Builder and are ready to install it on a device, switch the authentication mechanism that the REST service connection(s) in the mobile application use back to Propagate Current User Identity.

REST services that permit anonymous access can be accessed from a mobile application that uses Basic as its authentication mechanism if the REST service connection in the mobile application is configured to use Direct (Bypass Authentication Proxy) as the authentication mechanism.
Basic authentication is not a supported option for mobile apps that enable PWA support.

**Configure Data Cache and Offline Support**

Use the Oracle Offline Persistence Toolkit to enable data caching and offline support within your application.

The toolkit is a client-side JavaScript library that is delivered ready-to-use by Oracle Visual Builder. The toolkit provides caching and offline support at the HTTP request layer. This support is transparent to the user and is done through the Fetch API and an XHR adapter. HTTP requests made while the client or client device is offline are captured for replay when connection to the server is restored. Additional capabilities include a persistent storage layer, synchronization manager, binary data support and various configuration APIs for customizing the default behavior. This toolkit can be used in both ServiceWorker and non-ServiceWorker contexts within web and hybrid mobile apps.

Using the toolkit, you can configure your application to:

- Download content for offline reading where connectivity isn’t available.

  For example, an application could include product inventory data that a salesperson could download and read at customer sites where connectivity isn’t available.

- Cache content for improved performance.

- Perform transactions on the downloaded content where connectivity isn’t available and upload the transactions when connectivity returns.

  For example, the salesperson could visit a site with no Internet access and enter an order for some number of product items. When connectivity returns, the application can automatically send the transaction to the server.

- Provide conflict resolution when the offline data can’t merge with the server.

  If the salesperson’s request exceeds the amount of available inventory, the application can configure a message asking the salesperson to cancel the order or place the item on back order.

The architecture diagram illustrates the major components of the toolkit and how an application interacts with it.
To use the toolkit in a web or mobile application, you update the application’s `app-flow.js` file to include an `OfflineHandler()` function that determines the scope of data in your application to cache, what type of caching strategy from the toolkit to use, and so on. The following commented `app-flow.js` file demonstrates one scenario of how you might go about implementing caching and offline capabilities in your application. The following `app-flow.js` file also demonstrates how you enable the logging functionality of the toolkit while you develop the application that uses the toolkit. Enabling this type of logging during the development phase will help you understand what data the toolkit caches offline in your application. Disable the logging functionality when you are ready to publish your application in a production environment.

Responses from REST services to your application must not include either the `no-cache` or `no-store` value in the `Cache-Control` HTTP header as these values prevent the toolkit from working in your application. Work with the administrators of the REST services that your application connects to so that values in the `Cache-Control` HTTP header are configured appropriately.

Oracle maintains the toolkit as an open-source project. For additional information about using the toolkit, see the `README.md` and Wiki for the persistence toolkit on Github at https://github.com/oracle/offline-persistence-toolkit. API documentation for the toolkit is linked to from the aforementioned Github page, but can also be accessed directly at https://oracle.github.io/offline-persistence-toolkit/index.html.

```javascript
define(
  'vbsw/helpers/serviceWorkerHelpers',
  /**
   * Add the following entries to include the toolkit classes that you'll use. More information about these
   */
)
* classes can be found in the toolkit's API doc. See the link to the API doc in the paragraph before this sample file.

/*

'vest/persistenceManager',
'vest/defaultResponseProxy',
'vest/fetchStrategies',
/**
  * Add the following entry to enable console logging while you develop your app with the toolkit.
  */
'vest/impl/logger'

function (ServiceWorkerHelpers, PersistenceManager, DefaultResponseProxy, FetchStrategies, Logger) {

'use strict';

function AppModule() { }

function OfflineHandler() {

/**
  * Enable console logging of the toolkit for development testing
  */
Logger.option('level', Logger.LEVEL_LOG);
Logger.option('writer', console);

var options = {

/**
  * The following code snippets implements the toolkit's CacheFirstStrategy. This strategy
  * checks the application's cache for the requested data before it makes a request to cache
  * data. The code snippet also disables the background fetch of data.
  */

  fetchStrategy: FetchStrategies.getCacheFirstStrategy({
    backgroundFetch: 'disabled'
  }),
};

this._responseProxy = DefaultResponseProxy.getResponseProxy(options);
}

OfflineHandler.prototype.handleRequest = function (request, scope) {

/**
  * (Optional). Write output from the OfflineHandler to your browser's console. Useful to help
  * you understand the code that follows.
  */
console.log('OfflineHandler.handleRequest() url = ' + request.url + ' cache = ' + request.cache +

Chapter 4
Configure Data Cache and Offline Support
' mode = ' + request.mode);

/**
 * Cache requests where the URL matches the scope for which you want data cached.
 */
if (request.url.match('http://localhost:1988/webApps/ifaxitfaster/api')) {
    return this._responseProxy.processRequest(request);
} else {
    return PersistenceManager.browserFetch(request);
};

OfflineHandler.prototype.beforeSyncRequestListener = function(event) {
    return Promise.resolve();
};
OfflineHandler.prototype.afterSyncRequestListener = function(event) {
    return Promise.resolve();
};
AppModule.prototype.createOfflineHandler = function () {
    /** Create the OfflineHandler that makes the toolkit cache data URLs */
    return Promise.resolve(new OfflineHandler());
};
AppModule.prototype.isOnline = function () {
    return ServiceWorkerHelpers.isOnline();
};
AppModule.prototype.forceOffline = function (flag) {
    /** if online, perform a data sync */
    if (!flag) {
        return ServiceWorkerHelpers.syncOfflineData();
    }
    return Promise.resolve();

}).catch(function (error) {
        console.error(error);
});

); return AppModule;
});

Work with Application Resources

You can import and export the resource files that are used in pages in your application.

Topics:
• About Application Resources
About Application Resources

By default, each web and mobile application contains a resources folder for the components, images, translation files and other resources that can be used in the application. A flow artifact can also contain a resources folder for images and translation bundles used by pages in the flow. A page artifact can contain a resources folder for translation bundles.

To add resources, you can use the import tool in the Navigator and Source View to add folders and files to your application, for example, to add files and resources from other applications. Resources are typically files that you import to support or add functionality to pages in your application. For example, if you want to use an image in a page, you can import the image as a resource into an images folder and then use an image component in a page that references the image location.

Each of your web and mobile applications has one or more resources folder containing the application's resources. By default, the application artifact contains a resources folder containing folders for the following types of resources.

<table>
<thead>
<tr>
<th>Folder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>components</td>
<td>This folder contains your application's Composite Components. Composite Components that you import in the Page Designer are saved to this folder. You can also click Create Composite Component in the Navigator to create a custom Composite Component.</td>
</tr>
<tr>
<td>css</td>
<td>This folder contains the app.css stylesheet that is linked from pages in your application. The stylesheet is empty by default. You can edit the stylesheet to add custom styling to page elements. app.css is not used to control the styling of Oracle JET components.</td>
</tr>
<tr>
<td>images</td>
<td>This folder is the default location for any custom images that you might want to add to your application. This folder is empty by default. Images in the application's resources directory can be used in any page in the application. See Work with the Image Gallery. Flow artifacts can also contain an images folder that stores images that can be used in pages in the flow. When you add an image to a page, it is stored in the images folder of the flow that contains the page.</td>
</tr>
</tbody>
</table>
Import Resources

By default your application contains folders for components, stylesheets, images and translation files. To add resources, you use the Import command to add files to your application, for example, to add files and resources from other applications.

You can import resources by uploading individual files and ZIP archives using the Import command available when you right-click an artifact or a resources folder in the Navigator, or by dragging the files directly from your local file system onto a folder in the Navigator. You can create new sub-folders by modifying the path in the Import Resources dialog box.

You can import resources into the application's `resources` folder and sub-folders, or any other artifact in the Navigator. The location you select will determine the scope of the resources you import. When you import the resource, you should confirm the Import location. For example, you can import an image into the `flows` folder, but the expected location for an image is in an `images` folder in one of your application's `resources` folders. Images that are not in an `images` folder will not appear in the Image Gallery.

To import resources into a new application:

1. Open your web or mobile application in the Navigator and locate the folder or artifact where you want to import the resource.

2. Right-click the folder or artifact and choose `Import` in the popup menu.
Alternatively, drag a file from your local file system onto the folder or artifact in the Navigator to open the Import Resources dialog box.

3. In the Import Resources dialog box, choose the file or archive with the resources you want to import. You can drag the resource into the drop target area or click the drop target area to navigate to the resource on your local system.

Optionally, you can edit the path in the Import location field to create new folders.
4. Click Import.

Export Resources

You might want to export an application's resources when you want to import them into another application or share them with a team member.

You can download an application's resources by exporting the application as an archive.

To export an application's resources:

1. Open your web or mobile application and choose Export in the application's Options menu in the toolbar.
Alternatively, on the Oracle Visual Builder home page, locate the application and choose Export in the application's Options menu.

2. Click one of the export options to download the archive.

You can choose if you want the application archive to include the data stored in the application's custom business objects.

The application archive includes the application's resources.
Work with the Image Gallery

You use the Image Gallery to import image resources into your application and when selecting the image resource referenced by an image component. You open the Image Gallery from the Data tab in the Property Inspector when an image component is selected on the canvas.

Images in your application are stored in an images folder located in one of the resources folders in your application. A folder for resources in your application is created by default when the application is created. Images in the application's resources folder can be used in any page of your application. In addition to the application's default resources folder, each flow in the application might have a resources folder for resources used in pages in the flow.

You can use the Image Gallery to view and manage the images in your application. The Image Gallery only displays the images that are stored in the images folders of the application and the current flow. Images stored in other locations are not visible in the Image Gallery.

You can use the Image Gallery to perform the following tasks:

- Import images. You can choose to add images as resources of the current flow or the application.
- Select an image displayed by an image component. You can select images stored in the application's or the current flow's images folder. When you select the image, the path to the image (for example, {{ $flow.path + 'resources/images/myimage.png' }}) is entered in the component's Source URL field in the Property Inspector.

To import images into the Image Gallery:

1. Open a page in the Page Designer and select an image component on the canvas.
   
   You can select any image component on the canvas. Alternatively, you can temporarily drag an image component onto the canvas.

2. Open the Data tab in the Property Inspector and click the Image Gallery icon.
   
   When an Image component is selected on the canvas, the Data tab in the Property Inspector displays a Source URL field for the path to the stored image. The field is empty when no image has been defined.
You need to open the Image Gallery to add images to the application’s images folder. If you drag an image into the drop target area in the Data tab, the image is added to the flow in the Image Gallery and the path to the image is automatically entered in the Source URL field, but the Image Gallery dialog box does not open.


Alternatively, you can select Flow in the Image Gallery to import images into the flow’s images folder. The folder will be created automatically if it does not exist.
The Images panel of the Image Gallery displays a list of the images that can be used in pages in the application and in the current flow. You can select an image in the Images panel to preview the image. If you select an image in the Images panel and click Select, the path to the image will be entered in the Source URL for the image component in the Data tab.

4. Drag your image into the drop target area in the Image Gallery.

Alternatively, you can click the drop target area to select an image using your local system’s file browser. You can import as many images as you want, but you need to add them individually. You can choose if you want to import an image into the Application or Flow resource folders by selecting the folder in the Images panel.

5. Click Close to close the Image Gallery without selecting an image for the image component.

Work with OJET Web Components

Oracle JavaScript Extension Toolkit (JET) Composite Component Architecture are supported in the design-time and run-time of visual applications.

Topics:
- About OJET Web Components
- Work with the Component Exchange
- Import a Web Component Archive
- Create a Custom Component

About OJET Web Components

You can import web components as archives or directly from the Component Exchange associated with your instance.
Web components based on the Oracle JavaScript Extension Toolkit (JET) Composite Component Architecture are supported in the design-time and run-time of visual applications. See Web Components in the JET Cookbook for an example of a web component.

You can expand the resources node in the Artifact Browser to view the components that have been imported into your web or mobile app.

A web component must contain the following files. It may also contain additional files and folders, for example, a SCSS file or resources such as translation files.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loader.js</td>
<td>This is a RequireJS module that defines the web component dependencies for its metadata, View, ViewModel, and CSS. The naming convention for web components requires that the name of the file is loader.js.</td>
</tr>
<tr>
<td>component.json</td>
<td>This is a web component metadata file that defines its available properties and methods. The naming convention for web components requires that the name of the file is component.js.</td>
</tr>
<tr>
<td>view.html</td>
<td>This file is the view for the web component.</td>
</tr>
<tr>
<td>viewModel.js</td>
<td>This describes the ViewModel for the web component where methods defined in the web component metadata are defined.</td>
</tr>
<tr>
<td>styles.css</td>
<td>This contains the custom styling for this web component.</td>
</tr>
</tbody>
</table>
Note:

Oracle JET web components are currently not backward-compatible. When importing web components, and when upgrading web components or your app, you need to ensure that your application and all web components that you use in your application are using the same version of JET. The JET version for the current release is listed under Supported Oracle JET Version in What's New for Oracle Integration.

Work with the Component Exchange

The Component Exchange is a repository of components that can be installed in your Visual Builder instance.

Your Visual Builder instance administrator specifies the Component Exchange that is associated with the instance. All developers in the instance are able to the same set of components. The exchange contains a set of default components that have been provided by Oracle, and any components that other developers have published to your exchange.

You use the Components tab in the Navigator to install and manage the components that you download from the Component Exchange. The Components tab has the following tabs for locating and managing components from the exchange: Browse, Installed and Updates.

You use the Browse tab to search the Component Exchange for components that can be installed, to open a component's details page and to install components.

You use the Installed tab to view a list of installed components. The tab displays details about each component, and components are badged to indicate warnings or available updates.

You use the Updates tab to view a list of available updates and to update components to the latest version.
Get Components From the Component Exchange

When a Components Exchange is associated with your instance, you can use the Components tab in the Navigator to add and manage components from the exchange.

Your administrator needs to associate your instance with the Component Exchange before you can get components from the Component Exchange.

To add a component from the Component Exchange:

1. Open the Components tab in the Navigator.
   Alternatively, in the Components palette in the Designer you can click Get Components in the Options menu, or click the Search Exchange button that is displayed in the palette when you use the palette’s filter field.

2. Locate the component you want to install and click Install.
   You can use the filter field in the Components tab to filter the list of available components. You can click the component in the Components tab to open a tab containing details about the component, including a description and examples of how to use the component.
After you install the component, the new component is added to your Components palette. You can now drag the new component onto the canvas and use it in your pages.

Update a Component from the Component Exchange

When a newer version of an installed component is available, you can install the newer version in the Updates tab of the Components pane in the Navigator.

A notification is displayed in your browser window and the Components icon in the Navigator is badged to indicate a new version of a component is available.

To update a component from the Component Exchange:

1. Open the Components tab in the Navigator.
   When an update for one of your installed components is available, the Updates tab in the Components tab is badged.

2. Open the Updates tab in the Components tab.
   If you installed a component that is part of a pack, the Updates tab displays the name of the pack containing the newer version of your component.
3. Click **Update All** to install all updates available for installed components.

To update an individual component, you can click the component's name to open its detail page and use the update button in the page. The details page might also display a notification when the installed component is not compatible with the JET version of your Visual Builder instance.

---

**Uninstall a Component**

When you no longer want to use an installed component in your application, you can uninstall it to remove it from your Components palette.

To uninstall a component:

1. Open the Components tab in the Navigator and locate the component you want to uninstall.
   
   If you know the name or details about the component you can use the Search field to filter the list of components.

2. Click the component to open the component's details page.

3. Click **Uninstall Component** in the details page.
Import a Web Component Archive

When you have a web component archive, you can use it in your application by importing it in the Components palette.

To import a custom web component archive:

1. Open the application into which you want to import the component.
2. Open a page in your application in the Page Designer.
3. Locate the Custom section in the Components palette and click Import Custom Component ( ).

4. Upload your ZIP archive in the Import CCA Component dialog box. Click Import. You can add the ZIP archive by dragging it into the upload area or clicking the area and locating the file on your local system.
The imported component is displayed in the list of components in the Custom category of the Components palette or in the category specified by the component's metadata. After importing the custom component you can position it in your page and configure its properties in the Property Inspector as you would a standard component.

Create a Custom Component

You can create custom web components from a template in the resources folder of your application. The new component is automatically registered as a new component in the Custom category of the Components palette.

Custom web components are based on the Oracle JavaScript Extension Toolkit (JET) Composite Component Architecture. The template for creating the web component is included with Oracle Visual Builder. See the Oracle JET Cookbook for examples of a web component.

When you use the Create Composite Component dialog box to create the component, the new component contains the JavaScript, HTML, stylesheet and JSON files used in
composite components. The files contain sample code to help you get started. After
the component is created, you can click the component artifact in the Navigator to
open it in an editor. The editor contains editors for each of the component's files.

To create a custom web component:

1. Expand the resources folder of your web or mobile application in the Navigator.
2. Click Create Composite Component ( ) next to the components folder.
3. Type the Id for the component in the Create Composite Component dialog box.
   Click Create.

The new component is displayed under the components node in the Navigator.

Work with Translations

You can use the Property Inspector to bind text strings to values that are stored as
keys and values in JSON files. You can open and edit the translation JSON files in the
code editor, and download and upload the files as application resources.

Topics:

• About Translation Resources
• Understand the Structure of Translation Bundles
• Understand Translation Keys for Display Texts
• Generate Translation Keys for Display Texts
• Use Translation Strings in JavaScript Files
• Download Bundles for Translation
• Upload Translated Files
• Create Translation Bundles
About Translation Resources

The keys and values for translating your application are stored as JSON files in translation bundles in your application.

The names and locations of your translation bundles are up to you, but you must understand the rules governing the file names and the structure of your bundle to ensure that the bundles are recognized when you run your application.

By default, web and mobile apps will contain a default app-level bundle with the name app located in the resources/strings folder of the app. The relative path to the default translations bundle is stored in app-flow.json. The path identifies the top-level JSON file in the bundle that identifies the translation locales included in the bundle. If you open the application artifact in the code editor you can see the entry for the path to the translation bundle.

"translations": {
    "app": {
        "path": "/resources/strings/app/nls/app-strings"
    }
},

The suggested location for a bundle is resource/strings/<bundle-name>/nls/<bundle-name>-strings.json, where <bundle-name> is the name for the bundle you create. Bundles can be located where you choose, however the paths to the bundles must be specified in app-flow.json. For example, if you created a resource folder containing a bundle for the flow MyFlow, the path in app-flow.json might be similar to ./MyWebApp/flows/MyFlow/resources/strings/MyFlow/nls/MyFlow. You would use the period (.) at the beginning of the path to make it relative to the file.

When using translations in code, you can replace code that produces or uses an untranslated string with code that uses the translations object to retrieve the translated string from the bundle. For more on the translations object, see Translations.

Understand the Structure of Translation Bundles

You will want to understand the structure of translation bundles if you want to create additional translation bundles in your application.

The following guidelines describe the structure for the application’s default translation bundle:

• The bundle contains a root folder named nls.
• The nls folder should contain a JSON file identifying the translation locales contained in the bundle.
• All bundle files should be JSON files with -strings.json appended to the bundle name. (For example, the JSON files for the MyNewPage bundle will be named MyNewPage-strings.json.)

The path to the root JSON file in each bundle in your application must be specified in app-flow.json.
Your bundle contains a JSON file at the root level of the nls folder that identifies the supported languages, and one or more folders within the nls folder containing JSON files storing the keys and values of translation strings. By default, the nls folder contains a folder named root that contains the root translation file app-strings.json.

When no additional locales are specified, the root level app-strings.json file only contains "root": true. If one additional locale is supported, for example, a French locale and translations, the root level JSON file would contain the following locale identifiers:

```
{
  "root": true,
  "fr": true
}
```

For each locale, the nls folder should contain a corresponding folder containing a JSON translation file with the translation strings for that locale. The default name for the JSON file in the root folder is <bundle-name>-strings.json. If a French locale is added, the nls folder would also contain a folder named fr containing a JSON file <bundle-name>-strings.json. Each translation file contains key/value objects and
object metadata. For each object you can include descriptive metadata, including unique id identifiers for the element details and details about the object’s context that can be useful for translators.

Understand Translation Keys for Display Texts

To translate your application into other languages, you need to create translation bundles that store keys and values for the texts in your application’s UI.

When you save a string using the Translatable String popup in the Property Inspector, the value for the string is externalized to the translation bundle, and the value of the string is bound to the key in the translation file in that bundle. If you open the page in the Code view you can see the name of the translation bundle and the key bound to the string value. For example, an input label in the Code view code might be similar to

```html
<oj-input-text label-hint="Name" id="oj-input-text--452490439-1" value="{{ $page.variables.authors.name }}"></oj-input-text>
```

When you externalize the string, the value of the string is replaced with a string similar to the following that binds it to a key in the translation file:

```json
[[ $application.translations.app.input_text_label_hint_daea ]]
```

The expression identifies the scope and name of the translation bundle, and the translation key, using the following syntax: `<?scope>.translations.<bundle-name>.<key>.`.

You can also include parameters in the expression by using the syntax `<?scope>.translations.format('<?bundle>', '<?=key>', {params}).` For example, if you use the expression `[[ $application.translations.format('app', 'bind_text_value_372d', { region: $page.variables.Country.region }) ]]]` in the UI component, the key-value pair defined in the bundle might be "bind_text_value_372d": "(This country is in {region})". In this case, the string “This country is in “ would be translated, and region would be the value of the page variable.

When you open the translation file in the editor, the file contains a key and value pair similar to the following:

```json
"input_text_label_hint_daea": "Name",
   @input_text_label_hint_daea: {
      "description": ""
   }
```

By default, each key-value pair in the translation file contains a description field as part of the metadata for the pair that can be used to provide a description of the value, for example, to provide additional context for the string that can be useful when translating the string. You can modify the file in the editor to add description metadata and other metadata for each key-value pair.
Tip:

You can use the Code view to locate strings in a page that have not been externalized for translation and add the string to the translation bundle. You can use the Audits window to locate all the strings in the application that have not been externalized.

Generate Translation Keys for Display Texts

For static strings in the UI of your application, you can use the Property Inspector to add keys and values for the strings to a bundle for translation.

You can use the Property Inspector to generate and add keys for UI components to a translation bundle. When you use the Property Inspector to define the translatable string, a key is generated automatically, but you can specify your own key in the dialog box. You can also edit the keys, values and metadata in translation files directly using a text editor.

To generate a key for a string using the Property Inspector:

1. Select the component on the canvas that you want to be translatable.

   You can select components that use a static string in their text fields, for example, a Button component. You cannot create translations for elements where the display label or text is the result of an expression or variable.

2. Click the globe icon for the text field to open the Translatable String popup.
3. Confirm or edit the string to be translated. Click **Save**.

   The Translatable String popup contains a Text field for the string to be translated. The popup also contains a Description field that you can use to provide a description of the context for the string. The description text is included as metadata in the translation bundle.

**Download Bundles for Translation**

You can download translation files to your local system from the Translations tab in the Settings editor when you want to translate the application’s strings with your preferred translation tool or service.

When you download the translation bundles of your visual application, the archive that you download contains the translation files for all of the applications in your visual application. The archive contains a folder for each translation bundle in your application, and each folder contains a file in the `.arb` format with the strings to be translated. The file also contains metadata defining the bundle name and path that is used when uploading the file after the strings have been translated. The metadata in the file might be similar to the following:

```
"@@x-bundleName" : "app",
"@@x-bundlePath" : "webApps/mycontacts/resources/strings/app/nls/app-strings",
```
When downloading the bundle, you can choose to download an archive containing all strings that are stored in the translation bundle or an archive containing only those strings in the translation bundle that were added or updated since the last time you downloaded the translation bundle. The first time that you download the translation bundle you can only choose to download an archive containing all strings.

To download a translation resource bundle:

1. Open your web or mobile application and choose **Settings** in the application’s Options menu in the toolbar.

   [Image of a settings menu]

   Alternatively, on the Oracle Visual Builder home page, locate the application and choose Settings in the application’s Options menu.

2. Open the **Translations** tab in the Settings editor.

   [Image of a settings editor with a translations tab]
3. Click the link for the archive that you want to download to your local system.

Use Translation Strings in JavaScript Files

You can use translation strings in JavaScript files and retrieve the translated text from the bundle when the function is called.

When you use a text string in a JavaScript function, the text string can be externalized to a translation bundle. You can replace the code that produces or uses the untranslated string with code that uses the `translations` object to retrieve the translated string from the bundle. When you edit the JavaScript function you will need to either hard code the translation key in the code or pass the key into the function as a parameter. To use the `translations` object in the function, you need to pass the object into the function as a function parameter.

For example, you might have a UI component in your page that displays a text string that comes from a JavaScript function that is called by a callModuleFunction action in an action chain. In the action chain, you can pass the `translations` object (for example, `$application.translations`) to the function and then assign the result to a variable bound to the UI component that displays the string.

To display a translated string used in a JavaScript function:

1. Create a key for the string that you want translated and add the key and string to the translation bundle.
   
   You can use the Make String Translatable button in the Property Inspector to create the key in the bundle and generate the expression for the key. Alternatively, you can edit the translation file in the editor to create the key in the bundle and enter the expression for the translation string in the UI component's Text field in the Property Inspector.

2. Copy the expression containing the bundle name and key. The expression is displayed in the Text field of the component after the string is externalized.
   
   The expression might look similar to 
   
   ```
   [[ $application.translations.app.h1__text_041a ]].
   ```
   
   In this example, `app` specifies the bundle name, and `h1__text_041a` is the key. The bundle and key are used to evaluate the translated string. `$application.translations` specifies that the application-scoped translations object is used. The translations object might also be `$flow` or `$page` scoped, depending on where the bundle is located.

3. Modify the JavaScript function so that the `translations` object `$application.translations` can be passed to the function from the action chain.
   
   In the following example, the action for calling the function will use `translations` to pass the object to the function.
   
   ```javascript
   PageModule.prototype.getMessageFromBundle = function(translations) {
      ...
   };
   ```

4. Edit the function to replace the untranslated text that should be displayed in the component with code that retrieves the translated text using the `translations` object. When the object is available in the function, the bundle name and key are used to retrieve the translated string from the bundle.
For example, the function can return a simple translated string:

```javascript
PageModule.prototype.getMessageFromBundle = function(translations) {
    ...
    return translations.app.h1__text_041a;
};
```

You can also include parameters to generate a formatted message:

```javascript
translations.format('app', 'h1__text_041a', param1, param2)
```

5. Create an action chain that calls the function (callModuleFunctionAction) and assigns the result (assignVariablesAction) to a variable (in this example, Value).

In this example you can see that the translations object is a parameter of the callModuleFunction action that is passed to the function, and that assignVariables assigns the result of callModuleFunction to the page variable Value.

```json
"root": "callModuleFunction1",
"actions": {
    "callModuleFunction1": {
        "module": "vb/action/builtin/callModuleFunctionAction",
        "parameters": {
            "module": "{{$page.functions}}",
            "functionName": "getMessageFromBundle",
            "params": [
                "{{ $application.translations }}"
            ]
        },
        "outcomes": {
            "success": "assignVariables1"
        }
    },
    "assignVariables1": {
        "module": "vb/action/builtin/assignVariablesAction",
        "parameters": {
            "$page.variables.Value": {
                "source": "{{ $chain.results.callModuleFunction1 }}"
            }
        }
    }
}
```

6. Edit the Text field of the UI component to replace the generated expression with the page variable storing the result of the method (Value).
Upload Translated Files

You can upload the resource files containing translated strings in the Translations tab in the Settings editor.

To upload translated file you need to create a ZIP archive containing the ARB files with the translated strings. Before you create the archive, however, you need to modify the file names of the ARB files to append the locale abbreviation to identify the correct locale. For example, to upload an ARB file that contains translations for the French locale, you will modify the file name to `<bundle-name>-strings-fr.arb`.

To upload an archive with translation files:

1. Open your web or mobile application and choose Settings in the application's Options menu in the toolbar.
Alternatively, on the Oracle Visual Builder home page, locate the application and choose Settings in the application's Options menu.

2. Open the **Translations** tab in the visual application's Settings editor.

3. Locate the archive containing the translated files on your local system and drag it into the upload area in the Translations tab. Click **Close**.

If you added a file with translations for a new locale, a folder for the new locale is created in the bundle. The new locale is also automatically added to the file defining the locales in the bundle.
Create Translation Bundles

You can use the Translation tab in the Settings editor to create additional translation bundles for application artifacts, flows and pages.

By default, each web or mobile app contains a translation bundle in the app’s top-level resources folder. You can create additional translation bundles for the app in the app artifact’s Settings editor. You can also create bundles for individual flows and pages in the Settings editor for the flow or page when you want the translations to be flow-scoped or page-scoped.

When you create a new bundle in the Translations tab of the Settings editor, the metadata for the new bundle is automatically added to the JSON file of the app, flow or page. For example, if you create a translation bundle for a page in the page’s Settings editor, a resources folder containing the new bundle is created at the page level and the page’s JSON file is updated with the metadata for the bundle.
When you externalize a text for translation, the string and key are added to the nearest available bundle by default. For example, if you create a translation bundle for a page, the strings and keys are added to the page's bundle when you externalize strings in the page. Strings in other pages are not added to that bundle when they are externalized.

To create a translation bundle:

1. Open the Translations tab in the Settings editor of an application, flow or page artifact.

   The Translation Bundles tab displays a list of the bundles scoped for the selected artifact and the path to the bundle.
2. Click +Bundle to open the New Translation Bundle dialog box.

3. Supply the bundle name in the dialog box. Click Create.

When you click Create, the new translation bundle containing the translation files and folders is created in the strings folder in the resources folder of the artifact. The strings and resources folders are created if they do not exist for the artifact. The path to the new bundle is displayed in the Translations tab. The path is relative to the artifact's JSON file.

Integrate Your Application With a Git Repository

You can specify a Git repository for each of your applications to collaborate with other team members and backup your sources.

Topics:

• About Git Integration
• Add Credentials for Your Developer Cloud Service Account
• Link Your Visual Application to a Git Repository
• Pull Files From Your Git Repository
• Push Your Changes to Your Git Repository

About Git Integration

You can integrate your visual applications with a Git repository hosted with an Oracle Developer Cloud Service project.

You can use the Git integration to store versions of the source files of each of your visual applications in a Git repository that can be shared with other developers. After creating a link to a Git repository, you can use the commands in the Git menu to pull sources from the repository and push sources to the repository. A log of merge conflicts when pulling sources is displayed in the Audits window, and you can open the files in the source editor to view and resolve the conflicts.
Add Credentials for Your Developer Cloud Service Account

To integrate your visual application with a Git repository, you must provide the URL and credentials for a Oracle Developer Cloud Service account. Those credentials are used when linking your visual application to a Git repository of a Oracle Developer Cloud Service project.

You will need to contact your service administrator if you do not know the credentials for your Oracle Developer Cloud Service account. Credentials are not stored when you export your application.

To add your Oracle Developer Cloud Service account credentials:

1. Open your visual application.

2. Click the Git icon ( ) in the toolbar and select Configure DevCS Credentials in the menu.

3. Click Add Credentials in the Configure DevCS Credentials dialog box.

4. Enter the URL, username and password for your Oracle Developer Cloud Service account. Click Save Credentials.
After your credentials are checked and saved, the new credentials are added to the list in the Configure DevCS Credentials dialog box.

5. Click Close in the Configure DevCS Credentials dialog box.

After you enter and save your credentials for Oracle Developer Cloud Service account you can create a link between a visual application and a specific repository.

**Link Your Visual Application to a Git Repository**

After you provide your credentials to your Oracle Developer Cloud Service Oracle Developer Cloud Service account, you can create a link between your visual application and a Git repository of your Oracle Developer Cloud Service project.

After you link your application to the branch of a Git repository, you can pull source files from and push source files to the branch, for example, to create a copy of an application by pulling the source files into a new visual application.

The Git repository and branch must exist in your Oracle Developer Cloud Service project before you can link it to your visual application. When selecting the repository branch that you want to use, you will see an up-to-date list of the branches that are available. If you do not see the branch you want to use, you should check that the branch exists and that you are using the correct credentials. You cannot use Oracle Visual Builder to create repositories or branches.

To link a visual application to a branch in a Git repository:

1. Open your visual application.
2. Click the Git icon and select **Link DevCS Git Repository** in the Git menu.

   The dialog box will display the location of a Git repository if one is already linked to your visual application.
3. Click **Add Link** to open the Link DevCS Git Repository dialog box.

![Link DevCS Git Repository dialog box](image)

4. In the DevCS URL with Credentials field, select the URL of the account that you want to use.

   The dropdown list displays the Oracle Developer Cloud Service accounts that you provided credentials for.

5. Select the project, the Git repository and the repository branch. Click **Save Configuration**.

   The dropdown list displays the projects, repositories and branches that are available to you in the instance. The branches and repositories in the dropdown list are determined by the project that you select.

6. Click **Close** in the Linked DevCS Git Status dialog box.

   You can now use the push and pull commands in the Git menu of your visual application.

**Pull Files From Your Git Repository**

You use the Pull from Git command in the Git menu to update your visual application with the source files from the linked Git repository.

To pull source files from a repository:

1. Open your visual application

2. Click the Git icon in the toolbar and select **Pull from Git** in the menu.

   The Update Application from Git dialog box displays the details of the branch containing the source files.
3. Click **Update From Git**.
   The dialog box displays a progress bar while pulling the source files from the branch. You

4. Click **Close** when the update is finished.
   The dialog box displays a status message when the update is complete.

---

**Push Your Changes to Your Git Repository**

You use the Push to Git command in the Git menu to upload the source files in your visual application to the linked Git repository.

To help avoid merge conflicts, you should update the source files in your visual application by pulling the most recent versions from the repository before you push any changes. If the file versions in the repository are newer than the versions in your visual application, you'll see a status message when attempting to push your changes that the push was rejected and you should pull the most recent versions from the repository before pushing.

To push the visual application source files to the repository:

1. Open your visual application
2. Click the Git icon in the toolbar and select **Push to Git** in the menu.
   The Push Content to Git Repo dialog box displays the details of the target branch.
3. Type a comment that describes the content you are pushing to the repository. Click **Push to Git**.

   The comment message that you provide in the dialog box is displayed when you examine the Git activity log for the branch in Oracle Developer Cloud Service.

4. Click **Close** when the push is finished.

   The dialog box displays a status message when the push is complete.
Work with Business Objects

A business object is a resource, such as an invoice or purchase order, similar to a database table; it has fields that hold the data for your application. Like a database table, a business object provides the structure for data. Business objects are stored in a database. The apps in your visual application and other clients access the business objects via their REST endpoints.

Topics:
- About Working with Business Objects
- Create and Edit Business Objects
- Access and Secure Business Objects
- Create Rules for Business Objects
- View and Edit Data in Business Objects
- Work with the Data Manager
- Work with the Business Object Diagrammer

About Working with Business Objects

The Business Objects pane in the Navigator lists all the business objects that are available for use in your application.

You use the Business Objects pane to browse your business objects, create new business objects and open pages where you can edit the business objects. The pane contains a button for creating business objects and an Options menu to open the Data Manager and business object Diagrammer. See Work with the Data Manager and Work with the Business Object Diagrammer.

After a business object is created, you can select the business object in the Business Objects pane to view and edit the business object’s details, including details about its
fields and data, related objects and security settings. Selecting a business object in the pane opens the business object’s page in the editor. The following table describes the tabs in a business object’s page that you use to edit business objects.

<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
</tr>
</thead>
</table>
| Overview  | The Overview tab displays the Id of the business object and contains fields for specifying the singular and plural forms for the label used to identify the business object. The Overview tab also contains the following items:  
  • **Relationships.** Displays the relationships between the business object and other business objects in your application. See View, Create, and Edit Business Object Relationships.  
  • **Contains Application Setup Data.** When enabled, the data in the business object is considered to be required for the application to function properly, for example, data used in a list of values (LOV) referenced by another business object. When enabled, the data in the business object will always be included when you export or publish the application. See Export a Visual Application. |
| Fields    | The Fields tab contains a table displaying the fields defined for the business object. The tab contains a **New Field** button for defining new fields. You can select a field in the table to edit its properties in the editor. See Edit Fields of Business Objects. |
| Security  | The Security tab is used to enable role-based security for the business object, and, when enabled, to specify the operations that can be performed by users based on the user role they are assigned. See Secure Business Objects. |
| Business Rules | The Business Rules tab contains a visual editor for creating custom business rules that can perform functions, such as field validation, that can be triggered by object events and actions. See About Adding Business Rules. |
| Endpoints | The Endpoints tab contains a list of the endpoints that are available for the business object. It also contains the resource APIs, the URLs that can be used to access the metadata and data of the business object. To allow other clients and applications access to the APIs using basic authentication, see Manage User Roles and Access and About Allowing Access to the Catalog API. See View the Endpoints of Business Objects |
Create and Edit Business Objects

You use custom business objects to store data that is not provided by an external service. As you develop your applications, you can create business objects and edit the business object fields to meet your needs.

Topics:
- Create a Business Object
- View, Create, and Edit Business Object Relationships
- Add Fields to Business Objects
- Edit Fields of Business Objects
- Add a Field for Aggregating Data
- Add a Formula to a Field

Create a Business Object

You use custom business objects to store data that is not provided by an external service. The data in your custom business objects is stored in the database that was provisioned for your service instance.

When you create custom business objects, you specify each of the fields that your application needs. As you develop your application, you can modify your business objects to add and modify fields as needed. Your custom business object will be exposed as a set of endpoints that provide REST APIs for operations that you can call from page components.

To create a business object:

1. Open the **Business Objects** tab.
   - The Business Objects pane opens. By default, the **Objects** tab is selected.
2. Click the + sign (**Business Object**).
3. In the New Business Object dialog box, enter the **Label** for the business object and click **Create**.
   - The Id value is filled in automatically based on the Label.

When you click Create, an artifact window for the new business object opens in the main window and displays the Overview tab. The window contains additional tabs for viewing and editing the various attributes of the business object: Fields, Security, Business Rules, Endpoints, and Data.

You can see the endpoints for the business object that are created by default in the Endpoints tab.

<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>The Data tab contains a table that displays the data stored in the fields of the business object. The tab contains tools for adding and editing the data. See View and Edit Data in Business Objects.</td>
</tr>
</tbody>
</table>
You can also create business objects using the Diagrammer.

To delete a business object, right-click the business object and select **Delete**.

**View, Create, and Edit Business Object Relationships**

The Relationships area of the Overview tab for a business object displays the relationships between that business object and other business objects in your application.

When you create a reference to another business object, the relationship between the current business object and the one being referenced is, by default, many-to-one. For example, when you create a reference from an Employee business object to a Department business object, an Employee can belong to only one Department, but a Department normally has many employees. When you first create such a relationship, it is first shown as a dotted line for the object that represents the one side. When you click the line, the details from the many business object are loaded and the relationship is shown.

- **Relationships**

  - Employee ➔ Department

  The referenced business object, Department, also appears in the Endpoints tab for the Employee business object.

  You can delete a relationship, but only if its associated business objects are not used anywhere in your application.

To create a new relationship:

1. In the Business Objects tab, select the business object that you want to view. The contents of the Overview tab are displayed.
2. Click the **New Relationship** icon next to Relationships.
3. In the dialog box, select the business object you want to create a relationship with.
4. Select the cardinality for each object (Many or One). You can click the **Reverse Relationship** button to switch between one-to-many and many-to-one. You can specify cardinalities of one-to-one and many-to-many between business objects, in addition to many-to-one and one-to-many. If you specify many-to-many, an intersection business object with two reference fields is automatically created. Its name is a concatenation of the two business objects. For an intersection business object, you can't deselect the Required check box in either of the reference fields, and the default delete rule is Cascade.
5. Edit the properties of the relationship field (the Field Name, the Display Label, the Required check box, the Delete Rule, the Display Field). For the Delete Rule, the available choices are Cascade, Restrict, and Set To Null. Whether you can edit a property and what values are available depends on the nature of the relationship and the objects.
6. Click **Create Relationship**.

7. Click **Save** to save the new relationship.

The new relationship is displayed in the Relationships list. If you specified many-to-many as the cardinality, the new intersection business object is also displayed.

To edit a relationship, click the **Edit** icon next to the relationship name and make the desired changes. Some fields cannot be edited and are disabled. You can specify the Display Label and select the Display Field. You may be able to select the Required check box for the field, and you can specify a Delete Rule. The following Delete Rules are available:

- **Restrict**: you aren’t allowed to delete a parent that has children (the default rule for a one-to-many relationship)
- **Cascade**: when you delete a parent, the children of that parent are automatically deleted (the default rule for an intersection object)
- **Set To Null**: when you delete a parent, the parent relationship field in the children is set to null

### Add Fields to Business Objects

You can use the Fields tab of the artifact window to create fields for your business objects.

You can create new fields for your business object while it is in Development status.

To add a new field:

1. Open the **Fields** tab of the business object that you want to edit.
2. Click the **+ New Field** button.
3. In the **Label** field, enter the name of the field.
The **Field Name** value is filled in automatically based on the field label.

4. Select the **Type** of the field.
   
   Available types are String, Number, Boolean, Datetime, Date, Time, Reference, Email, Percentage, Phone, and Uri. For a Reference type, you'll need to specify the Referenced Business Object and the Display Field.

5. Click **Create**.

**Edit Fields of Business Objects**

You can use the Fields tab of the artifact window to modify the fields of your custom business objects.

You can edit the fields of business objects when your application is in Development status. You can edit fields to specify constraints and create validation rules for fields.

To edit the properties of a field:

1. Open the **Fields** tab of the business object that you want to edit.
   
   The Fields tab displays a table that lists all the fields that are defined in the schema for the business object.

2. Select the row of the field that you want to edit.
   
   When you select a row in the table, the editor displays the properties that you can edit.

3. Edit the properties of the field.
   
   The properties that are available will depend upon the data type of the field. For example, you might be able to specify an initial value for the field using a static value, an expression, or a formula. You can also specify field constraints.
Add a Field for Aggregating Data

Use aggregation fields in your custom business objects to aggregate the data of related business objects, for example, to calculate and store the total number of items in an order.

You can use an aggregation field to calculate and store the values of simple operations such as calculating an average or sum, or counting the number of related business objects. In the Fields tab of the Business Objects page, you can edit the properties of a field to specify the fields of the related object that you want to aggregate.

To add an aggregation field:

1. Open the Business Objects page and click the Fields tab of your business object.

   Your business object must have a related object with fields that can be aggregated using one of the available aggregation functions.
2. Click **New Field** and create a field of type Number.

3. Select the new field in the list and, if necessary, click **Edit** to open the properties editor.

4. Select the **Aggregate from related object data** option to open the Create Aggregation dialog box.

5. Select the object to aggregate, the aggregation function, and, usually, the field to aggregate, then click **Save Aggregation**.

You can choose from the following aggregation functions: Average, Count, Maximum, Minimum and Total. You can also select the **Filter Related Object Records** check box to create a filter to limit the fields that are aggregated.

After you click **Save Aggregation**, the aggregation value is saved in the field. You can edit the aggregation by clicking the **Edit** icon next to the aggregation formula. When the fields are populated, you can see the value in the Data tab.

### Add a Formula to a Field

You can edit fields of business objects to add formulas for calculating the value stored in the field.

You can create a formula to calculate a numerical value, such as a percentage, or you can create a Groovy expression that uses available fields to generate a value. For example, you can concatenate strings stored in local fields 

(firstName + ' ' + lastName) or determine a value based on a comparison or logical expression (qualityLevel != 5).
To add a formula for calculating a value:

1. On the Business Objects page, click the Fields tab of the business object you want to modify.
2. Select the row of the field that you want to edit.
3. If necessary, click the Edit icon above the table to open a dialog for editing the field.
4. Select the Calculate value with a formula option to open the Field Formula expression builder.
5. Enter the formula in the text area.

The formula that you enter must be a valid expression. You can specify operands by typing in the text area or by selecting the Insert arrow for a field in the list of Available Fields. You can click an operator in the toolbar to add it to the formula.

6. Click OK to save the formula and close the dialog.

The formula is displayed in the property inspector.

How to Format a Date Field

You can format date fields of business objects to match the format you need.

When first added to a business object, date fields use their default formatting. To format a date field:

1. Drag an inputDate component from the component palette and drop it on top of the date field.
2. Set that field's converter property to match the format you need.

Access and Secure Business Objects

Apps in your visual application and other clients access your business objects through REST endpoints. You can configure the security settings of each business object to
control the user roles that can access the endpoints and the types of operations they can perform.

Topics:
- Work with the Endpoints Tab of a Business Object
- View the Endpoints of Business Objects
- Secure Business Objects
- About Allowing Access to the Catalog API

Work with the Endpoints Tab of a Business Object

For each business object, the Oracle Visual Builder user interface displays information about the endpoints you can call in code.

The Resource APIs node displays the URIs you can use to retrieve metadata and modify data in the Development, Staging, and Live databases.

The endpoints for business object are displayed in a tabular format. The display includes the HTTP method, the endpoint URI, an endpoint name that you can use in code, and a description of the endpoint.

A filter field at the top of the page allows you to view a subset of the endpoints.

For each custom business object, five default endpoints are created:
- Two GET endpoints, to retrieve one or all business object instances
- A POST endpoint, to create a business object instance
- A PATCH endpoint, to update a business object instance
- A DELETE endpoint, to delete a business object instance

If the business object refers to other business objects, endpoints are provided that enable you to retrieve, create, delete, and update those business objects.
You can click an endpoint in the list to view the endpoint’s details, for example, details about the endpoint’s settings and the headers sent in the request. The details are displayed in read-only mode, but you can use the Test tab to see the response to requests sent with parameter values that you supply.

### Note:
The REST endpoints for business objects support using URL parameters when retrieving resources, for example, to filter payloads using a query parameter to control the behavior of the data retrieved.

### View the Endpoints of Business Objects

Open the Endpoints tab of a business object to view details about the available endpoints of the business object.

To view the endpoints for your business object:

1. Select the business object in the Business Objects tab in the Navigator.
2. Open the **Endpoints** tab for the business object.
Secure Business Objects

Authentication roles can be used to secure the data stored in business objects.

By default, the business objects in your application are accessible to all users that can access the application. To secure the data stored in objects, you can use authentication roles and user roles to restrict a user’s access to view, create, update and delete operations. For custom business objects, you can configure role-based access for the individual operations. Users can only perform the operations and interact with the business objects associated with the role that the user has been assigned.

To allow anonymous access to the data in a business object, for each operation you must explicitly set the permissions granted to the Anonymous User authentication role.

To enable role-based security for a business object:

1. Select the business object you want to secure.
2. Open the Security tab of the business object.
3. Click the Role-based security icon to enable security for the object.
   
   When you enable role-based security for a business object, you see a matrix of the existing application roles and the business operations that can be performed. By default, when you enable security all existing application roles are permitted to perform all operations. If you create a new application role, permissions to perform operations are disabled for the new application role and must be enabled manually.

4. Select the operations that can be performed by each authentication and user role.
   
   When configuring security for custom business objects, you can enable or disable permission for each operation.
You can further define security at the row level for View, Update, and Delete operations by using a query builder to define conditions. To specify which users the conditions apply to, select the user role in the table. You can select **Allow if user created the row** from the action menu to limit an operation to the user who created the row.

The action menu has Cut and Copy selections so that you can move conditions from one role or operation to another.
When configuring security for business objects exposed by an external service, you can only enable or disable permission for all operations or for none.

About Allowing Access to the Catalog API

You can configure the security settings of your application to allow other applications access to the business objects in your application.

The custom business objects of your application can be consumed by external clients through their REST APIs. You can view the location of each of the catalog APIs in the Business Objects tab of the visual application’s Settings editor. In the same tab you can also set the security options for accessing the business object APIs and generate an access token.

Descriptions of the custom business objects in your application are available at the API URLs in the Catalog API panel. The panel displays separate URLs for the Development, Staged and Live versions of the application. The Catalog API URLs provide a minimal description of just the custom business objects exposed in the application. Though the URLs for the Staged and Live applications are provided for development purposes, they will not provide any results until the applications are staged or published.
Each of your Development, Staging and Live versions of your visual application have their own catalog APIs that expose the REST endpoints in the application that can be consumed by other applications. Accessing the catalog using the Catalog API URLs requires authentication. You can access the APIs using an access token and by using Basic Auth.

The Security pane of the Business Objects tab in the Settings editor contains the following options that you can enable for allowing access to the business object APIs:

- Allow anonymous access to business objects describe endpoint
- Enable basic authentication for business object REST APIs

If you choose to allow anonymous access to the Describe endpoint, external clients accessing the endpoint will still need to add the header "Authorization: Public" to the
request. The header is injected automatically for requests sent from your visual applications.

Access to the data in business objects is based on authentication and user roles. For each business object you need to explicitly enable role-based security and specify the operations that each defined authentication and user role can perform. You configure the security settings in the business object's Security tab. See Allow Anonymous Access.

**Note:**

Applications in other domains might need to be added to the CORS whitelist of origins permitted to access applications in your domain. An administrator can add domains in Administrator Settings.

Additionally, for requests to access your APIs that are not made through a browser, the request might need to be explicitly modified to include an Origin header that matches the domain in the CORS whitelist. A more advanced alternative would be to add CSRF headers to POST requests that include the current CSRF token value and the session cookie so the server can match the token from the request with the one in the session cache.

Get an Access Token for Authentication

To access the APIs for the catalog or custom business objects from outside Oracle Visual Builder, you can get a bearer token to use with various authentication methods.

In the design-time, you can use the token to access any of your app's endpoints. At runtime you can use the token for reading the data in the app's business object.

When authentication is handled by IDCS, you can use the token with connections that are authenticated with OAuth. You cannot use the token with connections to Fusion Applications.

You can use the token with the following authentication methods:

- Oracle Cloud Account
- User Assertion OAuth 2.0
- Client Credentials OAuth 2.0
- Resource Owner OAuth 2.0

To generate a bearer token:

1. Open the Business Objects tab in the visual application's Settings editor.
2. Click **Get Access Token** in the Security pane.

The access token is generated and is displayed in the Access Token Value field. You can now copy the token and use it when accessing your application's APIs.

### Create Rules for Business Objects

Most applications require rules for business objects that are more complex than can be expressed in a simple validation expression on a field. For each custom business object in your application, you can create custom rules that validate objects and fields and that trigger actions.

**Topics**

- About Adding Business Rules
- About Triggers for Business Objects
- About Object Validators for Business Objects
- About Field Validators for Business Objects
- About Object Functions for Business Objects
- Log Scripting Events

### About Adding Business Rules

You can use the trigger designer and code editors in the Business Rules tab to create business rules for your custom business objects.
On the Business Objects page, you use the Business Rules tab to create and edit business rules for your custom business objects. The rules for the selected business object are grouped under the following tabs in the Business Rules tab:

- Object Triggers
- Field Triggers
- Object Validators
- Field Validators
- Object Functions

Each tab displays a list of the existing business rules and contains a button for creating a new rule. For each business rule you can use the Business Rules options menu to copy and delete rules or to open the editor. You can toggle the state of a rule by selecting and deselecting the Active checkbox.

To add a new rule for a business object, select the object for which you want to add your rule and then open the Business Rules tab. In the Business Rules tab, open the tab for the type of rule that you want to add and click the button to create a new rule, for example, New Field Trigger. When creating a rule, you need to specify a name, and, depending on the type, you might also need to specify other rule properties.

Note:
You can only add business rules for custom business objects.

To define validation rules and object functions, you can use a code editor to write your Groovy scripts. To create triggers, you use the trigger designer, a visual editor for creating sequences that execute actions on your business objects. For additional help on writing Groovy scripts, see the Groovy Scripting Reference.
About Triggers for Business Objects

Triggers are scripts that you can write to complement the default processing logic for custom business objects. A trigger defines behavior that happens in response to a specific business object event, for example, inserting or updating a record, or in response to a field value change.

In the Business Rules tab of a business object, an object trigger refers to the sequence of actions that starts when a specific event occurs.

A field trigger refers to the sequence of actions that starts when a field value changes.

The trigger designer provides a visual representation of your sequence where you can define the conditions that determine the actions that will be executed. For each criterion you can assemble a list of actions composed of functions and Groovy scripts.

Topics:
- About Object Triggers
- About Field Triggers
- Add an Action to Send Email Notifications
- Convert a Trigger to Editable Code
- Build Conditions for Triggers

About Object Triggers

You can create an object trigger to specify a sequence of actions that starts when a specific event occurs.

A typical event triggering the sequence is adding or updating a record in the business object. When you create a trigger you specify the triggering event and the actions that are then executed. You can refine the sequence so that actions are executed only when some criteria are met. A sequence can define multiple criteria and multiple actions.

To create an object trigger for a business object:

1. Select the business object to which you want to add an object trigger.
   Triggers can be defined only for custom business objects.
2. Select the Business Rules tab of the business object.
3. Click the Object Triggers tab to see a list of all object triggers that are defined for the business object.
4. Click New Trigger, and in the Create Trigger dialog box, type a name to identify the trigger, and select the Start Event for the sequence. You can modify these later if you want.
5. Click **Create Object Trigger** in the dialog box to open the trigger designer.

You use the designer to build a sequence of criteria and actions that are triggered by the Start Event. No criteria or actions are defined when you first open the designer.
6. Click **Create New Criteria** (the plus sign) and choose a criteria type.

Three types of criteria are available in the dialog box. If you select **Execute Conditionally**, you can define the conditions that must be met to execute actions that you define. If the conditions are not met, the actions are skipped and the sequence advances to the next step.
7. If you specify Execute Conditionally, follow the instructions to build conditions for the trigger.

8. Click the Add Actions box, then specify a Name for the Action Trigger and click Add Actions.

The Configure Actions page opens when you click Add Actions. You can add one or more actions to the list by dragging predefined functions and custom Groovy scripts into the list.
If you drag Custom Groovy Code into the list of actions, you'll need to click **Edit Custom Code** and type your script in the editor. If you drag Send eMail Notification into the list, you'll need to specify email details such as the recipients, the sender and the contents of the message.

9. Click **Done** when you complete your list of actions.

You can continue to add more criteria nodes and actions to build up the sequence for the trigger. When you select a criteria node in your sequence you can use the Property Inspector to edit the criteria name, type and conditions. Depending on the criteria type that you select, you might need to specify conditions that need to be satisfied before the corresponding actions are executed. You can use the Conditions Builder to set conditions, or you can use the code editor to write custom code.
You can click Code Editor to view the read-only code that is generated by the designer.
10. When you are finished designing the trigger, click the **Object Triggers** tab to return to the Object Triggers page.

**About Field Triggers**

You can create a field-level trigger to define conditions that apply whenever a specific custom business object field changes in value.

In contrast to an object trigger, which defines conditions that apply when a specific event happens, a field trigger applies conditions when a field value changes.

To create a field-level trigger for a business object:

1. Select the business object to which you want to add a trigger.
   
   Triggers can be defined only for custom business objects.

2. Select the **Business Rules** tab of the business object.

3. Click the **Field Triggers** tab to see a list of all field triggers that are defined for the business object.
4. **Click New Field Trigger**, and in the Create Field Trigger dialog box, enter a name to identify the trigger, and select the field name from the Field drop-down list.

![Create Field Trigger dialog box](image)

5. **Click Create Field Trigger** in the dialog box to open the trigger designer. You use the designer to build a sequence of criteria and actions that are triggered by a change in the field value. No criteria or actions are defined when you first open the designer.

6. **Click Create New Criteria** (the plus sign) and choose a criteria type.

7. If you specify Execute Conditionally, follow the instructions to build conditions for the trigger.

8. **Click the Add Actions box**, then specify a **Name** for the Action Trigger and click **Add Actions**.

   The Configure Actions page opens when you click Add Actions. You can add one or more actions to the list by dragging predefined functions and custom Groovy scripts into the list.

   If you drag Custom Groovy Code into the list of actions, you'll need to click **Edit Custom Code** and type your script in the editor. If you drag Send eMail Notification into the list, you'll need to specify email details such as the recipients, the sender and the contents of the message.

9. **Click Done** when you complete your list of actions.
You can continue to add more criteria nodes and actions to build up the sequence for the trigger. When you select a criteria node in your sequence you can use the Property Inspector to edit the criteria name, type and conditions. Depending on the criteria type that you select, you might need to specify conditions that need to be satisfied before the corresponding actions are executed. You can use the Conditions Builder to set conditions, or you can use the code editor to write custom code.

You can click Code Editor to view the read-only code that is generated by the designer.

10. When you are finished designing the trigger, click the Field Triggers tab again to return to the Field Triggers page.

Add an Action to Send Email Notifications

Drag Send eMail Notification into your list of actions when you want to send an email notification that is triggered by a business object event.

If you add Send eMail Notification to your list of actions, you'll need to specify the email template for the message and the message recipients in the Configure Actions dialog box. You can create your own email template or use an existing one.

To add a Send eMail Notification action to your list of actions:

1. For new and existing triggers, click Add Actions in the trigger designer to open the Configure Actions dialog box.

   The Send eMail Notification action is available in the list of Suggested actions and under Other Scripting actions.
2. Drag the **Send eMail Notification** action into the list.

   After adding the action to the list, select the email template that you want to use, and then specify the email Recipients.
3. Select the email template.

You can select an existing email template from the dropdown list or create a new email template. If you select an existing template, you can click Edit to modify the template.

Depending on the template that you select, you might need to supply additional parameters that are used when generating the email. For example, you might want to specify a reference as a parameter used to generate the email subject or in the body of the message. If a template uses parameters, you'll need to define the values of the parameters, or you can edit the template to remove the parameters that you do not want to use. Parameter values can be a static value, a Groovy expression or a reference to a field in the business object. In the following example, you could replace Parameter1 and Parameter2 with field names from your business object.
4. Define values for the Recipients.

You can use static values for the Recipients, or the values can be generated with a Groovy expression or reference to a field in the business object. You can use the drop-down list next to the Recipients field to select the type of value in the field.

Convert a Trigger to Editable Code

To edit the entire trigger script in a code editor instead of using the visual trigger designer, you need to convert the script generated by the trigger designer.

When you create a trigger script in the visual trigger designer, you build up the sequence by creating groups of actions that are performed when criteria are met. The trigger designer provides tools to help you build the sequence, and you can use custom code to define criteria and actions individually, but you cannot edit the entire trigger script. If you want to freely edit the entire script in a code editor, you need to convert to code the script generated by the trigger designer.
Note:

You will not be able to edit the trigger script in the visual trigger designer after it is converted to code. After a script is converted, it cannot be converted back to script that can be edited in the visual designer.

You can view the entire script generated by the trigger designer by clicking in the designer. The script displayed in the code editor is read-only. To edit the script in a code editor, click Convert to Custom Code Trigger.

Build Conditions for Triggers

If you select Execute Conditionally as the criteria type for a trigger, the Conditions Builder can help you specify the conditions that need to be satisfied before the actions are executed.

When you set a criterion to execute conditionally, you use the Conditions Builder to define the set of conditions that determine when actions will be executed. You open the Conditions Builder by selecting the criterion in the designer and clicking Add Conditions in the Property Inspector.
To set up conditions in the Conditions Builder, you must select a field, select an operator, and set a value. The Condition Builder provides menus for selecting the fields in the business object, selecting operators, and helping you specify values. When specifying values, you can choose to use a static value, a field reference or an expression. You can create complex conditions by adding multiple conditions and grouping conditions together.
About Object Validators for Business Objects

An object-level validation rule is a constraint you can define on any custom object. The rule is used to evaluate the object when attempting to submit an object.

An object-level rule is appropriate when validation requires using two or more fields. Validation using an object-level rule ensures that regardless of the order in which the user assigns the values, the rule will be consistently enforced.

The expression or script that defines the rule must return a boolean value that indicates whether the object is valid. The object is saved if all the rules validating the object return true. If any of the rules return false, the error message of the failed rule is displayed and the object is not saved. If the rule returns true, then the object validation will succeed so long as all other object-level rules on the same object return true. For example, this type of validation would be needed when specifying a value for one field in a form requires that a value is also assigned to another field (for example, selecting ‘High’ in a Priority field requires a name is entered in the Assignee field).

To create a validation rule for a business object:

1. Select the business object for which you want to create the rule.
2. Select the Business Rules tab of the business object.
3. Click the Object Validators tab.

You see a list of all object validators that are defined for the business object.
4. Click + New Object Validator and enter a validator name to identify the rule, and then enter the error message to be displayed if validation fails. You can modify these later if you want.

5. Click Create Object Validator in the dialog box to open the editor.

6. Create your rule by typing in the editor and by using the business object fields and functions in the palettes. Use the palettes to help you add fields and functions that you might use to create your rule.

   Click the arrow next to the function or field in the palette to insert it at your insert cursor in the editor. When you select a function in the palette, a description of the function and an example of how to use it are displayed in the palette. Any object functions that you created for the business object will be listed in the Functions palette.

7. Click the Object Validators tab again to apply your rule to the object and exit the editor.

About Field Validators for Business Objects

A field-level validation rule is a constraint you can define on any custom field. The rule is used to evaluate the value of the corresponding field each time a new value is submitted.

A field-level rule is appropriate when the rule that is to be enforced depends only on the new value being set. At runtime your field validation rule is executed before the field’s value is saved.

The expression or script that defines the rule must return a boolean value. The value is saved if all the rules validating the field return true. If any of the rules returns false, the error message of the failed rule is displayed and the new value is not saved. For example, when a form has several fields, the values for all the fields must pass all the validation rules before any new values are saved.

To create a field validation rule for a business object:

1. Select the business object for which you want to add the new rule.

2. Select the Business Rules tab of the business object.

3. Click the Field Validators tab.

   You see a list of all field validators that are defined for the business object.
4. Click **New Field Validator** and type a name to identify the rule, the field that the rule will validate and the error message that is displayed if validation fails. You can modify these later if you want.

5. Click **Create Field Validator** in the dialog box to open the editor.

6. Create your rule by typing in the editor and by using the values and functions in the palettes. Use the palettes to help you add field values and functions that you might use to create your rule.

   Click the arrow next to the function or value in the palette to insert it at your insert cursor in the editor. When you select a function in the palette a description of the function and example of how to use it are displayed in the palette. Any object functions that you created for the business object will be listed in the Functions palette.
The Field Values palette contains the variables **newValue** and **oldValue**. Your script can use **newValue** to reference the new value that will be assigned if validation passes. To reference the existing field value, use **oldValue**.

7. Click the **Field Validators** tab again to apply your rule to the field and exit the editor.

### About Object Functions for Business Objects

An object function is useful for encapsulating business logic for a specific business object. After you define an object function, you can call the function by name from other scripts related to the business object.

When you create an object function on an object named Department, the following are true by default:

- Other scripts on the same object can call it.
- Any script written on another object that obtains a row of type Department can call it.

You can alter some of the default behavior by changing some of the properties in the Property Inspector. If the **Callable by External Systems** property is enabled, an external system working with a Department object will be able to invoke your object function. Enable this when the business logic it contains should be accessible to external systems. If instead you do not enable this property, then the object function can only be called by some other script on the Department object. Enable the **Privileged** property to indicate that the object function should run with the data security visibility of a privileged user. This can be necessary to enable the business logic to see rows of business object data when the current user might not have the right to access the data. If the **Privileged** property is not enabled, the script can only query rows that the current user has the right to access.

To create an object function rule for a business object:

1. Select the business object for which you want to create the rule.
2. Select the **Business Rules** tab of the business object.
3. Click the **Object Functions** tab.
   
   You see a list of all object functions that are defined for the business object.
4. Click **New Object Function** and type a name to identify the object function. You can modify this later if you want.
5. Click **Create Object Function** in the dialog box to open the editor.
6. Open the Property Inspector for the rule if not already open and select the **Function Return Type** from the drop-down list.
7. In the Property Inspector, click **Parameters** and specify the parameters for your function. Click **All Properties** to return to the Property Inspector.
8. Create your function by typing in the editor and by using the Business Object and Functions palettes. Use the palettes to help you add fields and functions that you might use to create your function.

Click the arrow next to the function or value in the palette to insert it at your insert cursor in the editor. When you select a function in the palette, a description of the function and example of how to use it are displayed in the palette.

9. Click the **Object Functions** tab again to add your function and exit the editor.
Log Scripting Events

When you are developing the rules for a business object, you may want to enable logging and use the log viewer to help you see exactly which events are triggered.

For example, you might have a Before Update trigger script that calls several object functions. In the Code Editor for object functions, you can add a `println` to your object functions to print a message to the log each time the function is called. If you are using the Configure Actions window to create and edit triggers, you can drag the Log Message action in the Other Scripting category into your action group to add a `println` to the action chain. By looking at the log, you can see the functions that were triggered based on the messages, as well as any exceptions in your script that are written to the log.

In addition to custom messages that you add to your code to log actions, the following actions are always recorded in the logs:

- Trigger starts
- Trigger ends

The viewer in the Logs window displays the most recent 250 log entries. The menu in the Logs window contains the following tools for working with the log entries.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter field</td>
<td>Type text in the filter field to filter out messages that do not contain the text.</td>
</tr>
<tr>
<td>Enable Logging</td>
<td>Deselect this check box to stop logging actions.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Click to ensure that the most recent log entries are displayed in the log viewer.</td>
</tr>
<tr>
<td>Export</td>
<td>Click to export your log as a text file to download to your local system.</td>
</tr>
<tr>
<td>Clear</td>
<td>Click to clear all the entries that are displayed in the viewer.</td>
</tr>
</tbody>
</table>

Enable Logging of Scripting Events

You can enable logging to record the events triggered by your custom business rules.

By default, logging is not enabled. To enable logging:

1. Click the **Logs** link at the bottom of the window to open the Logs page.
2. Select the **Enable Logging** check box.

After you enable logging, it remains enabled in your session for as long as you are logged in. If your session has expired, after you log in you will need to re-enable logging.
View and Edit Data in Business Objects

You can view the data associated with your custom business objects in any of your databases (Development, Staging, and Live) and, in some cases, the data retrieved from external services.

You can use the Data tab to view the data stored in fields of business objects.

For custom business objects, you can view the data no matter what status the application is in. For business objects from an external service, you can view the data only when the application is in Development status. Data for related objects is not displayed.

To view the data in a business object:

1. Select the business object that you want to view and open the Data tab.
   The table in the Data tab displays the data stored in the fields of the business object. The table displays the data for editable and read-only fields.

2. Select the arrow for a column to sort the data by that column, or expand the Query node and enter search criteria to display only the rows that meet your criteria.
   You can sort the data for a custom business object, but not for a business object from an external service.

Edit the Data in Business Objects

You can directly edit the business object data stored in your databases.

Use the Data tab to modify, export, and import the data in custom business objects. If you have the appropriate rights, you can also edit the data in top-level business objects for external services in the Data tab.

The Data tab displays a table with the data for each of the business objects in your application. You can add data by creating new rows and adding data to the fields in business objects. Use the tools in the toolbar to perform the following functions:

- Edit the data stored in the editable fields of an object by editing the individual rows
- Delete a row
- Export the entire table as a CSV file
- Import a CSV file or Excel spreadsheet (.xls or .xlsx) to replace or append data

To edit the data in a single row:

1. Select the business object you want to edit and open the Data tab.
   The table in the Data tab displays the data stored in the fields of the business object. For business objects provided by external services, the table displays the data in top-level business objects only.

2. Select the row that you want to edit and click the Edit Row icon in the table toolbar to open the Edit Row dialog.
The Edit Row dialog displays the editable fields in the business object and the current data stored in the selected record. Some fields, such as the creation date, are populated automatically and are not displayed in the edit dialog.

3. Make your changes in the dialog. Click **Save**.

   The validation rules for fields are enforced when you edit the data stored in the records.

### Edit Data in a Table

You can edit the data stored in a custom business object by importing a file. When you import the file, you can choose to replace all the current data or to add the data in the file as new rows appended to the existing data.

You cannot edit data in business objects for external services by importing a file.

To edit data in a table:

1. Open the **Data** tab of the custom business object that you want to edit.

   The table in the Data tab displays the data stored in the fields of the business object.

2. Click **Import From File** in the toolbar.

   The Import Data dialog provides options to either replace the existing data or to append the data in the file to the existing data.
When you append the data from a file, the ids for the new data are renumbered to prevent duplicating ids.

3. Upload the file by browsing your local file system or by dragging the file into the dialog box.

4. Select **Append** or **Replace** in the dialog box. Click **Import**.

5. Click **OK** in the success or failure dialog box.

**Work with the Data Manager**

Use the Data Manager to manage the data stored in your application during the development, staging and live phases.

To open the Data Manager, click the **Options** menu on the Business Objects page and select **Data Manager**.

**Topics**

- Manage Data During the Development Lifecycle
- Import Data From a File
- Import Data From a Database
- Export the Data in Your Database
- Resolve Problems When Importing Data
- Import Data from the Command Line
- Export Data from the Command Line
Manage Data During the Development Lifecycle

Oracle Visual Builder provides tools to help you migrate data between your databases and to import and export data.

When developing your application you might have three versions of your application, each in a different status: development, staging, live. Each version uses an independent database that is used for that phase of development, and during the development lifecycle you need to manage the data that is stored in each database. To populate your databases you can add data manually, migrate data between the development, staging and live databases, or import data from files. See Export the Data in Your Database and Import Data From a File.

Each database uses a schema to describe the fields of the business objects. In the development phase, the schema of your development database is modified as you modify the business objects in the application. The development database schema replaces the schema of the staging database when you stage the application, and the staging database schema replaces the schema of the live database when you publish the application.

Note:

You cannot use the Data Manager to manage the data for business objects for external services. The definitions for business objects provided by external services are stored in your database schema, but the data from the service is not stored in your database.

The following table describes the data typically stored in the database for each phase of the development lifecycle and the data management tasks performed during the phase.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
</table>
| Development | Your development database will typically only contain some basic data to help you while you build your pages. To check the behavior of the application, you might add some sample data manually by using the forms you created in your application or by editing the data in the Data tab. You will typically perform the following tasks with the data in your development database:  
  • Manually add and modify data in Live mode using the forms in your application  
  • Import sample data from a file |
<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Staging</td>
<td>Your staging database will typically contain a set of data that is as realistic as possible to be used when testing the staged version of the application. You can add data manually using your application’s user interface or import data from a file or database. When you are ready to publish your application, if the database schema has changed since the previous version, you will want to import the data from the live database into your staging database. This data is then copied to the live database when you publish the application. You will typically perform the following tasks in your staging database: • Add and modify data to test the application’s UI behavior and business logic • Import data from a file to test that the data and schema are compatible • Import data from the live database</td>
</tr>
<tr>
<td>Live</td>
<td>Usually you will not want to modify the data in your live database except as part of the publishing process if the data is copied from the staging database. If it is necessary to modify your data because you changed the structure of your application, it is important to make and test the changes on your staging database before you publish the application. During this stage you will typically perform the following tasks: • Export the data as a file or copy the data to the staging database using the Data Manager • Lock the application to prevent changes to the data • Unlock the application to enable changes to the data</td>
</tr>
</tbody>
</table>

Import Data From a File

You can replace the data in one or more of your custom business objects by importing CSV files and Excel spreadsheets. Use the Import from File tool in the Data Manager to simultaneously update the data for one or more business objects, for example, to import data for testing the application or in preparation for publishing the application.

To import data from a CSV file, you will need one CSV file for each of the business objects that you want to update, and the name of the CSV file must be the same as the business object. You can upload CSV files individually or upload a zip archive that contains multiple CSV files.

When importing an Excel spreadsheet (.xls or .xlsx), the spreadsheet can contain one or more sheets. The title of each sheet must be the same as the name of the business object that you want to update. If the data in a cell is calculated using a formula, only the data is imported. The formula is not imported.

When you use the import tool to replace the data in a business object, all the data for that business object is deleted from the database. If a field is defined in the schema for the object but no data for the field is contained in the file, the field is set to the default value, if there is one.
Each business object has five default fields: id, creationDate, lastUpdateDate, createdBy, and lastUpdatedBy.

The correct format for a Date field is yyyy-mm-dd (for example, 2006-06-17). If you edit a .csv file in Excel, Excel converts it to an incorrect format, as shown in the image above. To resolve this problem, you can use Format Cells in Excel to change the date format for the column; you need to specify a locale that supports yyyy-mm-dd, such as English (United Kingdom). Alternatively, edit the file in a text editor.

The import tool will not create or remove fields for business objects. You can use the Business Objects tabs to create or remove fields, to edit data, and to append data to a business object by importing a file.

To import data into a database:

1. Click the Options menu on the Business Objects page and select Data Manager.
2. Select the database that you want to update.
3. Click Import from File.
4. Drag the file into the Import Data dialog box.

Alternatively, click and locate the file on your local system.

5. Click Import.

When you import the file, you will see a success message if the import is successful. If there are any problems importing the data, you will see a message that describes the problem. For example, the message might list fields that were not imported because the fields were not defined in the schema. You can try to resolve the problem by comparing the fields in the schema to the columns in the CSV or spreadsheet file and
either modifying the schema in the Fields tab or modifying the data in the file. Import the file again to correct the data.

Import Data From a Database

You can import data by using the import tool to copy data from one database to another.

Each phase in the development lifecycle of your application uses an independent database for storing data. You can use the import tools in the Data Manager to import data from one database into another, for example, to import the data in your live database into your staging database.

To import data into a database:
1. Click the Options menu on the Business Objects page and select Data Manager.
2. Select the database that you want to update in the drop-down list at the top of the page.
3. Click the tile to import data into your database.
   The page contains several import options. Unavailable options are grayed out.
4. Click Import in the Import Data dialog.
   All the data in the target database is deleted and replaced when you import data. When the task is complete, a dialog opens that confirms that the data was successfully imported or warns you that there was a problem.

Export the Data in Your Database

You can export all the data contained in your database as CSV files. The export tool creates one CSV file for each of the custom business objects in your database and packages the files as a ZIP archive.

To export the database data as a CSV file:
1. Click the Options menu on the Business Objects page and select Data Manager.
2. Select the database that you want to export from the drop-down list.
3. Click the Export All Data tile to download a ZIP archive that contains CSV files with the data.
Alternatively, you can export the data contained in an individual custom business object in the Data tab for the business object.

Resolve Problems When Importing Data

When you import data from a file, if you see a warning message that the data was not imported or only partially imported, you might need to make changes to the file and import the file again.

To resolve problems during import, you might want to compare the data in your file to the data in the database. You can see the actual data in the database in the Data tab of your business object. You can also export the current data as a CSV file and compare the data using a tool on your local system.

When you import a CSV file with a Date field, dates must be in the standard ISO format, for example, 2017-09-31.

When you see a warning message:

1. Confirm that the name of the file or Excel workbook is the same as the name of the business object.
2. Compare the columns in the files to the fields in the business objects.
   
   Importing a file will not create fields in the database schema. Columns in the file are ignored if a field with that name does not exist in the business object. The import tool expects the data in the first row of the CSV file or Excel workbook to be the name of the field.
   
   3. Confirm that the format and type of the data in the file are the same as those specified in the schema.

Import Data from the Command Line

You can perform bulk import and export of data from the command line using Oracle Visual Builder APIs.

The Oracle Visual Builder APIs are accessible from the command line to an application’s team members using basic authentication, just as the data APIs can be used to query individual objects and perform single-row operations. For example, you can set up a cron job to import and export data to ensure that the data in two tables are synchronized.

To import data in a business object from the command line, you will need to upload .csv, .xls, or .xlsx files containing the data. The name of the file (for .csv files) or the workbook name (for .xls and .xlsx files) must match the object ID of the business object that you want to update. To import data for a single business object, you can upload a single file. When importing data for multiple business objects, you will need to upload a ZIP archive containing one or more .csv, .xls, or .xlsx files.

To import the development schema for version 1.0 of an application named MyApp, you would use a POST method with the following endpoint to update multiple business objects.

POST https://host:port/design/MyApp/1.0/resources/datamgr/import
For the query parameter `filename`, you will need to specify the name of your ZIP archive.

To import data for a business object named `MyObject` in version 1.0 of an application named `MyApp`, you would use a POST method with the following endpoint.

```
POST https://host:port/design/MyApp/1.0/resources/datamgr/import/MyObject
```

For the query parameter `filename`, you will need to specify the name of your file. Additionally, you can use the Boolean query parameter `append`, which, when explicitly set to `true`, will add the rows in your file as new rows in your business object. The default value is `false`, which results in the data in your file replacing the existing data in the business object.

### Export Data from the Command Line

To export the development data for version 1.0 of an application named `MyApp`, you would use a GET method with the following endpoint to get the data in all the business objects:

```
GET https://host:port/design/MyApp/1.0/resources/datamgr/export
```

The result of exporting the data for an application is a ZIP archive containing a `.csv` file for each business object in the application.

To export development data for the business object `MyObject` in version 1.0 of an application named `MyApp`, you would use a GET method with the following endpoint:

```
GET https://host:port/design/MyApp/1.0/resources/datamgr/export/myObject
```

The result of exporting the data in a single business object is a `.csv` file containing the data in the business object you specified.

---

**Note:**
To import or export data from the staging or live database, replace `/design/` with `/deployment/` in the path of your endpoint. If your application is live, use the options menu to lock the application before you import or export data. Unlock the application after you finish.

---

### Create Business Objects From a File

You can create new business objects by importing spreadsheet files and `.csv` files using the Import New Business Objects wizard.

The files that you upload are analyzed to determine the business objects that can be created. You may upload comma-separated value text files (.csv) or Excel spreadsheets (.xls, .xlsx). When using `.csv` files to create business objects, one business object is created for each file, and the name of the business object is based on the file name. When using `.xls` or `.xlsx` files to create business objects, one
business object is created for each worksheet in the file, and the name of the business
object is based on the worksheet name. If the worksheet contains one or more tables,
a business object is created for each table based on the table name, and the
worksheet name is ignored. The first row of .csv files, worksheets, and tables must be
a header row, and the column headers are used to determine the names of the fields.
The data in each column is parsed to help determine the data type for the field, but
you should confirm the suggested data type is correct in the Fields step of the wizard.

To upload multiple files, you need to create a ZIP archive containing the files you want
to import.

To open the Import New Business Objects wizard:
1. Click the Options menu on the Business Objects page and select Data Manager.
2. On the Data Manager page, click Import Business Objects.

To create business objects in the Import New Business Objects wizard:
1. Open the Import New Business Objects wizard.
2. In the Upload File step of the wizard, drag the file from your local system into the
wizard, or click in the upload box and locate the file on your local system.

After the upload is complete, the wizard displays a list of the business objects and
records found in the upload.

Click Next.

3. In the Business Objects step of the wizard, select the business objects that you
want to create.

The wizard displays a list of the business objects that can be created and the files
in your upload that they are based on. You can select which business objects you
want to create, and edit the names and object IDs of the new business objects.
Click Next.  

4. In the Fields step of the wizard, click the business object name to edit the names and types for each of the fields in the business object. The wizard displays tabs for each new business object. Each tab displays the fields that will be created in the business object, and a sample of the values stored in the field. For each business object you can edit the ID, display label, and data type of the fields, and specify whether they are required.

You can click the Data Type icon for each field to open a dialog box where you can modify the type. For Reference fields, you can select the related business object from an existing business object or from those that you are importing, and select the field in the related object to display.
Click **Finish**.

The wizard displays a list of the new business objects that were successfully imported.

---

**Work with the Business Object Diagrammer**

With the Diagrammer, you can create entity relationship diagrams for your business objects, and you can create and edit additional business objects.

To create a Business Object Diagram:

1. Click the **Diagrams** tab in the Business Objects pane, then click the + sign (**Diagram**).

2. In the Create Business Object Diagram dialog box, enter a name in the **Diagram name** field and click **Create**.
   
   An empty page for the diagram opens, along with a Property Inspector.

3. In the Property Inspector for the diagram, select the check boxes for the business objects you want to display, or click **Select All** to display all of them. If you have many business objects, you can create multiple diagrams to display them.

The Diagrammer shows the selected business objects and their user-defined fields. The name and type of each field are displayed. The Diagrammer also shows the relationships between the objects.
By default, the relationship diagram is displayed horizontally. To change it, right-click in the Diagram page and select **Layout**, then click **Vertical** to change the display to vertical.

You can use the Diagrammer to perform the following tasks:

- Click the name of a business object to see the Property Inspector for the business object, where you can view or edit the overview information for the object. If you double-click the name of a business object, you go to the Overview tab for the business object.
- Click a field of a business object to see the Property Inspector for the field, where you can edit the properties of the field.
- Click the triangle next to a business object name to collapse or expand the fields display.
- Click a relationship between two business objects to see the Property Inspector for the relationship, where you can edit the relationship.
- If the diagram displays many business objects, you can click **Find** to locate one of them.
- To specify how business object fields are displayed, right-click in the Diagram page and select **Fields** to display the Fields menu.

By default, any reference fields are displayed, and the number of visible fields is 10 (though you can scroll to view more).
– Select **Show Audit Fields** to display the fields that are automatically created when you create a business object.
– Select **20** to display up to 20 fields by default.
– Select **Show All** to display all fields, or select **Hide All** to hide all fields.

To delete a diagram, right-click the diagram in the Diagrams tab and select **Delete**.

**Create Business Objects with the Diagrammer**

You can use the Diagrammer to create new business objects.

To create a new business object:

1. Right-click in the Diagram page and select **+ New Business Object** from the menu. Alternatively, select **+ New Business Object** from the Options menu in the Diagram Property Inspector.

![New Business Object dialog box](image)

2. In the New Business Object dialog box, enter the business object name in the Label field and click **Create**.

   The Name value is filled in automatically based on the Label value.

3. To add a field to the new business object, right-click the object and select **+ New Field**. In the dialog box, enter the field name in the Label field and select the type, then click **Create**.

4. To delete a field, right-click the field and select **Delete**.

   Double-click a field to open the Property Inspector for the field, where you can edit its name, data type, and other properties.

**Create Relationships with the Diagrammer**

You can use the diagrammer to create new relationships between business objects.

To create a new relationship:

1. Right-click the business object name and select **+ New Relationship**.

   A dotted red line appears in the business object.

2. Drag the dotted red line to the other business object.

   The Create Relationship dialog box appears, with a new Reference field in the original object that has the name of the other object.
3. Select the cardinality for each object (Many or One).

   You can specify cardinalities of one-to-one and many-to-many between business objects, in addition to many-to-one and one-to-many. If you specify Many-to-Many, an intersection business object with two reference fields is automatically created. Its name is a concatenation of the two business objects. For an intersection business object, you can't deselect the Required check box in either of the reference fields, and the default delete rule is Cascade.

4. Edit the properties of the relationship field (the Id, the Display Label, the Default Display Field, the Required check box, the Delete Rule).

   For the Delete Rule, the available choices are Cascade, Restrict, and Set To Null. Whether you can edit a property and what values are available depends on the nature of the relationship and the objects.

5. Click Create.

   An arrow appears between the two objects. Mouse over the arrow to see a description of the relationship.

Right-click a relationship arrow and select Edit to make modifications in the relationship, or select Delete to remove the relationship.

The new relationship appears on the Overview page for each business object. You can edit and delete the relationship on that page as well as in the diagrammer.
Work with Service Connections

When you want to expose business objects from an external source in your visual application, you can add and manage connections to sources in the Service Connections tab of the Navigator.

Topics:

• About Service Connections
• Create a Service Connection
• Edit a Service Connection
• Configure Authentication for Service Connections

About Service Connections

When you want to expose business objects from an external source in your application, you need to register a connection to the external REST APIs as a service connection.

Each service connection describes the connection to a specific service, including the connection details and properties, and the specific REST endpoints provided by the service that you want to use in your applications.

You can create a service connection in the following ways:

• Selecting a service in your integrated service catalog of Oracle SaaS REST Services
• Providing a document that describes the external service
• Providing the location of a REST service endpoint of the external service

You use the Services tab in the Navigator to add new connections and manage your existing service connections. Selecting a service connection in the list opens a tab for the connection in the editor where you can view and edit the details of the connection and the endpoints that are available.
You can delete a service connection by right-clicking the service in the Navigator and choosing Delete.

For each of your service connections, you can use the following tabs in the connection editor to view and edit the connection’s details.

<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
</tr>
</thead>
</table>
| Service | The Service tab displays the Base URL and Service Id of the service. You can edit the name of the service as it appears in your visual application editors.  
When Server Only Connections is selected in the Service tab, the service connection can only be used by the server, for example, from a Groovy script. |
| Endpoints | The Endpoints tab displays a list of the endpoints of the service that you selected when you created the connection. You can use the tab to add and manage endpoints.  
You can click + Endpoint to open a menu where you can choose to add more endpoints from the service or to add custom endpoints.  
Each endpoint in the list has an options menu where you can choose to edit, duplicate or delete the endpoint.  
See Understand Data Access Through REST for more on the options and parameters that you can use to configure service connections. |
<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>The Authentication tab displays the authentication mechanism used to connect to the service, for example, authentication using OAuth. You can select the type of authentication used when connecting to the service. When Token Relay is enabled, requests are made directly from the web client instead of via the Oracle Visual Builder proxy. This can improve performance significantly in some situations. Authentication token relay is available for the following authentication types: Oracle Cloud Account, User Assertion OAuth 2.0, Client Credentials OAuth 2.0, and Resource Owner OAuth 2.0.</td>
</tr>
<tr>
<td>Headers</td>
<td>The Headers tab displays the headers written in the REST call to the service. You can add and edit headers in the tab.</td>
</tr>
</tbody>
</table>
| Transforms   | The Transforms tab displays the transform.js JavaScript file with the functions used to interpret requests to and responses from the service. You can edit the file to add transform functions. You typically use transform scripts to perform the following types of functions:  
- Pagination functions that limit the number of records that are displayed on a page in the REST client.  
- Filtering functions that define filter operations to specify the data to be returned and displayed in the REST client.  
- Sorting functions specify how to sort items returned from a collection resource. |
| Source       | The Source tab displays the Swagger description of the service’s REST API. The file contains the details about the connection settings, response and request definitions and other parameters that are used when designing your applications (Design Time Model) and when an application is published (Runtime Model). |

**Set the URL of Your Service Catalog**

You use the Services tab in your application's Settings editor to set the catalog URL and authentication mechanism for connecting to the catalog of services provided by a service instance.

Your service administrator can specify the catalog URLs for the services that are used to populate the Service Catalog of applications in the instance. If no catalog URL has been set for your instance, or you want to override the instance’s settings to specify a different catalog URL for your application, you can add the catalog URL and set the authentication mechanism in your application’s Settings editor.
The list of REST services in the Service Catalog is populated with services provided by a service instance, for example, Oracle Fusion Applications. For Oracle Fusion Applications, the interfaceCatalogs endpoint of a Fusion Applications instance provides a list of available services from the most recent ADF Describe. You cannot specify an API version. To retrieve the list of services, you supply the URL for the interfaceCatalogs endpoint (for example, https://my-fa-instance.example.com/otherResources/latest/interfaceCatalogs) in the Services tab of your application's Settings editor. You also need to specify the authentication mechanism for accessing the instance. Your visual application only supports one URL for the interfaceCatalogs endpoint at a time. It is possible to edit the base URL of a service after it is registered.

To retrieve the list of services from a Mobile or Integrations instance, you supply the URL of the Cloud Server and specify the authentication mechanism for accessing the instance. You will typically use an OAuth authentication mechanism.

If you do not see any services in your catalog after confirming the Catalog URL and authentication mechanism are correct, you should consult your administrator to confirm that you have the proper credentials and that your user role is authorized to access services from the service instance.

To set the URL of the Service Catalog for your application:

1. Open your web or mobile application and choose Settings in the application's Options menu in the toolbar.

Alternatively, on the Oracle Visual Builder home page, locate the application where you want to change the settings and choose Settings in the application's Options menu.

2. Open the Services tab in the Settings editor.

3. Select the type of service that you want to add.
4. Enable **Override Tenant settings** if you want your application to use the URL and authentication mechanism you provide in the application’s settings instead of the settings for the instance.

5. Type the catalog URL for the endpoint for retrieving the list of services available from the instance.

6. Select the authentication mechanism for the instance.
   
   You can override these settings in the Authentication tab of the service connection.

7. Click **Initialize** to initialize a new connection, and **Test** to confirm that connecting to the service succeeds using the details you provided.

---

**Create a Service Connection**

You can create service connections by selecting a service in your catalog, by providing a specification document for a service, or by providing the location of a service endpoint. After specifying the service you want to use you can select which service endpoints you want to expose.

**Topics:**

- Create a Service Connection from the Catalog
- Create a Service Connection from a Service Specification
- Create a Service Connection from an Endpoint

---

**Create a Service Connection from the Catalog**

You create a connection from the Service Catalog when you want to connect to associated Oracle Cloud services in your identity domain.

The Services Catalog is automatically populated with a list of services and endpoints available to you from your associated Fusion Applications, Integrations or other service instances. To access a service that is not listed in your catalog, you can try to
create the service connection by using a service specification (ADF BC REST, Swagger) or by specifying an endpoint URL.

If you do not see any services in the Service Catalog, you should confirm the following:

- You have supplied a URL for the service instance providing the services and selected the correct authentication mechanism. If you are connecting to a Fusion Applications or Integrations service instance, use the Services tab in the app's Settings editor to check the URL and authentication mechanism.

- You are authorized to access the service with your credentials. Contact the service's administrator to confirm that your credentials are authorized.

- The service is currently available. Use a command line tool to check the connection and status of the service.

To create a connection to a service in the catalog:

1. Open Service Connections in the Navigator and click **Create Service Connection** (+).

2. Click **Select from Catalog** in the Select Source step of the Create Service Connection wizard.

3. Click the tile for the service catalog.

The window displays tiles for the available service catalogs.
4. Select the service and endpoints you want to add. Click Create.

When you add connection to Integrations, you can select any of the activated REST endpoints provided by your Integrations service. When you add connections to Fusion Applications services, you first choose a service in the catalog and then select the REST endpoints.

The Select Endpoints pane displays a list of the endpoints available for each resource provided by the service.
Tip:

You can select a top-level object to select all endpoints for that object, or select individual endpoints to improve performance.

After a service connection is created, you can select it in the Navigator to open the connection in the editor and edit the endpoints associated with the service and other connection details.

The connection details are generated automatically by the wizard. You should exercise caution when editing the connection details to avoid changing details that would cause the connection to fail.

Create a Service Connection from a Service Specification

You create a connection from a service specification when you know the URL of the Swagger or ADF file that describes the service, or you can upload the describe file from your local system.

To create a connection from a service specification:
1. Open Service Connections in the Navigator and click **Create Service Connection** (+).

2. Click **Define by Specification** in the Select Source pane of the Create Service Connection wizard.

3. Type a name for the connection in the **Service Id** field.
   
   The Service Id is the display name for the connection in your application.

4. Select the **API Type** and the location of the **Service Specification** document. Click **Next**.
   
   If you select Web Address for Service Specification, enter the URL of the service description on the text field. If you select Document, upload the file describing the
service from your local system by dragging the file into the dialog box or by navigating to the location in your local file browser.

5. Select the resources and endpoints you want to add. Click Create.

**Tip:**
The Select Endpoints pane displays a list of the endpoints and child objects available for each resource provided by the service. You can select a top-level object to select all endpoints for that object, or expand the top-level object node and select individual endpoints to improve performance.

After a service connection is created, you can select it in the Navigator to open the connection in the editor and edit the endpoints associated with the service and other connection details.

**Create a Service Connection from an Endpoint**

You create a connection from an endpoint when you know the base URI of a service and can provide the necessary parameters for connecting to the service, such as authentication details and an example of the body of the Response.

To create a service connection from the URL of an endpoint:

1. Open Service Connections in the Navigator and click **Create Service Connection** (+).

2. Click **Define by Endpoint** in the Select Source pane of the Create Service Connection wizard.

3. Select the HTTP method and type the URL of the endpoint. Click **Next**.
If you know what type of action you want to perform, select it in the Action Hint dropdown list.

4. In the **Service** tab, confirm that the Service Base URL, Service Name and Service ID are correct.

   Depending on the endpoint that you specified, these values might be automatically populated for you.

5. Open the **Authentication** tab and select **Enable authentication / proxy** to enter authentication details, if required.

   Enable authentication / proxy is disable by default.

6. Open the **Request** tab to add Headers and URL parameters to the request.

   Depending on the endpoint, you might want to add custom headers or path or query parameters that are passed as part of the Request.
7. Open the **Response** tab and enter the response body for the endpoint.

The Response tab contains a text area where you can paste in an example of the body of the response. The example in the Response is the definition of the structure of the returned object. If you do not have an example, you can use the Test tab to send a request to the service and view the response. You can edit the response body in the text area in the Response tab.

8. Open the **Test** tab to test your request and view the response from the endpoint.

Click **Send** to view the Response body and headers that will be received to confirm that you will receive the data you want.
You can experiment with the different request parameters until you achieve the response you want.

You use the Test tab to test the service and view the response to the request that will be sent to the endpoint based on the settings in the other tabs. If your response returns an error you will need to check the details of your connection, for example, to ensure that you are using the correct credentials or that the service uses a valid SSL certificate. For development purposes you can try to enabling Token Relay for authentication if the service supports it.

9. Click **Copy to Response Body** to copy the current response into the text area in the Response tab, where you can edit it further if you require

10. Click **Create** when you are satisfied with the parameters of your request and the response.

**Tip:**

After you add an endpoint from the service, you can add more endpoints from the same service by clicking **+ Endpoints** in the Endpoints tab of the connection and choosing From Original Service in the menu.
Edit a Service Connection

After you create a service connection, you can edit the service connection, for example, to add and remove endpoints, modify requests and add functions for filtering and sorting responses.

Topics:

- Edit Service Endpoints
- Add More Endpoints to a Service Connection
- Edit the Authentication Details of a Service Connection
- Add Filtering, Sorting and Pagination Functions

Edit Service Endpoints

After you create a service connection and select the service endpoints, you can edit the endpoint settings, request parameters and response for each endpoint in the Endpoints tab.

If you edit an endpoint after you have created a type from it, you will need to manually edit the type to use any of the changes to the endpoint. A type created from an endpoint is not updated automatically when the endpoint is modified.

To edit a service endpoint:

1. Open the Endpoints tab of a service connection.
Tip:
You can use the Replace definitions button to update the service definitions of all selected endpoints with the latest updated definitions from the service. The Replace Definitions button is available when the registered service connection is registered via the Catalog (Fusion Applications) or a service specification document ("ADF Describe"). The button opens the service wizard and you are prompted to verify the definitions that will be replaced.

2. Click the endpoint you want to edit.
   For each endpoint the editor provides tabs for editing the endpoint’s settings, the request sent to the endpoint and the structure of the response.

3. Click the service connection link in the breadcrumb to return to the list of service endpoints.

Add More Endpoints to a Service Connection

After you create a service connection, you can add more endpoints from the same source or add custom endpoints.

To add an endpoint to a service connection:
1. Open the Endpoints tab of a service connection.
2. Click **+ Endpoint** and select **Custom** or **From Original Service** in the dropdown list.

3. Select or define the new endpoint in the Add Endpoint dialog box.

The options in the Add Endpoint dialog box for adding an endpoint depend upon the type of service connection and if you choose Custom or From Original Service. For example, if your service connection was created for a service from the catalog, if you choose to add an endpoint from the original service you can choose from the list of endpoints available in that service. If your service connection was created from an endpoint, you will need to specify details about the request and response to add a new endpoint from the same source.
Edit the Authentication Details of a Service Connection

After a service connection is created, you can edit the authentication details in the Authentication tab.

You might want to edit the authentication details when the authorization requirements of your app change, for example, you need to allow anonymous access to the service, you need to override the Tenant Settings for a service, or the authentication mechanism used during development won't work when you publish the app.

To edit a service's authentication settings:

1. Open the **Authentication** tab of the service connection.
2. Select **Enable authentication / proxy**.

   If the connection is to a service in your Service Catalog, you can select the Override option in the Authentication tab to override the default authentication options for the service.
3. Select the Authentication Mechanism for authenticated users from the dropdown list and provide the required authentication details.
4. (Optional) Select **Allow anonymous access** and select the Anonymous Authentication Mechanism from the dropdown list and provide the required authentication details.
Add Filtering, Sorting and Pagination Functions

You can write custom JavaScript functions to interpret requests to and responses from the service.

The Transforms tab contains a JavaScript editor that you use to edit the transform.js file that contains your transform functions. You typically use transform scripts to perform the following types of functions:

- Pagination functions that limit the number of records that are displayed on a page in the REST client.
- Filtering functions that define filter operations to specify the data to be returned and displayed in the REST client.
- Sorting functions specify how to sort items returned from a collection resource.

To add transformation functions to a service connection:

1. Open the Transforms tab of a service connection.
2. Select **Enable filtering, sorting and pagination transforms** to display the JavaScript editor.

When you enable the transform functions, the JavaScript editor displays some code fragments that provide examples of transform functions for requests and responses. The code is provided only as an example and should not be used in your application.

You should deselect the checkbox and disable the transform functions if you are not going to write any functions.

---

**Configure Authentication for Service Connections**

How you configure authentication for connections to services depends on the type of mechanism you want to use, as well as what the external service supports. Depending on the authentication you choose, an administrator might need to configure settings in Visual Builder, the external service and Identity Cloud Service (IDCS).

**Topics:**

- About Service Connection Authentication
- Connect to Fusion Applications APIs With End User Propagation for Authenticated Flows
- Connect to Fusion Applications APIs Not in the Catalog Using Fixed Credentials
- Connect to Oracle Integration APIs Using Identity Propagation
- Connect to Oracle Integration APIs Using Fixed Credentials
- Connect to Process APIs Using Identity Propagation
- Connect to Process APIs Using Fixed Credentials
- Connect to Content and Experience Cloud APIs Using Identity Propagation
• Connect to Content and Experience Cloud APIs Using Fixed Credentials
• Connect to Oracle Mobile Hub APIs Using Fixed Credentials
• Connect to ORDS APIs Using Fixed Credentials

About Service Connection Authentication

You configure how communication between your application and external services is authenticated in the Authentication tab of the service connection.

When configuring the connection details for a service, you can use the Authentication tab to do the following:

• Override authentication settings set at the tenant level
• Manage the credentials for accessing the service (if credentials are required)
• Manage identity propagation of the end user logged in to the web or mobile app (if the service supports the standard IDCS OAuth flows)
• Manage how your application connects to the service (via proxy or via Direct call)
• Manage how anonymous users can access the application

Note:

In some cases, an administrator might need to configure the Visual Builder tenant settings or the settings of the external service or identity provider before you can connect to the service.

To connect to a service that is available on HTTPS, does not require authentication, and has no special CORS requirement, the default settings in the Authentication tab are sufficient. By default, the “Enable authentication / proxy” checkbox isn’t selected, and the authentication options are not shown. In this case, any end user (anonymous or authenticated) of the web or mobile application will be able to access the service. Sending a request to a service that requires authentication will return an error if you do not provide the authentication details.

To connect to a service that requires authentication or that needs to be CORS-enabled, you need to select the “Enable authentication / proxy” checkbox in the Authentication tab and specify the authentication options and details for the service, including the authentication mechanism and if anonymous access is allowed.
You use the Authentication tab to configure the authentication details used for accessing the service during development and when the app is staged and published. Some mechanisms are better suited for app development, and in some cases you will want to change the authentication mechanism when you publish the app. You can select authentication mechanisms for logged in users (authenticated users) and non-logged in users (anonymous users). For a description of user roles, see About Authentication Roles and User Roles.

The mechanism you select for authenticating communication with a service will depend on the type of authentication the service supports and the authentication mechanism you use in your web or mobile app. Additionally, whether your external service supports CORS or not can also affect the choice of authentication. The authentication mechanisms can be grouped into two types, grouped by whether the identity of the end user is passed on to the service (Identity Propagation) or not (Fixed Credentials).

Identity Propagation Authentication Mechanisms

Authentication mechanisms that use identity propagation allow the identity of the end user who is signed in to the mobile or web app to be passed on to the service and used for authentication. These types of mechanisms cannot be used to authenticate anonymous access.

To use identity propagation, the service should be able to understand the IDCS identity token coming from Oracle Visual Builder and extract the user (or subject) from it. Oracle Visual Builder supports only JWT tokens procured using OAuth 2.0 flows.

Tokens are a way of encoding the calling user identity into a string according to different specifications like SAML or JWT format. For example, if the user is John.Doe, then the corresponding JWT token would take the format `<header.body.signature>` and would look like the following:
Decoding the body of the token yields details about the user identity and possibly the resources which he is authorized to access. The signature part is encrypted by the authority which authenticated the user, and can be easily verified by using the authority's public key. Thus, a valid user's identity is encoded into the token so that the services (namely REST APIs) which receive this token can consider the user as authenticated. The most common way of passing this token to REST services is to send it as a bearer token. i.e. pass "Bearer <token>" in the Authorization header.

The following table describes the authentication mechanisms that use the identity propagation type of mechanism.

<table>
<thead>
<tr>
<th>Authentication Mechanism – Identity Propagation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propagate Current User Identity</td>
<td>Select this to pass the identity of the user to the service (either as a token representing the logged in user or as the credentials used for signing in) and use it for authentication. It is recommended that you select this mechanism in the following cases: • Mobile apps that use Basic authentication • Web apps that use implicit OAuth For web apps, you must select this option if you want to use Implicit grant type for calls to external Fusion Applications services.</td>
</tr>
<tr>
<td>Oracle Cloud Account</td>
<td>Select this to communicate with Fusion Applications services or any co-hosted Oracle PaaS Service, for example, Oracle Integration Cloud. When this mechanism is selected, the user must sign in with the credentials of a valid account in the Oracle Identity Cloud Service that is associated with Oracle Visual Builder. If the “Enable Token Relay” option is selected, the service will be invoked directly from the web or the mobile app, and the CORS settings will need to be set on the service being called.</td>
</tr>
<tr>
<td>User Assertion OAuth 2.0</td>
<td>Select this to call services of external system which can be represented as a Resource app in Oracle Identity Cloud Service (IDCS). If the “Enable Token Relay” option is selected, the service will be invoked directly from the web or the mobile app, and the CORS settings will need to be set on the service being called.</td>
</tr>
</tbody>
</table>

Fixed Credentials Authentication Mechanisms

Authentication mechanisms that use fixed credentials pass a fixed identity to the service and ignore the identity or credentials of the end user who is signed in. All requests sent from the app to the service use the same app id for authentication.
The following table describes the authentication mechanisms that use the fixed credentials type of mechanism.

<table>
<thead>
<tr>
<th>Authentication Mechanism — Fixed Credentials</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct (Bypass Authentication Proxy)</td>
<td>Select this to call the service directly (see the Fetch API at <a href="https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API">https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API</a>) from the web or the mobile app. No authentication proxy is used, and the developer is responsible for providing all the necessary headers required by the service, including the Authorization header (if required). This option requires configuring the CORS settings on the service being called.</td>
</tr>
<tr>
<td>None</td>
<td>Select this for services that do not accept Authorization headers and no authentication is needed. This option uses the Oracle Visual Builder authentication proxy. This option is recommended for services where you are unable to set the CORS settings on the service.</td>
</tr>
<tr>
<td>Basic</td>
<td>Select this for services where a fixed username and password are required for authentication. The credentials of the signed in user are not used for authentication. This option uses the Oracle Visual Builder authentication proxy. This mechanism is only recommended during development.</td>
</tr>
<tr>
<td>Client Credentials OAuth 2.0</td>
<td>This mechanism is the recommended option if you want to use a Fixed Credentials mechanism and the service supports OAuth 2.0. This mechanism is part of the OAuth 2.0 grant types that are used when you don’t need a specific user’s credentials to connect to the service. Consult the service’s OAuth 2.0 documentation for the values for the Client Id, Client Secret and token URL fields. If no values are supplied, they will be interpreted as the visual application’s client id and secret, and the token URL will be interpreted as IDCS’s token URL. If the “Enable Token Relay” option is selected, the service will be invoked directly from the web or the mobile app, and the CORS settings will need to be set on the service being called.</td>
</tr>
</tbody>
</table>
Use the Oracle Visual Builder proxy to bypass CORS

To avoid CORS access issues when calling a service, you will want to add the Visual Builder domain to the service's whitelist of trusted domains. If you cannot control the service's CORS settings, you can use the Oracle Visual Builder proxy to avoid CORS access issues.

Very often the external services you are calling will not be in the same web domain as your Oracle Visual Builder application. Calling the REST APIs of these services directly (via fetch, or “old-style” XHR calls) from a web app or from the Oracle Visual Builder Services editor might result in Cross-Origin access issues when the request is not sent from a trusted domain.

The following options can be used to circumvent these types of issues:

- Add your Oracle Visual Builder domain to the service's whitelist. The whitelist is maintained by the external system hosting the service. Domains in the whitelist are recognized as trusted domains by the system hosting the service domain. Domains need to be added to the service's whitelist directly at the external system hosting the service you are calling.

  This method allows calls to go directly between the web or mobile app and the external service.

- Use the Oracle Visual Builder authentication proxy. When you use this option, every service request is sent via the Oracle Visual Builder proxy. The Oracle Visual Builder proxy is a trusted server-side component that can call service hosted on the external system on behalf of the web app.

  In Visual Builder, some authentication methods will require using the proxy, but some authentication methods will allow the proxy to be bypassed. Security best practices should be observed when using this option.

Use a token relay

When using the Oracle Cloud Account or OAuth authentication mechanisms, the app needs an access token to access the service.
Your app can get the token directly from the identity provider (via proxy, or using Implicit grant type for web apps), or your app can use the token relay mechanism where Oracle Visual Builder gets the token for your app and your calls to the service can then bypass getting token from the Oracle Visual Builder authentication proxy each time. You might choose to use the token relay option to improve performance when your app and the service it is calling are not located in the same identity domain.

If you choose to enable token relay for the authentication mechanism for a service, the Implicit grant type setting is ignored in the web app’s Security tab in the Settings editor. You will also need to identify your domain as a trusted domain by configuring the CORS setting at the service you are calling.

Work with HTTP-based Endpoints

Oracle Visual Builder uses HTTPS for its applications, and we highly recommend also using HTTPS for all Service Connections to external endpoints.

For development purposes only, in order to build a Service Connection that uses HTTP, you will need to use the Visual Builder proxy to forward your request. The way to accomplish this is to use an appropriate mechanism, depending on what the Service accepts. The following mechanisms use the Visual Builder proxy to forward requests to the actual endpoint.

- None
- Basic
- Oracle Cloud Account
- OAuth 2.0 Client Credentials
- OAuth 2.0 Resource Owner Password
- OAuth 2.0 User Assertion

You shouldn’t select Token relay (where applicable).

Allow anonymous access to the service data

You can select Allow anonymous access in the Authentication tab if you want to allow anonymous users access to the data from services.

If you allow anonymous access to the service data, you must also allow anonymous users access to the app. You enable anonymous access to the app in the Security tab of the app’s Settings editor. See Allow Anonymous Access.

Note:

REST services that permit anonymous access can be accessed from a mobile application that uses Basic as its authentication mechanism if the REST service connection in the mobile application is configured to use Direct (Bypass Authentication Proxy) as the authentication mechanism.
Connect to Fusion Applications APIs With End User Propagation for Authenticated Flows

To connect to Fusion Applications APIs and use end user propagation, the identity provider used by Fusion Applications and the IDCS used by Oracle Visual Builder need to be federated, and two applications representing a Fusion Applications resource need to be created in IDCS.

Fusion Applications use their own identity provider (IdP), and Oracle Visual Builder uses only IDCS as its identity provider. The identity providers used by each service need to be federated to establish trust between them. The list of users can be maintained in either identity provider, but not both, and it is recommended that you use the Fusion Applications IdP. The Fusion Applications IdP can be further integrated via SAML with other identity providers.

Before configuring the connection, an administrator needs to work with Oracle support to federate IDCS and Fusion Applications. This process will result in creating an application in IDCS that represents the Fusion Applications resource. The process is described in Enable Oracle Fusion Applications Cloud Service Federation and OAuth Trust with Oracle Identity Cloud Service.

An administrator will then need to perform the following steps:

1. In IDCS, create a second application representing the Fusion Applications resource that is identical to the first, except that ":443" is appended to the URL for the primary audience. For example, if the Primary Audience URL for the first IDCS application is https://<your identity provider REST API host name>, the Primary Audience URL for the second IDCS application will be https://<your identity provider REST API host name>:443.

2. In the Visual Builder Tenant Settings, enter the Fusion Applications Base URL, if empty. This URL is the location of the API of the interface catalog for the Fusion Application defined in IDCS, and should be similar to https://<your identity provider REST API host name>/helpPortalApi/otherResources/latest/interfaceCatalogs.

When creating the service connection, you use the following authentication mechanism:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Cloud Account</td>
<td>The default authentication for the connection is determined by the authentication set in the Settings editor and in the Tenant Settings. You can view the connection details in the Settings editor. You can choose to override the settings to select the Authentication Mechanism and enable the following options.</td>
</tr>
<tr>
<td></td>
<td>• Enable Token Relay. Token relay should be enabled to allow Direct flows, or disabled if you want to use proxy-based flows. • Allow anonymous access. Select this to enable anonymous access and to select one of the authentication methods that support fixed credential authentication</td>
</tr>
</tbody>
</table>

To connect to Fusion Applications APIs with end user propagation:
1. Open Service Connections in the Navigator and click **Create Service Connection** (+).

2. Click **Select from Catalog** in the Create Service Connection wizard.

3. Click **FA Services**, and select a service and endpoint. Click **Create**.

4. Test the service connection.

5. Optional: Select "Allow anonymous access" if you want to make the service connection accessible to anonymous user of the app.

   If you select "Allow anonymous access", you can select "Basic" and supply the credentials, or "Same as authenticated user" as the authentication method.

---

**Connect to Fusion Applications APIs Not in the Catalog Using Fixed Credentials**

To connect to a service in a Fusion Application instance that is not associated with your Visual Builder instance, you can create a connection using the credentials of a fixed user registered in the Fusion Applications instance.

To create the connection, you need to have the credentials of a fixed Fusion Applications user of the instance, or an administrator will need to create the Fusion Applications user with the necessary privileges for you. When you use this user to create the connection, all requests to the Fusion Applications REST APIs will use the fixed user's credentials. The credentials of the logged in end user are not used when communicating with the service.

Fusion Applications are usually associated with your Visual Builder instance in the Tenant Settings. If your instance is already associated with a Fusion Applications instance and you want to connect to a different instance in your visual application using the Service Catalog, you can open your visual application's Settings editor and override the Tenant Settings to edit the application's Fusion Applications instance settings. When you override the default instance settings, your application will not be able to access any services provided by the default instance because a visual application can only have one default Fusion Applications instance that is used to populate the Service Catalog.

If you do not want to change the default Fusion Applications instance for your application, you can create a service connection by selecting the Define by Endpoint option in the Create Service Connection wizard.

When creating the service connection, you can use the following authentication mechanism for the service connection:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
</table>
| Basic                    | To use this option you need to provide the following details:  
  • Username and Password. These can be the valid credentials of any user that has access to the Fusion Applications REST APIs. |

To connect to a Fusion Applications service that is not in your catalog:

1. Open the **Services** tab of the visual application's Settings editor.
2. Select **FA Services** in the Services tab.

3. Enable Override Tenant settings.

4. Provide the Catalog Base URL of the Fusion Applications instance.
   The format of the URL should be similar to `https://<FA instance>/helpPortalApi/otherResources/latest/interfaceCatalogs`.

5. Select **Basic** as the Authentication Mechanism and provide the Username and Password of the fixed user.

6. Open Service Connections in the Navigator and click **Create Service Connection** (+).

7. Click **Select from Catalog** in the Create Service Connection wizard.
   If you did not change the default Fusion Applications instance of your application, you can choose Define by Endpoint and provide the URL of the endpoint.

8. Click **FA Service** and select a service and endpoint. Click **Create**.

9. Confirm the connection is working.

10. Optional: Select “Allow anonymous access” if you want to make the service connection accessible to anonymous user of the app.
    If you select "Allow anonymous access", you can select "Basic" and supply the credentials, or "Same as authenticated user" as the authentication method.

### Connect to Oracle Integration APIs Using Identity Propagation

To connect to Oracle Integration using identity propagation, the Oracle Integration and Oracle Visual Builder instances should be in the same domain. If your Oracle Visual Builder was provisioned with Oracle Integration, both services are accessible from the Oracle Integration home page and menu.

An administrator will need to confirm that the Oracle Integration URL is correct in the Oracle Visual Builder Tenant Settings. The Oracle Integration URL in the Tenant Settings should be similar to `https://<Integration Cloud Instance full URL>:443` and the authentication mechanism should be “Oracle Cloud Account”. This will appear as the default URL and authentication mechanism in the visual application's Settings editor. Using “Oracle Cloud Account” authentication provides identity propagation from Oracle Visual Builder to Oracle Integration without the need for any additional configuration.

When creating the service connection, you use the following authentication mechanism:
Authentication mechanism | Details
--- | ---
Oracle Cloud Account | The default authentication for the connection is determined by the authentication set in the Settings editor and in the Tenant Settings. You can view the connection details in the Settings editor. You can choose to override the settings to select the Authentication Mechanism and enable the following options.
  - Enable Token Relay. Token relay should be enabled to allow Direct flows, or disabled if you want to use proxy-based flows.
  - Allow anonymous access. Select this to enable anonymous access and to select one of the authentication methods that support fixed credential authentication.

To connect to Oracle Integration APIs using identity propagation:

1. Open the **Services** tab of the visual application’s Settings editor.
2. Select **Integrations** in the Services tab and confirm that “Oracle Cloud Account” is selected as the default Authentication Mechanism and that Enable Token Relay is selected.

   You can override the URL coming from Tenant settings, but keep in mind that the identity propagation will only happen for the co-located Integration instance. No additional CORS configuration is needed when Oracle Visual Builder and Oracle Integration are in the same domain.

3. Open Service Connections in the Navigator and click **Create Service Connection** (`+`).
4. Click **Select from Catalog** in the Create Service Connection wizard.

   Alternatively, you can click “Define by endpoint” and provide the URL of the sample Integration and select “Oracle Cloud Account” as the Authentication Mechanism, and enable or disable Token Relay according to your preference.
5. Select the sample integration endpoint from the list of Catalog endpoints. Click **Create**.
6. Test the Service Connection.
7. Optional: You can also make the service connection accessible to anonymous users by enabling **Allow anonymous access** and selecting one of the authentication methods that support fixed credential authentication.

**Connect to Oracle Integration APIs Using Fixed Credentials**

To connect to Oracle Integration APIs using fixed credentials, you can choose to use either Basic Auth or OAuth 2.0 Resource Owner Password as the authentication mechanism.

To access the Oracle Integration APIs using fixed credentials, the Oracle Integration and Oracle Visual Builder instances do not need to be located in the same domain or governed by the same IDCS instance. Configuration is the same in both cases.

When you create the service connection in the Create Service Connection wizard, you choose the service by either selecting it in the Service Catalog or by defining its endpoint or specification. If you want to select a service from the catalog, you will first
need to open the Services tab of the visual application's Settings editor and override the tenant-level settings for Integrations and select the authentication mechanism you want to use instead of the default Oracle Cloud Account mechanism.

You do not need to override the tenant-level settings if you are defining the service connection by endpoint or specification in the Create Service Connection wizard. The authentication mechanisms are the same in both cases.

If you want to use OAuth 2.0 Resource Owner Password as the authentication mechanism, a service administrator needs to perform the following steps in the IDCS instance governing the Oracle Integration instance to get its Client ID, Client Secret, and Scope. These details are not needed when using Basic authentication.

1. Open Applications in IDCS and locate the Oracle Integration application which frontends the Integration instance.
2. Open the application and copy the Client ID and Client Secret in the General Information panel of the Configuration tab.
3. Expand the Resources panel of the Configuration tab and copy the Primary Audience and the scope that corresponds to the REST APIs. These are combined to give the full scope, and might be similar to `https://<primary-audience-unique-id>.integration.ocp.oraclecloud.com:443/ic/api`.

When creating the service connection, you can use either of the following authentication mechanisms:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
</table>
| Basic                    | To use this option you need to provide the following details:  
• Username and Password. These can be the valid credentials of any user that has access to the Integration REST endpoint. |
| OAuth 2.0 Resource Owner Password | To use this option you need to provide the following details:  
• Client Id and Secret. This is from the IDCS of Oracle Integration  
• Username and Password. These can be the valid credentials of any user that has access to the Integration REST endpoint.  
• Token URL. The URL will be similar to `https://<base url of IDCS of Integration Cloud>/oauth2/v1/token`  
• Scope. This is from the IDCS of Oracle Integration  
• Token Relay checkbox. This can enabled for Direct flows, or disabled for Proxy-based flows. If you choose to use Direct flows, CORS needs to be configured at the Integration Cloud REST endpoint definition to add the Visual Builder domain to the list of allowed origins. |

If you do not have access to IDCS, you will need to request the connection details from an administrator if you want to use the OAuth 2.0 Resource Owner Password authentication mechanism.

To connect to Oracle Integration APIs using fixed credentials:

1. Open Service Connections in the Navigator and click **Create Service Connection**.
2. Select the source in the Create Service Connection wizard.
   
   You can choose **Select from Catalog** if the Integrations service you want to access is in your Service Catalog and you have overridden the tenant-level
settings in the application’s Settings editor. If it is not in your Service Catalog, choose **Define by Specification** or **Define by Endpoint**.

3. Step through the wizard to define the service connection.

4. Select one of the supported authentication mechanisms and provide the authentication details.

   If you chose a service from your Service Catalog, you can override the default authentication settings in the service’s Authentication tab.

5. Test the service connection.

6. Optional: You can also make the service connection accessible to anonymous users by enabling **Allow anonymous access** and selecting one of the authentication methods that support fixed credential authentication.

## Connect to Process APIs Using Identity Propagation

To connect to Process APIs using identity propagation, the Process and Oracle Visual Builder instances must be located in the same domain and use the same IDCS instance for authentication.

Using "Oracle Cloud Account" authentication provides identity propagation from Oracle Visual Builder to Process without the need for any additional configuration. No CORS configuration is required because the instances are in the same domain. When creating the service connection, you use the following authentication mechanism:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Cloud Account</td>
<td>You can choose to override the settings to select the Authentication Mechanism and enable the following options.</td>
</tr>
<tr>
<td></td>
<td>• Enable Token Relay. Token relay should be enabled to allow Direct flows, or disabled if you want to use proxy-based flows.</td>
</tr>
<tr>
<td></td>
<td>• Allow anonymous access. Select this to enable anonymous access and to select one of the authentication methods that support fixed credential authentication or by using &quot;Same as authenticated user&quot;.</td>
</tr>
</tbody>
</table>

To connect to Process APIs using identity propagation:

1. Open Service Connections in the Navigator and click **Create Service Connection** (+).
2. Click **Define by Endpoint** in the Create Service Connection wizard.
3. Provide the full applicable URL.
   
   For example, the URL might be similar to `https://<process cloud instance>/bpm/api/4.0/processes` to retrieve a list of processes.
4. Select **Oracle Cloud Account** as the authentication mechanisms.
5. Optional: Select Token Relay to use a Direct (non-proxy) flow.
6. Test the service connection.
7. Optional: You can also make the service connection accessible to anonymous users by enabling **Allow anonymous access** and selecting one of the authentication methods.
## Connect to Process APIs Using Fixed Credentials

To connect to Process APIs using fixed credentials, you can choose to use Basic Auth, OAuth 2.0 Client Credentials or OAuth 2.0 Resource Owner Password as the authentication mechanism.

For access to Process APIs using fixed credentials, the Process and Oracle Visual Builder instances do not need not be located in the same domain or governed by the same IDCS instance. The steps are the same if they use the same IDCS instance. When creating the service connection, you can use one of the following authentication mechanism for the service connection:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>To use this option you need to provide the following details:</td>
</tr>
<tr>
<td></td>
<td>• Username and Password. These can be the valid credentials of any user that has access to the Process API.</td>
</tr>
<tr>
<td>OAuth 2.0 Client Credentials</td>
<td>To use this option you need to provide the following details:</td>
</tr>
<tr>
<td></td>
<td>• Client Id and Secret. This is from the IDCS of Process</td>
</tr>
<tr>
<td></td>
<td>• Token URL. The instance token URL from IDCS.</td>
</tr>
<tr>
<td></td>
<td>• Scope. This is from the IDCS of Process.</td>
</tr>
<tr>
<td></td>
<td>• Token Relay checkbox. This can enabled for Direct flows, or disabled for Proxy-based flows. If you choose to use Direct flows, CORS needs to be configured at the Process REST endpoint definition to add the Visual Builder domain to the list of allowed origins.</td>
</tr>
<tr>
<td>OAuth 2.0 Resource Owner Password</td>
<td>To use this option you need to provide the following details:</td>
</tr>
<tr>
<td></td>
<td>• Client Id and Secret. This is from the IDCS of Process</td>
</tr>
<tr>
<td></td>
<td>• Username and Password. These can be the valid credentials of any user that has access to the Process REST endpoint.</td>
</tr>
<tr>
<td></td>
<td>• Token URL. The instance token URL from IDCS.</td>
</tr>
<tr>
<td></td>
<td>• Scope. This is from the IDCS of Process.</td>
</tr>
<tr>
<td></td>
<td>• Token Relay checkbox. This can enabled for Direct flows, or disabled for Proxy-based flows. If you choose to use Direct flows, CORS needs to be configured at the Process REST endpoint definition to add the Visual Builder domain to the list of allowed origins.</td>
</tr>
</tbody>
</table>

If you do not have access to IDCS, you will need to request the authentication details from an administrator if you want to use the OAuth 2.0 Client Credentials or OAuth 2.0 Resource Owner Password authentication mechanisms.

To connect to Process APIs using fixed credentials:

1. Open Service Connections in the Navigator and click **Create Service Connection** (➕).
2. Click **Define by Endpoint** in the Create Service Connection wizard.
3. Provide the full applicable URL.
   - For example, the URL might be similar to `https://<process cloud instance>/bpm/api/4.0/processes` to retrieve a list of processes.
4. Select one of the supported authentication mechanisms and provide the authentication details.

5. Test the service connection.

6. Optional: You can also make the service connection accessible to anonymous users by enabling **Allow anonymous access** and selecting one of the authentication methods that support fixed credential authentication or by using "Same as authenticated user".

**Connect to Content and Experience Cloud APIs Using Identity Propagation**

To connect to Content and Experience Cloud (CECS) using identity propagation, an administrator will need to configure your Visual Builder application in IDCS before you can use the default "Oracle Cloud Account" to connect to CECS in your application.

Content and Experience Cloud and Oracle Visual Builder are not provisioned together, and as a result the service administrator needs to perform the following steps in IDCS to add CECS as a resource of the Visual Builder application. This adds CECS as a resource to a specific application, so the administrator would need to perform these steps again for each new Visual Builder application, as well as for each new version of an application and duplicate of an application that connects to CECS using identity propagation.

1. In the Configuration tab for the Visual Builder application in IDCS, expand the Client Configuration panel and click [Add Scope](#) in the Token Issuance Policy section.

2. In the Select Scope dialog box, choose the scope corresponding to the CECS instance "/documents" endpoint and save the application. The added scope should now be visible in the Application in the Resources list.

If other CECS functionality (for example, Social) is required, the corresponding scope will need to be added.

After the administrator has added the resource in IDCS, you can create a connection to CECS with identity propagation. If you don't have access to IDCS, the administrator will need to provide you with the CECS Scope that you need to enter in the Authentication tab.

When creating the service connection, you use the following authentication mechanism for the service connection:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
</table>
| OAuth 2.0 User Assertion | To use this option you need to provide the following details:  
|                         | • Client Id and Secret. This is blank.  
|                         | • Token URL. This is blank.  
|                         | • Scope. This the scope added from IDCS corresponding to the CECS instance. This is the full scope, including "/documents".  
|                         | • Token Relay checkbox. This can enabled for Direct flows, or disabled for Proxy-based flows. If you choose to use Direct flows, CORS needs to be configured at the CECS REST endpoint definition to add the Visual Builder domain to the list of allowed origins. |
To connect to Content and Experience Cloud:

1. Open Service Connections in the Navigator and click **Create Service Connection**.
2. Click **Define by Endpoint** in the Select Source pane of the Create Service Connection wizard.
3. Select the HTTP method and type the URL of the endpoint in CECS.
   For example, the URL of your endpoint might be similar to the following: `https://<CES_INSTANCE>/documents/api/<VERSION>/folders/{folderId}`
4. In the Authentication tab, select **OAuth 2.0 User Assertion** as the Authentication Mechanism.
5. In the Scope field, enter the scope corresponding to the CECS instance that was added in IDCS.
   The Client Id, Secret and Token URL fields are blank.
6. Optional: Select Token Relay if you are using a Direct (non-Proxy) flow.
   To use a Direct flow, the CECS administrator needs to add the Visual Builder domain to the list of Front Channel CORS Origins in the Security panel of the CECS Settings page.
7. Test the service connection.
8. Select "Allow anonymous access" if you want to make the Service Connection accessible to anonymous user of the app.
   If you select "Allow anonymous access", you can supply the allowed set of Fixed Credential authentication methods or by using "Same as authenticated user".

**Connect to Content and Experience Cloud APIs Using Fixed Credentials**

To connect to Content and Experience Cloud (CECS) using fixed credentials, the CECS and Oracle Visual Builder instances do not need to be located in the same domain or governed by the same IDCS instance. You can use Basic or OAuth 2.0 Resource Owner Password authentication for the service connection.

If you want to use Basic authentication for the connection to CECS, you need to provide a username and password that are valid in IDCS.

If you want to use OAuth flows for authenticating your connection to CECS, you need to retrieve details about the Client Secret, Client Id and the URL associated with the scope that you want to access. Typically, the scope you will want to access will be "/documents", but if you want to access other CECS functionality (for example, Social), you will need the URL that corresponds to its scope. If you do not have access to the IDCS instance used by CECS, you will need to request the details from a user with access to the instance.

To retrieve the details of the CECS application from IDCS:

1. In the Configuration tab for the CECS application in IDCS that represents the CECS instance, expand the General Information panel and note the Client Id and Client Secret.
The name of the CECS application will usually be similar to CECSXXX_<instance name>.

2. Expand the Resources panel in the Configuration tab and note the URL for the scope you want, typically the scope corresponding to the CECS instance "/documents" endpoint. The URL will be similar to https://<primary audience url>/documents.

If other CECS functionality (for example, Social) is required, you will need to note the URL for the corresponding scope.

When creating the service connection, you can use one of the following authentication mechanisms for the service connection:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
</table>
| Basic                    | To use this option you need to provide the following details:  
  • Username and Password. These can be the valid credentials of any user from IDCS. |
| OAuth 2.0 Resource Owner Password | To use this option you need to provide the following details:  
  • Client Id and Secret. You get these from the CECS application in IDCS.  
  • Username and Password. These can be the valid credentials of any user that has access to the CECS REST endpoint.  
  • Token URL. The URL for the endpoint used to obtain an access token from IDCS. The form of the URL will be similar to <base URL corresponding to CECS in IDCS>/oauth2/v1/token.  
  • Scope. You get this from the CECS application in IDCS.  
  • Token Relay checkbox. This can enabled for Direct flows, or disabled for Proxy-based flows. If you choose to use Direct flows, CORS needs to be configured at the CECS REST endpoint definition to add the Visual Builder domain to the list of allowed origins. |

If you do not have access to IDCS you will need to request the CECS application details from an administrator.

To connect to CECS using fixed credentials:

1. Open Service Connections in the Navigator and click Create Service Connection (+).
2. Click Define by Endpoint in the Select Source pane of the Create Service Connection wizard.
3. Select the HTTP method and type the URL of the endpoint in CECS.
   For example, the URL of your endpoint might be similar to the following: https://<CECS_INSTANCE>/documents/api/<VERSION>/folders/{folderId}
4. In the Authentication tab, select OAuth 2.0 Resource Owner Credentials as the Authentication Mechanism.
5. Enter the details for the Client Id, Client Secret, Scope and Token URL. The Client Id, Client Secret and Scope details are the ones that you noted for the CECS application in IDCS.
6. Optional: Select Token Relay if you are using a Direct (non-Proxy) flow.
To use a Direct flow, the CECS administrator needs to add the Visual Builder domain to the list of Front Channel CORS Origins in the Security panel of the CECS Settings page.

7. Test the service connection.

8. Select “Allow anonymous access” if you want to make the Service Connection accessible to anonymous user of the app.

If you select “Allow anonymous access”, you can supply the allowed set of Fixed Credential authentication methods or by using “Same as authenticated user”.

Connect to Oracle Mobile Hub APIs Using Fixed Credentials

To connect to Oracle Mobile Hub using fixed credentials, you can choose to use Basic Auth, OAuth 2.0 Client Credentials or OAuth 2.0 Resource Owner Password as the authentication mechanism.

To access Oracle Mobile Hub from Oracle Visual Builder, the settings of the container for the mobile APIs need to be configured to allow connections from your visual application using one of the authentication mechanisms. To use Basic Auth, the mobile backend must have “HTTP Basic” enabled. To use OAuth 2.0, the “OAuth Consumer” option must be enabled.

An administrator will need to perform the following steps to configure the mobile backend:

1. Open the settings for the Oracle Mobile Hub mobile backend.
   This backend represents the container for the various custom/platform APIs that you will connect to using the service connection.

2. Enable the connection for the type of authentication you want to use, and note the details for the connection.
   If the connection will use Basic Auth, confirm that the Mobile backend has “HTTP Basic” enabled. If the connection will use OAuth 2.0, confirm that “OAuth Consumer” is enabled. Depending on which you choose, you might need the value of the Anonymous Key, Mobile Backend ID, Client ID and Client Secret.

If Direct flows are needed, then the CORS need to be added to Security_AllowOrigin policy at the OMH backend side.

When creating the service connection, you can use one of the following authentication mechanisms:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
</table>
| Basic                    | To use this option you need to provide the following details:  
  • Username and Password. These can be the valid credentials of any user that has access to the Oracle Mobile Hub API.  
  • Static header. You need to add a static header named oracle-mobile-backend-id in the Headers tab of your service connection. The value of the header needs to be set to the Mobile Backend ID from Oracle Mobile Hub. |
<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
</table>
| None                     | To use this option you need to edit the service connection to add the following:  
  • Secure static header. You need to add a static header named `Authorization` in the Headers tab of your service connection. The value of the header needs to be set to `Basic <Anonymous key>`, where `<Anonymous key>` is the value from the Oracle Mobile Hub backend.  
  • Static header. You need to add a static header named `oracle-mobile-backend-id` in the Headers tab of your service connection. The value of the header needs to be set to the Mobile Backend ID from Oracle Mobile Hub. |

| OAuth 2.0 Client Credentials | To use this option you need to provide the following details:  
  • Client Id and Secret. This is from the Oracle Mobile Hub backend.  
  • Token URL. This is the OAuth Token endpoint from the Oracle Mobile Hub backend.  
  • Scope. See Authenticate with OAuth in Direct REST Calls in Developing Applications for Oracle Mobile Hub. The format will be similar to `https://<baseURL>urn:opc:resource:consumer::all`. For example, if the baseURL given in the Oracle Mobile Hub settings is `https://abcdef.mobile.ocp.oraclecloud.com:443`, the scope would be `https://abcdef.mobile.ocp.oraclecloud.com:443urn:opc:resource:consumer::all`.  
  • Token Relay checkbox. This can enabled for Direct flows, or disabled for Proxy-based flows. If you choose to use Direct flows, CORS needs to be configured at Oracle Mobile Hub to add the Visual Builder domain to the list of allowed origins. |
Authentication mechanism | Details
--- | ---
OAuth 2.0 Resource Owner Password | To use this option you need to provide the following details:
- Username and Password. These can be the valid credentials of any user that has access to the Oracle Mobile Hub API.
- Client Id and Secret. This is from the Oracle Mobile Hub backend.
- Token URL. This is the OAuth Token endpoint from the Oracle Mobile Hub backend.
- Scope. See Authenticate with OAuth in Direct REST Calls in Developing Applications for Oracle Mobile Hub. The format will be similar to https://<baseURL>urn:opc:resource:consumer::all. For example, if the baseURL given in the Oracle Mobile Hub settings is https://abcdef.mobile.ocp.oraclecloud.com:443, the scope would be https://abcdef.mobile.ocp.oraclecloud.com:443urn:opc:resource:consumer::all.
- Token Relay checkbox. This can enabled for Direct flows, or disabled for Proxy-based flows. If you choose to use Direct flows, CORS needs to be configured at Oracle Mobile Hub to add the Visual Builder domain to the list of allowed origins.

If you do not have access to the Oracle Mobile Hub mobile backend, you will need to request the connection details from an administrator.

To connect to Oracle Mobile Hub using fixed credentials:

1. Open Service Connections in the Navigator and click Create Service Connection (➕).
2. Click Define by Endpoint in the Select Source pane of the Create Service Connection wizard.
3. Select the HTTP method and type the URL of the endpoint in Oracle Mobile Hub.
4. In the Authentication tab, select the Authentication Mechanism you want to use.
5. Enter the required details based on the Authentication Mechanism you are using.
6. Optional: Select Token Relay if you are using the OAuth 2.0 Resource Owner Password mechanism and want to use a Direct (non-Proxy) flow.

   To use a Direct flow, the administrator needs to add CORS to the Security_AllowOrigin policy on the Oracle Mobile Hub backend.
7. Test the service connection.
8. Select "Allow anonymous access" if you want to make the Service Connection accessible to anonymous user of the app.

   If you select "Allow anonymous access", you can supply the allowed set of Fixed Credential authentication methods or by using "Same as authenticated user".
Connect to ORDS APIs Using Fixed Credentials

To connect to Oracle REST Data Services (ORDS) using fixed credentials, you can use OAuth 2.0 Client Credentials for authentication. Oracle Visual Builder

Before creating a connection to ORDS, a role and privilege to protect your REST service need to be created and the OAuth client needs to be registered in the ORDS service. The following steps briefly describe this process. See Protecting and Accessing Resources.

When creating the service connection, you can use the following authentication mechanism for the service connection:

<table>
<thead>
<tr>
<th>Authentication mechanism</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth 2.0 Client Credentials</td>
<td>This is the recommended authentication option. To use this option you need to provide the following details:</td>
</tr>
<tr>
<td></td>
<td>• Client Id and Secret. From ORDS</td>
</tr>
<tr>
<td></td>
<td>• Token URL. From ORDS, for example, <a href="https://example.com/ords/ordstest/oauth/token">https://example.com/ords/ordstest/oauth/token</a></td>
</tr>
<tr>
<td></td>
<td>• Scope. This is blank.</td>
</tr>
<tr>
<td></td>
<td>• Token Relay checkbox. This can enabled for Direct flows, or disabled for Proxy-based flows. If you choose to use Direct flows, CORS needs to be configured at the ORDS REST endpoint definition to add the Visual Builder domain to the list of allowed origins.</td>
</tr>
</tbody>
</table>

1. Create a role and privilege to protect your REST service in ORDS

   begin ords.create_role('HR Administrator');
   ords.create_privilege(
     p_name => 'example.employees',
     p_role_name => 'HR Administrator',
     p_label => 'Employee Data',
     p_description => 'Provide access to employee HR data');
   commit;end;

2. Associate the privilege with resources (i.e. your ORDS REST APIs)

   begin ords.create_privilege_mapping(
     p_privilege_name => 'example.employees',
     p_pattern => '/examples/employees/*');
   commit;end;

Accessing the /example/employees REST resource should now result in a 401 unauthorized as below

   curl -i https://example.com/ords/ordstest/examples/employees/
HTTP/1.1 401 Unauthorized
Content-Type: text/html
Transfer-Encoding: chunked
3. Register the OAuth client with grant type Client Credentials

begin oauth.create_client(
    p_name => 'Client Credentials Example',
    p_grant_type => 'client_credentials',
    p_privilege_names => 'example.employees',
    p_support_email => 'support@example.com');
commit;end;

Check the registered client id and secret

select client_id,client_secret from user_ords_clients where name = 'Client Credentials Example';

To create a connection to ORDS using fixed credentials:

1. Open Service Connections in the Navigator and click Create Service Connection ( ).
2. Click Define by Endpoint in the Select Source pane of the Create Service Connection wizard.
3. Select the HTTP method and type the URL of the endpoint in ORDS.
4. In the Authentication tab, select OAuth 2.0 Client Credentials as the Authentication Mechanism.
5. Provide the details for the Client Id, Secret and Token URL fields based on your ORDS configuration.
6. Test the service connection.
You can use business processes in your application to automate assigning tasks to users and creating task lists. After registering your business processes in your visual application, you can add UI components to your pages that start processes and that can be use to complete user tasks.

**Topics**

- About Using Processes in Your Application
- Add a Business Process to an Application
- Modify a Business Process Alias
- Navigate to Your Process Instance
- Add a Process to a Page
- Add a Process to an Action Chain
- Start and Complete Processes from Oracle Visual Builder
- Configure the Connection to a Process Server
- Process Server Authentication Options
- Process Server Instance Options

---

**Note:**

To work with a business process, you need to provision the Enterprise edition of Oracle Integration, which includes the Processes and Oracle Visual Builder features. See Ready, Set Up, and Go in *Administering Oracle Integration*.

---

**About Using Processes in Your Application**

You can create business processes to help automate tasks that need to be performed by application users. A process might include tasks such as reviewing data, approving or rejecting requests, and submitting files.

For example, completing a service or travel request is a process that might have several steps and involve several people. You can create a business process that is automatically initiated when a user submits a travel request in the application. Your process can generate the user tasks, such as getting approvals and submitting documents, and assign them to the appropriate users when they need to be performed. Logged in users might see a list of tasks they need to do or the status of their requests. Employees might see a list of requests that they submitted, and managers might see the requests that they need to approve or reject.
After you create a process in a Process in Oracle Integration application, you can register the process in your visual application and add code to pages in your web or mobile applications to initiate the process. You can also add process actions to action chains that can be triggered by events in your application.

Note:

You need to configure the Process Server before you can add a business process.

Add a Business Process to an Application

When adding a business process to an application you can use business process aliases that are registered on the Process Server.

When you use an alias, the code that you add to the page refers to the alias rather than to a specific version of the process, and you can update the process version without changing the code by updating the alias in the Process Alias editor.

To add a business process to an application:

1. Open Processes in the Navigator and click to open the Register Deployed Process dialog box.

![Register Deployed Process dialog box]

2. Select the process you want to add.
3. Type the alias you want to use for the process. Click Add.
4. Repeat steps 2 and 3 to add aliases for more business processes. Click **Close** when you are finished adding aliases.

The aliases for each of the business processes that you added to your application are listed in the Processes pane. You can select an alias to open it in the editor.

### Modify a Business Process Alias

A business process alias points to a revision of a process in a specific Process application registered in a test instance or a production instance.

You can use the process editor to modify the target of your process alias to point to a different process and to automatically point to a default or more recent revision instead of a specific revision.

When you are developing your application you will want your alias to point to a process on a test instance. When you are ready to stage or publish your application you will need to modify the alias to point to a process on a production instance.

To update a process alias:

1. Open the alias in the Process Alias editor.

   The Process Alias editor contains read-only fields that identify the name of the alias, the Process Application and the Process Name that the alias points to. The editor also contains a Code Snippets section with the code that you will copy into pages in your application.
2. Click the ellipsis (…) next to the Registered Deployed Process field to select a new process in the Registered Deployed Process dialog box.

The read-only text field displays the process that the alias points to. You need to click the ellipsis button to select a different target.

If you select the Static option when the alias points to a process on a test instance, you will see the following warning: A test instance has been selected. When the application is staged or published, another deployed process will need to be selected. The default is Dynamic.

Navigate to Your Process Instance

If you have a configured connection to a Process Server, you can use the Processes menu to navigate directly to your Process instance.

Click the menu and select Design Processes to navigate to the Process Applications tab of Process in Oracle Integration, where you can create, edit, publish, and test processes.

Click the menu and select Manage Processes to navigate to the My Tasks tab of Process in Oracle Integration, where you can complete assigned tasks.

Click the menu and select Refresh Processes to refresh the process alias metadata for your processes.
Add a Process to a Page

After you have created an alias for a business process, you can add the code for the process by copying the process's HTML and JSON code into the page's source code in the editor. The Code Snippets panel in the Process Alias editor contains a description and code used for each of the process's available operations.

To use code snippets to add a process to a page:

1. Open the process you want to add in the Process Alias editor.
2. In the Code Snippets panel, select the operation you want to add.

When you select an operation, the Info tab contains details about the operation, including descriptions of objects, and the properties, parameters and outcomes of actions.
3. In the HTML tab of the Code Snippets panel, copy the HTML source for the operation.

4. Open the page where you want to add the process in the Designer.

5. Click Code in the toolbar to open the page’s HTML code in the editor.

   When you are in the Designer, you use the Design and Code buttons in the toolbar to switch between the page’s layout editor and the HTML editor.
6. Paste the HTML code you copied into the page’s HTML editor.
   When you paste the HTML code into the page you might see errors in the editor.
   The errors should be resolved after you add the JSON metadata.

7. In the Process Alias editor, copy the content in the alias’s JSON tab.

8. In the application designer, open the page’s Metadata editor to edit the page’s JSON file.
   Each page uses a JSON file to describe the page’s metadata, such as the variables and action chains that are defined for the page.
9. Paste the JSON metadata for the process into the correct location in the page's JSON file.

   The JSON metadata for a process might contain definitions for various artifacts, including action chains, variables and imports. You need to edit the page's JSON to add the process's metadata in the correct location. The editor will show warnings and hints in the margins to indicate where there are possible problems with the syntax.

   Code snippets are not the only way to add a process to a page. You can also display processes and tasks in tables and lists, and you can use the Add Task Actions quick start for a table or list to allow users to complete tasks.

**Add a Process to an Action Chain**

When you are building an action chain in the editor, you can add business process actions to the chain.

You can add process actions to an action chain by dragging the actions from the Process category in the Action Chain editor. The actions correspond to the operations described in the Code Snippets panel of the Process Alias editor. The Info tab in the Code Snippets panel contains a description of each action and its properties and parameters. A business process might have multiple interfaces. In the editor, when selecting a process, you will need to select one of the process's interfaces, and the interface you select will determine the action parameters that you will need to specify in the Property Inspector. See [Create an Action Chain](#).

To add a process operation to an action chain:

1. Open the action chain in the Action Chain editor. Select the component on the canvas.

   Alternatively, you can create a new action chain that is triggered by a component event and add process actions to the chain.

2. Locate the Process category in the Action Chain editor.

3. Drag the action you want to add onto the Add icon ( + ) in the action chain where you want to add the action.

The following steps apply if you select the Start Process action.


5. Select the process interface for the business process in the Select Process dialog box. Click Select.

In the Select Process dialog box you can choose a process alias that you have created or a process definition available on the Process server. The list of process definitions on the server is based on the options you selected in the Process Configuration editor.

The business process that you select may have one or more process interfaces. You expand the process node in the Select Process dialog box to see the available interfaces.
6. Specify the interface’s required properties and parameters in the Property Inspector.

Depending on the process interface for the process, you might need to map the interface’s required input parameters to variables. If so, the interface displays a warning icon.
Start and Complete Processes from Oracle Visual Builder

To be able to start a process from Oracle Visual Builder, you must configure it in your Process instance to use a Message Start or Form Start event.

You can use the following kinds of start events in Process in Oracle Integration:

- Message Start events with simple type parameters
- Message Start events with complex business type parameters
- Form Start events, which can use a mixture of types as parameters

In Oracle Visual Builder, you can map the message parameters to page variables, business objects, or service connection endpoints. You can then set up forms and action chains that execute your processes, and tables or lists that display processes and tasks. An Add Task Actions Quick Start for lists and tables allows you to add components that perform a task.

If you want to display the contents of the message parameters that you specify, you can set up a data association for the message in Process in Oracle Integration and then use it in a description property in the getTask endpoint in Oracle Visual Builder.

Configure the Connection to a Process Server

When developing applications that use business processes, you configure the settings used to connect to the Process server, the processes that are available to your application and the credentials that are used when communicating with the processes on the Process server.

The options that you select in the Process Configuration editor will determine how a process user is authenticated when communicating with the processes on the Process server, and the processes that can be selected as targets of process aliases in your application.

Process Server Authentication Options

You can use the Process Configuration window to choose the authentication method that will be used for accessing processes during development. The option only applies during the development of the application.

When an application is staged or published, access will always be based on the permissions granted to the logged in user and authenticated using the Identity Manager. By default, access to processes when designing the application, and for staged and published applications, is based on the logged in developer’s or user’s credentials. This means that by default the processes that a developer can access might be different from the processes that a logged in user can access.

During development, a developer can choose to enable Basic Authentication and provide the credentials of a user registered on the Process server. Selecting Basic Authentication enables the developer to log in to the Process server as a different user and access processes that they would not be able to otherwise. When selected, the developer can provide the credentials of different users to access the processes available to that user. These credentials and Basic Authentication are not used when the application is staged or published.
Process Server Instance Options

When you are developing your application, you can create process aliases that target processes deployed to production instances or test instances of the Process server.

When you select the **Allow test instances** check box in the Process Configuration editor, you can add process aliases and select processes deployed to test instances as the target. When the check box is deselected, you can only choose processes on the runtime instance (deployed processes configured as default) as targets. You can switch between targets on test and runtime instances by selecting **Test** or **Runtime** in the Process Configuration editor. When you select **Runtime**, the application will use the deployed process marked as default on the runtime instance.

After configuring the connection details you can choose which deployed processes you want to register with your application. Processes deployed to test instances are not available if you deselect **Allow Test Instances**.

Before you stage and publish your application, make sure you have registered the runtime process instance you will use, and deselected the **Allow Test Instances** check box. A published application must access runtime process instances.
Understand the Page Model

The page model consists of a JSON file. To work with the page model by hand, you should understand the structure and components of this JSON.

Topics:
- Variables
- Actions
- Action Chains
- Components
- Security
- Translations
- Events

Variables

Variables are the basic blocks of state management. Components and expressions in applications are bound to variables, and the variables and their structure must be defined to ensure that the design time and runtime work properly.

Topics:
- Object Variables
- Array Variables
- Built-in Variables
- Types
- Default Values
- Input Variables
- Persisted Variables

Object Variables

Variables may also be objects that contain properties.

A variable may also be an object that can contain properties. In this case, the type of the variable should be an object that defines what properties are allowed in that object.

The following variable in javascript:

```javascript
let nameOfVariable = {
  foo: "someString",
  bar: 10
}
```
could be defined like this:

```
"nameOfVariable": {
  "type": {
    "foo": "string",
    "bar": "number"
  }
}
```

**Example 8-1   An Object Containing Another Object**

This javascript object

```javascript
let otherObject = {
  foo: {
    name: "myName"
  },
  bar: 10
}
```

can be described by the following structure:

```
"otherObject": {
  "type": {
    "foo": {
      "name": "string",
    },
    "bar": "number"
  }
}
```

**Array Variables**

Variables can also represent arrays. These are defined the same way as objects. However, in this case, the object type is inside an array.

Arrays can have nested objects or arrays as well, and object types can also contain nested arrays

**Example 8-2   An Array Represented by a Variable**

A Javascript array

```javascript
let myArray = [
  {
    foo: "someString",
    bar: 10
  },
  {
    foo: "someOtherString",
    bar: 11
  }
]
```

can be represented like this:

```
"nameOfVariable": {
  "type": {
    "foo": "string",
    "bar": "number"
  }
}
```
Example 8-3    An Array of Strings

"nameOfVariable": {
  "type": "string[]"
}

Built-in Variables

There are several built-in variables available.

**currentPage**

To access some of the current page's metadata such as ID and title, there is a built-in variable named `currentPage` on the application object. The `currentPage` variable automatically update as the current page changes during navigation. This can be used to update a navigation component with the currently selected page.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$application.currentPage.id</code></td>
<td>The path of the current page. The path describes the location of the page in the flow hierarchy.</td>
</tr>
<tr>
<td><code>$application.currentPage.path</code></td>
<td>The path of the current page for the application. The path describe the location of the page in the flow hierarchy.</td>
</tr>
<tr>
<td><code>$application.currentPage.title</code></td>
<td>The title of the current page. The title is formed by pre-pending all the titles of the shells in the flow hierarchy to the current page.</td>
</tr>
<tr>
<td><code>$flow.currentPage</code></td>
<td>The id of the current page for this flow.</td>
</tr>
</tbody>
</table>

**currentFlow**

If there is a routerFlow in the page, the `$page.currentFlow` variable can be used to retrieve the id of this flow.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$page.currentFlow</code></td>
<td>The id of the current flow.</td>
</tr>
</tbody>
</table>

**path**

The path variable is used to build the path to a resource like an image located in a folder in the application or in a flow.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$application.path</code></td>
<td>The path needed to retrieve a resource located in the application folder.</td>
</tr>
<tr>
<td><code>$flow.path</code></td>
<td>The path needed to retrieve a resource in the flow folder.</td>
</tr>
</tbody>
</table>

**user**

The user variable is used to access information about the current user.
### Name | Description
---|---
$application.user.userId | The user id <string>.
$application.user.fullName | The user full name <string>.
$application.user.email | The user email <string>.
$application.user.roles | The user roles (array of strings).
$application.user.roles.<roleName> | Returns true if <roleName> is a role of this user.
$application.user.permissions | User permissions (array of strings).
$application.user.permissions.<permName> | Returns true if <permName> is a permission of this user.
$application.user.isAuthenticated | Returns true if this user is authenticated.

**translations**

This is not a variable, but an API available for getting localized strings using `$<container>.translations.<bundlename>.key,` or `$<container>.<translations>.format (<bundlename>, <key>, args...)`.

This exists for $application, $flow, and $page, but is only useful if you have defined translation bundles. If translation values are needed in Javascript function modules, they must be passed as arguments to the function.

**responsive**

This is not a variable, but contains several Knockout Observables that represent JET Media Queries. The following are available, and are accessible via $application.responsive.XXX (for example, $application.responsive.sm_Up): smUp, mdUp, lgUp, xlUp, smOnly, mdOnly, lgOnly.

**info**

Some information from the application and page descriptor can be retrieved using the info keyword.

### Name | Description
---|---
$application.info.id | The application id defined in app-flow.json
$application.info.description | The application description defined in app-flow.json
$flow.info.id | The flow id defined in <flow-id>-flow.json
$flow.info.description | The flow id defined in <flow-id>-flow.json
$page.info.title | The page title defined in <page-id>-page.json
$page.info.description | The page description defined in <page-id>-page.json

**components**

This is not a variable, but contains utility methods for finding JET components on a page. These methods return an element that is a JET component. If no element is found, or if the element is not part of a JET component, these methods will return null.
Note:
These methods are not for finding general elements To find elements on the page, use methods such as `document.getElementById`, `document.querySelector`..

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$page.components.byId('myCard')</td>
<td>Similar to document.getElementById, but returns a JET Component or null.</td>
</tr>
<tr>
<td>$page.components.bySelector('#myCompId')</td>
<td>Similar to document.querySelector, but returns a JET Component or null.</td>
</tr>
</tbody>
</table>

Types

Types can be defined at the application, flow, and page level, and can be referenced by variables.

In addition to variables, there are types. Types define structure in much the same way as variables.

Types can be defined once at the application level in the application model. This can help you to avoid using the same structure repeatedly in different variables.

Example 8-4 Using Types in the Application Model

```javascript
types: {
  "myType": {
    "type": {
      "foo": "string",
      "bar": "number"
    }
  }
}
```

Example 8-5 Referencing Types in a Variable

To reference types in a variable, prefix the type with 'application:', for example:

```javascript
"nameOfVariable": {
  "type": "application:myType"
}
```

Built-in Types

Three built-in types are available.

Topics:

- Service Data Provider
- MultiService Data Provider
- Built-in Array Data Provider
Service Data Provider

ServiceDataProvider represents a data source that fetches data from a service endpoint and that can be bound to listview and table components. It encapsulates various capabilities such as filtering, sorting, and pagination. It also allows externalizing fetches through an action chain.

ServiceDataProvider has built in support for various features like sorting, filtering, and paging of collection data, and can either fetch data implicitly, using the configured endpoint, or can fetch data externally by delegating to an external action chain. A variable that uses this built in type can be bound to collection components like listview, table, combobox/select, chart and other JET components that accept a data provider.

- **Properties**
- **Implicit and Externalized Fetches**
- **MergeTransformOptions Function**
- **Request Transformation Functions**
- **Features and Capabilities**

The ServiceDataProvider does not trigger a valueChange event, since, being stateless, its value never changes. Instead, when its properties change, or when certain methods are called, data source events are raised. Currently, UI components are the only listeners for these events.

Service Data Provider Properties

ServiceDataProvider exposes properties that a variable of this type can use to configure. All properties are directly accessible through the variable.

- **endpoint** A string that is the Rest endpoint in the format `<serviceName>/ <endpointName>`. The endpoint is typically a GET endpoint that returns a collection, and is defined in the service model.

- **fetchChainId** a string that is the 'id' of the actionChain to use to fetch the results.

- **headers** an object of the names of one or more header properties, and the corresponding values. Any headers specified here are also set on the externalized REST action by design time. Alternately, if a fetchChainId is not specified, headers are passed through to the internal REST calling mechanism by the Service Data Provider.

- **idAttribute** supports composite keys, using multiple properties. ia string or array, that is the field or fields in the response data for each row, that represents the 'id', or key, field.

Can be:

- a property name - the key, in various contexts, will also be a string.
- an array of property names - the key will also be an array, of values.
- @value, use all properties - the key will also be an array, of values.
- @index use the index as the key, - the key will be an integer.

- **itemsPath** a string that is the path to the root of the actual collection in the response. Example ‘result’ if the response returned from the endpoint looks like `{count: 10, result: [...]}`
**capabilities** an object that defines the capabilities the ServiceDataProvider and the endpoint it works with supports. The capabilities object is defined by the JETDataProvider API.

**responseType**

the type of the response that is returned by the ServiceDataProvider. This can be an object or array. By default the Service Data Provider implementation attempts to automatically map the response from fetch to the responseType. But an externalized fetch relies on the external chain to map from fetch response to the responseType. The responseType can also used in conjunction with transforms to construct a query parameter to select the object fields returned by the service. When used with the built-in vb/BusinessObjectsTransform (typical for Business Objects or RAMP services), it creates a 'fields' query parameter, which includes all the fields, both scalar and objects (and recursively includes the object's fields, as well). This will both include and expand fields.

**uriParameters**

an object that defines one or more properties that are parameters on the endpoint url. For example the FixitFast service has an endpoint to retrieve all incidents for a technician using this url - http://.../incidents?technician={technician}. Here 'technician' is a query parameter, that will be defined under uriParameters like so -

"uriParameters": {
    "technician": "{{ $page.variables.appUser.name }}"
},

The uriParameters are used to perform a simple string replacement if the url includes parameters that must be substituted before it's resolved. Otherwise the url parameters are appended to the url. The uriParameters are also passed to the query transform function (details below), so page authors can use the value of the above property to tweak the URI further if needed.

**pagingCriteria**

an object that defines the paging defaults if needed. Generally a paging component (like listview or table) will provide the data provider with size or offset or both. If the component does not provide either size or offset Service Data Provider will use the values set on this property as defaults. The pagingCriteria is then passed to the paginate transform function (see below). Supports the following properties.

- **size**: number of rows to fetch by default, when no size is provided by caller
- **offset**: the offset to start the fetch from. Defaults to 0
- **maxSize**: the default max limit of number of rows to fetch when the caller (usually components) request an unlimited number of rows. Some JET components like oj-chart often request unlimited rows by setting { size: -1 }. This property can be used to control the max rows to fetch, when it may not be performant to ask service endpoint to return indeterminate number of rows. If this property is not set then the size: -1 is passed through to the paginate transforms and it may be necessary for transforms authors to handle -1 as size.

Page authors need to understand how the above properties are used by the Service Data Provider during a fetch call:
1. Generally, the page size used by a fetch can be defaulted using the pagingCriteria.size. This is only used when a component does not explicitly provide a size. The same is true for an offset.

2. When the size is provided by caller (e.g., components), this overrides the default pagingCriteria.size set on the Service Data Provider.

   **Note:**
   When components do ask for a specific # of rows, and the Service Data Provider returns more rows than what was explicitly requested, some components can get in an indeterminate state. In such cases, to control the fetchSize, it's better to set this on the component. Specifically, oj-list-view has a scrollPolicyOptions.fetchSize.

3. Some components do not support a fetchSize property. If this is the case, you can force the fetch to be a different value from what the component requested by writing a paginate transform function where the size can be tweaked.

4. It is generally not recommended you set endpoint specific size and offset parameters using uriParameters property directly (example ramp supports 'limit' and 'offset' query parameters that are the equivalent of the pagingCriteria.size and offset). Mostly because if you do then you are on your own to write a custom ramp transform that can merge/use the value set both in uriParameters and pagingCriteria property. And you are also likely run into the caveats explained in #3.

   **filterCriterion**
   An object representing a single attribute filter criterion with properties \{ op, attribute, value \}, where 'op' is one of the supported JET attribute operators, 'attribute' and 'value', the name and value of the attribute respectively. It may also represent a compound filter criterion \{op, criteria\}, where 'op' is a compound operator, and 'criteria' is an array of attributes or compound criterion.

   Most complex filter expressions can be expressed using the new filterCriterion structure. It may sometimes be needed to externalize fetches to build your filter criteria for the REST action.

   **Note:**
   The RAMP transforms shipped with VB has support for all attribute operators except $regex. It can transform a simple attribute filter or a compound filter that is an array of attribute filter criterion. It cannot transform a compound filter that has nested compound filters.

   ```json
   {  
      "op": "$eq",
      "attribute": "empName",
      "value": "Lucy"
   }
   ```
// In RAMP, the above criterion will become the following query parameter
"q=empName = 'Lucy'"

{
    "op": "$or",
    "criteria": [
        {
            "op": "$gt",
            "attribute": "hireDate",
            "value": "2015-01-01"
        },
        {
            "op": "$le",
            "attribute": "hireDate",
            "value": "2018-01-01"
        }
    ]
}

// In RAMP, the above criterion will become the following query parameter
"q=hireDate > '2015-01-01' or hireDate <= '2018-01-01'

{
    "op": "$and",
    "criteria": [
        {
            "op": "$eq",
            "attribute": "project",
            "value": "BUFP"
        },
        {
            "op": "$or",
            "criteria": [
                {
                    "op": "$co",
                    "attribute": "label",
                    "value": "foo"
                },
                {
                    "op": "$co",
                    "attribute": "label",
                    "value": "bar"
                }
            ]
        }
    ]
}

sortCriteria

An array of objects, where each object is an atomic sort expression of the form. If you have more complex structures for representing sortCriteria then you can use the
externalized fetch option to build sort criteria and provide it to the REST action. See Externalized Fetches below for details.

```json
{
  "attribute": "<name of the field>",
  "direction": "<'ascending' (default) or 'descending'>"
}
```

**mergeTransformOptions**

This property allows page author to have a hook to fix up / merge the final transforms options that are passed to the transforms functions configured on the Service Data Provider. Let's say a sample endpoint, GET /customers endpoint supports an 'ids' query parameter that can used to query customers by specific keys.

For the above to work there is no simple way at design time to map the keys provided by component programatically to the 'ids' query parameter on the URL. In some cases it might be necessary for the page author to use this property to wire up a function that will fix or merge the transforms option.

**transforms**

An object that has 2 properties for specifying 'request' and 'response' transform functions. Request functions are primarily used to transform the url or Request configuration before calling the endpoint. Response functions can be used to process the response and return any additional state along with the response. Additional state is saved as internal state on the data source variable.

At design time, the page author will need to know whether the endpoint supports paging, sorting, filtering (or QBE), and the format/syntax for specifying these. Using the transform functions, the page author can tweak the Request to build a URL containing the paging, sorting, filtering params, and additional endpoint specific query params.

**transformsContext**

This property allows authors to set contextual information to pass to the transforms functions. This context is made available to all transforms functions both request and response. For fetchFirst calls the context will be available for all iterations. Authors can manage this object as they see fit.

**request**

An object whose properties refer to the type of the request transform functions, and the value the actual function. The following types are supported. Refer to the section on Request Transform Functions below for details.

- **paginate**: a paginate function that implements code to transform the request for pagination (or iterating through record sets) specific to the endpoint.
- **sort**: a sort function that implements code to transform the request for sorting, specific to the endpoint
- **filter**: whose value is the sort function. Note: Refer to the next section for details on how to use the transform functions.
- **query**: can be added if page author sees a need to pre-process query parameters available through uriParameters property.
- **body**: a body transform function that allows page authors to tweak the body if needed before the fetch call is made.

**response**

An object whose properties also refer to the type of the response transform function. Refer to the section on Response Transform Functions below for details.

- **paginate**: This transformation function is called immediately after the REST layer receives a response. The `paginate` response transform function is called with the response so this function can process it and return an object with the following properties set. The returned object is the primary way ServiceDataProvider obtains information about the paging state of the request:
  - **totalSize**: Optional. Used to inform ServiceDataProvider what the totalSize of the result is.
  - **hasMore**: Usually required, as with the new JET DataProvider API, the paginate response transform function is relied upon to inform ServiceDataProvider when to stop requesting to fetch more data. It is a boolean that indicates whether there are more records to fetch. For example, in RAMP use cases, this would map to the hasMore boolean property commonly returned in the response.

  An iterating component such as ServiceDataProvider requires this information in order to know when to stop iterating when fetching data.

  - **pagingState**: Optional. This can be used to store any paging state specific to the paging capability supported by the endpoint. This property can be used in the response paginate transform function to set an additional paging state. This will then be passed as is to the request paginate transform function for the next fetch call.

- **body**: this transform function is called immediately after the REST layer receives a response, is a hook for authors to transform the response body, and is not guaranteed to be called in any specific order.

**Paginate** is the only response transformation function that is currently supported. This transformation function is called immediately after the REST layer receives a response. The paginate response transform function is called with the response so that this function can process it and return an object. The returned object is the primary way a Service Data Provider knows about the paging state of the request, and is returned with the following properties set:

- **totalSize**: used to inform the Service Data Provider what the totalSize of the result is.
- **hasMore**: A boolean that indicates whether there are more records to fetch. Example in RAMP use cases this would map to the hasMore boolean property commonly returned in the response.
- **pagingState**: This can be used to store any paging state specific to the paging capability supported by the endpoint. In 1.0.0, this property can be used in the response paginate transform function, to set additional paging state. This will then be passed unchanged to the request paginate transform function for the next fetch call.

With the new JET DataProvider API we now rely on the paginate response transform function to inform the Service Data Provider when to stop requesting to fetch more data. An iterating component will get the AsyncIterator from the dataProvider and continue iterating until there is no more data to fetch, until the component viewPort is
filled, or until its current scrollPosition is reached, whichever comes first. It's important for the Service Data Provider to have the above information so that it can determine when to stop iterating.

Implicit and Externalized Fetches

When the Service Data Provider is configured with properties described in the Service Data Provider Properties section, it will, with the exception of the 'fetchChainId', it manage fetching data and notifying components implicitly.

Implicit Fetch

A typical configuration for an implicit fetching the Service Data Provider would look like this:

```json
"incidentListDataProviderImplicit": {
  "type": "vb/ServiceDataProvider",
  "description": "configuration for implicit fetches",
  "input": "none",
  "defaultValue": {
    "endpoint": "ifixitfast-service/getIncidents",
    "headers": {},
    "idAttribute": "id",
    "itemsPath": "result",
    "uriParameters": {
      "technician": "{{ $application.user.userId }}"
    }
  }
}
```

It is important to note that a Service Data Provider variable only stores the configuration in Redux. The data it fetches is not cached by the Service Data Provider instance, and is not part of the variable state stored in redux. It is not persisted to history, session or local storage.

Since the data can be arbitrarily large data sets, it is recommended that page authors use other means to cache data on client, such as the JET offline toolkit cache. This applies to externalized fetches as well.

Externalized Fetch via an Action Chain

When a 'fetchChainId' property is present, the Service Data Provider delegates the fetch to the action chain. A typical configuration for a Service Data Provider variable (supporting a fetchFirst capability), that externalizes REST will look like below. The only properties that are allowed to be configured (or that are relevant) are the ones listed below.

- capabilities: when it's not set "fetchFirst" fetch capability is assumed
- fetchChainId
- idAttribute
- itemsPath
- mergeTransformOptions: this property is defined on the Service Data Provider variable, because merging transform options only applies when an action chain (with a RestAction) is called in the context of a data provider fetch call.
• `responseType`

```json
"variables": {
  "incidentListTableSource": {
    "type": "vb/ServiceDataProvider",
    "input": "none",
    "persisted": "session",
    "defaultValue": {
      "fetchChainId": "fetchIncidentListChain",
      "idAttribute": "id",
      "itemsPath": "result",
      "responseType": "application:incidentsResponse"
    }
  },
},
"chains": {
  "fetchIncidentListChain": {
    ...
  },
}
```

The type definition of "application:incidentsResponse" used by 'responseType' property can be seen in this example. This structure is similar to the one returned from a REST response. Note that itemsPath is always located within the 'body' of the response.

```json
"incidentsResponse": {
  "type": {
    "status": "string",
    "headers": "object",
    "body": {
      "result": "application:incidentSummary[]"
    }
  },
},
"incidentSummary": {
  "type": {
    "id": "string",
    "problem": "string",
    "priority": "string",
    "status": "string",
    "customer": "application:customer"
  }
},
```

Generally, users externalize fetches to ensure full control over how the request and response are processed.

Users can connect custom sort and filter query parameters either in the service endpoint or in the REST action. This is the preferred configuration approach. If, however, properties like sortCriteria, filterCriterion, transforms etc., are defined on the ServiceDataProvider, they will be merged with those configured on the Rest action, after the ones on the REST action have been processed, when building the Request.
In the example below the action chain 'fetchIncidentListChain' defined in the fetchChainId property of the ServiceDataProvider variable above has a typical chain configuration, one of which is a RestAction.

1. The 'hookHandler' property under configuration chain variable will be automatically generated at design time and is always set to vb/RestHookHandler.

2. If the REST response returns a structure that is exactly what the Service Data Provider expects this can be returned directly (as the sample below). But if the REST response is different from the expected responseType, then an action that maps the REST response to the structure defined by 'responseType' can be configured.

3. The last action in the chain will always be a ReturnAction whose payload resembles responseType. The response variable in the chain is provided for clarity but is not used by the chain.

"chains": {
    "fetchIncidentListChain": {
        "variables": {
            "configuration": {
                "type": {
                    "hookHandler": "vb/RestHookHandler"
                },
                "description": "the configuration for the rest action",
                "input": "fromCaller",
                "required": true
            },
            "response": {
                "type": "application:incidentsResponse"
            }
        },
        "root": "fetchIncidentList",
        "actions": {
            "fetchIncidentList": {
                "module": "vb/action/builtin/restAction",
                "parameters": {
                    "endpoint": "ifixitfast-service/getIncidents",
                    "uriParams": {
                        "technician": "{{ $application.user.userId }}"
                    },
                    "hookHandler": "{{ $variables.configuration.hookHandler }}",
                    "requestTransformOptions": {
                        "sort": "{{ $page.variables.sortExpression }}",
                        "filter": "{{ $page.variables.filterAtomicExpression }}"
                    },
                    "requestTransformFunctions": {
                        "paginate": "{{ $page.functions.paginate }}",
                        "query": "{{ $page.functions.query }}",
                        "filter": "{{ $page.functions.filter }}",
                        "sort": "{{ $page.functions.sort }}"
                    },
                    "responseTransformFunctions": {
                        "paginate": "{{ $page.functions.paginateResponse }}"
                    }
                }
            }
        }
    }
}
"outcomes": {
  "success": "returnSuccessResponse",
  "failure": "returnFailureResponse"
}
},
"returnSuccessResponse": {
  "module": "vb/action/builtin/returnAction",
  "parameters": {
    "outcome": "success",
    "payload": "{{ $chain.results.fetchIncidentList }}"
  }
}
},
"returnFailureResponse": {
  "module": "vb/action/builtin/returnAction",
  "parameters": {
    "outcome": "failure",
    "payload": "{{ $chain.results.fetchIncidentList }}"
  }
}
}
}
}

MergeTransformOptions Function

A page author can use the mergeTransformsOption property on the ServiceDataProvider fetch to fix up the transforms options that will be passed to the transforms functions, if and when needed. The function will be passed two parameters: 'configuration' and 'transformOptions'.

The configuration object will contain one set of { capability, context , fetchParameters } as a request servicing one fetch capability. For the configuration object:

<table>
<thead>
<tr>
<th>Property</th>
<th>Sub-property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capability</td>
<td>idAttribute</td>
<td>fetchByKeys</td>
<td>A hint that supplies the author the fetch capability. Provides a snapshot of the ServiceDataProvider variable at the time the fetchByKeys() call is made. For external chains, the state may not include all properties listed here.</td>
</tr>
<tr>
<td>context</td>
<td>itemsPath</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>uriParameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>filterCriterion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>sortCriteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>pagingCriteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>responseType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>capabilities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>fetchByKeys</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>keys</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fetchKeys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Sub-property</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>capability</td>
<td></td>
<td>fetchByOffset</td>
<td>A hint telling the author the fetch capability for the current request.</td>
</tr>
<tr>
<td>context</td>
<td>idAttribute</td>
<td></td>
<td>A snapshot of the value of the ServiceDataProvider variable at the time the fetchByOffset() call was made.</td>
</tr>
<tr>
<td></td>
<td>itemsPath</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uriParameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>filterCriterion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sortCriteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pagingCriteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>responseType</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>capabilities</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>fetchByKeys</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>keys</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fetch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td>filterCriterion</td>
<td></td>
<td>The original parameters passed in via the fetchByOffset call.</td>
</tr>
<tr>
<td></td>
<td>size</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sortCriteria</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Sub-property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capability</td>
<td></td>
<td>fetchFirst</td>
<td>A hint telling that the request is a fetchFirst capability.</td>
</tr>
<tr>
<td>context</td>
<td>idAttribute</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>itemsPath</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uriParameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>filterCriterion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sortCriteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pagingCriteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>responseType</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>capabilities</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>fetchByKeys</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>keys</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fetch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td>filterCriterion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>size</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sortCriteria</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Property**
- query
- filter
- paginate
- sort
- select

*These are the properties when the Service Data Provider is configured for implicit fetch*

*When the Service Data Provider is configured to use an external fetch chain, the options configured on the RestAction 'requestTransformOptions' property will be made available here.*

A sample endpoint, GET /customers endpoint, supports an 'ids' query parameter that can used to query customers by specific keys, such as customers?
ids=cus-101,cus-103.
For this to work, there is currently no easy way at design time to map the keys provided by component programmatically to the 'ids' query parameter on the URL. It might be necessary for page authors to use this property to wire up a function that will merge the transforms option.

This should be configured as follows:

1. The Service Data Provider variable below defines a fetchByKeys capability.
   - The 'mergeTransformOption' property is configured to point to a page function.

   "customerSingleSDP_External": {
     "type": "vb/ServiceDataProvider",
     "defaultValue": {
       "endpoint": "demo-data-service/getCustomers",
       "idAttribute": "id",
       "itemsPath": "result",
       "capabilities": {
         "fetchByKeys": {
           "implementation": "lookup"
         }
       }
     }
   }
   "mergeTransformOptions": {
     "{{ $page.functions.processOptionsForGetCustomers }}"
   }

2. The page author uses the function to fix up the 'query' transform options that will be passed to the query transform function.
   - The page function "{{ $page.functions.processOptionsForGetCustomers}}" will look like the following:

   /**
   * fix up the query transform options.
   * When the fetchByKeys capability is set, the 'keys' provided via the fetch call can be looked up via the configuration.fetchParameters. This can be set/merged onto the 'query' transform options (1).
   * This allows the transform function to then use the keys to build the final 'ids=' query param on the url.
   * See queryCustomersByIds method.
   *
   * Note: (1) this is needed because there is no way through DT configuration to define a mapping of 'keys' that provided via a fetch call, to the 'ids' query parameter.
   *
   * @param configuration a map of 3 key values, The keys are
   * - fetchParameters: parameters passed to a fetch call
   * - capability: 'fetchByKeys' | 'fetchFirst' | 'fetchByOffset'
   * - context: the context of the SDP when the fetch was initiated.
   *
   * @param transformOptions a map of key values, where the keys are the names of the transform functions.
   * @returns {"*}
PageModule.prototype.processOptionsForGetCustomers =
  function (configuration, transformOptions) {
    var c = configuration;
    var to = transformOptions;
    var fbkCap = !!((c && c.capability === 'fetchByKeys'));
    var keysToFetch = fbkCap ? (c && c.fetchParameters &&
      c.fetchParameters.keys) : null;

    if (fbkCap && keysToFetch && keysToFetch.length > 0) {
      // join keys
      var keysToFetchStr = keysToFetch.join(',');
      to = to || {};
      to.query = to.query || {};
      // ignore ids set on the query options and instead use ones passed
      in by fetchByKeys call
      to.query.ids = keysToFetchStr;
    }

    return to;
  };

3. • A query transform function is not needed in the above example because the
query parameters are automatically appended to the final request url if no
additional transformation of the query options to the query parameter is
needed.
  • A query transform function might be needed in more complex usecases.

Request Transformation Functions

A request transformation function is generally specified on the service endpoint. This
also can be specified on the Service Data Provider variable, which overrides the
endpoint one.

A request transformation function has the following signature: function
(configuration, options) { return configuration }. The parameters to the
function are:

• configuration: an object that has the following properties:
  – url: Full url of the request.
  – readOnlyParameters: Path and query parameters. These are not writable.
  – initConfig: Map of other configuration passed into the request. The ‘initConfig’
    exactly matches the ‘init’ parameter of the request.

• options: An object that is relevant to the type of transformation function. For a
  filter function this would be the filterCriterion.

• context: A context object that is passed to every transform function to store or
  retrieve any contextual information for the current request lifecycle.

If transformations are needed for a specific data provider instance, these functions can
be defined on the ServiceDataProvider variable under the ‘transforms’ property. For
externalized fetch cases the RestAction properties can be used for configuring
transformations.
Types of request transform functions

paginate

The 'pagingCriteria' is passed in as the 'options' parameter to the paginate function. The pagingCriteria is often based on the current paging/scrolled state of the component.

- For 'implicit fetches' the pagingCriteria provided to the 'paginate' transform function can be used to construct an URL with the right paging query
- For 'externalize fetches' the pagingCriteria is always set on the Rest instance through the hook handler. This simply means that if the RestAction has a responseTransformFunctions.paginate transform function property configured then it can expect the pagingCriteria to be provided to it

For offset based paging:

- **size**: Specifies how many items should be returned.
- **offset**: Specifies which item should the response begin from.
- The default value for the pagingCriteria can be set on the configuration, but will generally a component that is bound to the Service Data Provider variable will provide the values, where offset and size will be based on the configuration set in the component.

```javascript
"incidentListTableSource": {
  "type": "vb/ServiceDataProvider",  // variable of type
  "input": "none",
  "defaultValue": {
    "pagingCriteria": {  // default size
      "size": 10
    },
    "transforms": {  // transform function for paginate
      "request": {
        "paginate": "{{ $page.functions.paginate }}"
      }
    }
  }
}
```

// Transform Function appends limit and offset parameters to the url
PageModule.prototype.paginate = function (configuration, options, context) {
  const c = configuration;
  let newUrl = c.url;
  newUrl = `${newUrl}&limit=${options.size}&offset=${options.offset}`;
  c.url = newUrl;
  return c;
};

filter

For this transform function, the 'filterCriterion' property is passed in as the 'options' parameter. The filterCriterion JSON property is an object representing a attribute
criterion or a compound criterion. This example defines a simple structure for filter
criteria that is a single criterion:

```
"incidentListTableSource": {
  "type": "vb/ServiceDataProvider",
  "input": "none",
  "defaultValue": {
    "filterCriterion": { // filterCriterion
      "attribute": "",
      "op": "eq",
      "value": ""
    },
    "transforms": {
      "request": {
        "filter": "{{ $page.functions.filter }}" // transform function for filter
      }
    }
  }
}
```

A sample filter transform function written to convert the filterCriterion property to query parameter appropriate to the endpoint:

```javascript
/**
 * Filter Transform Function Implementation
 * @param configuration
 * @param options the JSON payload that defines the filterCriterion
 * @param context an object to store/retrieve any contextual information for the current request lifecycle.
 * @returns {object} configuration object. the url looks like ?filter=foo eq 'bar'
 */

PageModule.prototype.filter = function (configuration, options, context) {
  const c = configuration;
  const filterCriterion = options;

  function jetFilterOpToScim(fop) {
    switch (fop) {
      case '$eq':
        return 'eq';
      case '$ne':
        return 'ne';
      case '$co':
        return 'co';
      default:
        console.warn('unable to interpret the op ' + fop);
        return null;
    }
  }

  function isEmpty(val) {
```
return (val === undefined || val === null || val === '');
}

if (typeof filterCriterion === 'object' && Object.keys(filterCriterion).length > 0) {
    if (filterCriterion.op && filterCriterion.attribute && !isEmpty(filterCriterion.value)) {
        const atomicExpr = {};
        atomicExpr.op = jetFilterOpToScim(filterCriterion.op);
        atomicExpr.attribute = filterCriterion.attribute;
        atomicExpr.value = filterCriterion.value;

        if (atomicExpr.op && atomicExpr.attribute) {
            c.url = URI(c.url).addQuery({
                filter: `${atomicExpr.attribute} ${atomicExpr.op} ${atomicExpr.value}`
            }).toString();
        }
    }
}

return c;

sort

For this transform function, the 'sortCriteria' is passed in as the 'options' parameter. If page authors have complex sort expressions that cannot be expressed as a simple array, they can externalize the fetch to configure their own sort criteria and build a request using that.

"incidentListTableSource": {
    "type": "vb/ServiceDataProvider",
    "input": "none",
    "defaultValue": {
        "sortCriteria": [  // sortCriteria property
default value
            {
                "direction": "ascending"
            }
        ],

        "transforms": {
            "request": {
                "sort": "{{ $page.functions.sort }}"  // transform function for sort
            }
        }
    }
}
PageModule.prototype.sort = function (configuration, options, context) {
    const c = configuration;

    if (options && Array.isArray(options) && options.length > 0) {
        const firstItem = options[0];
        if (firstItem.name) {
            const dir = firstItem.direction === 'descending' ? 'desc' : 'asc'
            let newUrl = c.url;
            newUrl = `${newUrl}&orderBy=${firstItem.attribute}:${dir}`;
            c.url = newUrl;
        }
    }
    return c;
};

query

For this transform function the 'uriParameters' property is passed in as options. Normally uriParameters are appended to the URL automatically, but there may be cases where user would want to adjust the query parameters. For example, when the endpoint GET /incidents supports a query parameter called "search", which does a semantic search. If there is a specific transform that needs to happen before calling the endpoint, then the transform function could be used for that.

Variable Configuration:

"incidentListTableSource": {
    "type": "vb/ServiceDataProvider",
    "input": "none",
    "defaultValue": {
        "uriParameters": {
            "technician": "hcr",
            "search": "{{ $page.variables.searchBoxValue }}" // search query parameter bound to some UI field.
        },
        "transforms": {
            "request": {
                "query": "{{ $page.functions.query }}" // transform function for query
            }
        }
    }
}
query() Transform Function:

PageModule.prototype.query = function (configuration, options, context) {
    const c = configuration;
    if (options && options.search) {
        let newUrl = c.url;
        newUrl = `${newUrl}&search=${options.search}`; // appends 'faq' to the search term
        c.url = newUrl;
    }
    return c;
// configuration, options};
}

select

This transform typically uses the 'responseType' to construct a query parameter to select and expand the fields returns from the service. The built-in vb/BusinessObjectsTransforms creates a 'fields' query parameter, such that the response will include all fields in the responseType structure, including expanded fields.

function appendToUrl(url, name, value) {
    // skip undefined and null
    if (value !== undefined && value !== null) {
        var sep = url.indexOf('?') >= 0 ? '&' : '?';
        return url + sep + name + '=' + value;
    }
    return url;
}

function queryParamExists(url, name) {
    const q = url.indexOf('?', url.indexOf(name) + name.length);
    if (q >= 0) {
        return (url.indexOf('${name}') === q) || (url.indexOf('&${name}') > q);
    }
    return false;
}

PageModule.prototype.select = function(configuration, options, context) {
    // the options should contain a 'type' object, to override
    var c = configuration;

    // do nothing if its not a GET
    if (c.endpointDefinition && c.endpointDefinition.method !== 'GET') {
        return c;
    }

    // do nothing if there's already a '?fields='
    if (queryParamExists(c.url, 'fields')) {
        return c;
    }
// if there's an 'items', use its type; otherwise, use the whole type
var typeToInspect = (options && options.type && (options.type.items || options.type));

if (typeToInspect && typeof typeToInspect === 'object') {

    var fields; // just an example; query parameter construction is left to the developer
    if (fields) {
        c.url = appendToUrl(c.url, 'fields', fields);
    }
    return c;
}

Request Transformation Function

Features and Capabilities

At design time, a page author may need to know which features and capabilities that the endpoint supports, and they will need to configure the correct properties and transform.

Events

The events raised by data provider are as per contract defined for oj.DataProvider. These events are fired at appropriate times to notify UI components. Page authors may need to force the variable to fire some of the DataProvider events, including 'add', 'remove', 'refresh', and 'update'.

vbDataProviderNotification

Page authors can now register an event listener of this type, in order to be notified of catastrophic errors that may occur when something goes wrong during an implicit fetch. For externalized fetch where the fetch is externalized to a action chain, the current mechanism of handling failure outcomes can continue to be used. The event payload available to the listener is an object that has the following properties:

- severity, // string
- detail, // any, details of the error, this could have the Rest failure details
- capability: // object with the capabilities configured on the ServiceDataProvider
- fetchParameters: // object with the parameters passed to the fetch
- context: // object representing the state of the SDP at the time fetch was initiated
- id: uniqueld, // string, id of the SDP instance
- key: // since the event can be fired multiple times, this identifies the event instance

Page authors can use this to display an error message.

Example 8-6  Firing a DataProvider event via fireDataProviderEvent action

A page is configured to have a master list and detail form showing the details of the current selected row on the list. If the form is wired to PATCH to a different endpoint
than the one configured on the list. When user updates the form data, it’s desirable for
the same actionChain to also raise the ‘update’ event on the Service Data Provider so
it can show the changes to the current row. To configure the page:

```html
<!-- list view bound to page variable incidentListTableSource -->
<oj-list-view id="listview"
    data="{{variables.incidentListTableSource}}"

...  
</oj-list-view>

<!-- form UI fields bound to page variable currentIncident -->
<div class="oj-form-layout"
    <div class="oj-form"
        <div class="oj-flex"
            <div class="oj-flex-item"
                <oj-label for="problem"Problem</oj-label>
            </div>
            <div class="oj-flex-item"
                <oj-input-text id="problem"
                    value="{{variables.currentIncident.problem}}"
                    required=true</oj-input-text>

                </div>
            </div>

...  
</div>

<!-- Save button bound to componentEvent handler 'saveIncident' -->
<oj-button href="#" id='saveButton'
    label='Save'
    on-dom-click='[[${componentEvents.saveIncident}]]'</oj-button>

// saveIncident calls the actionChain 'saveIncidentChain', which
// (1) defines 2 variables - incidentId and incidentPayload
// (2) then calls a REST action to put/patch payload
// (3) then it takes the result from (2) and assigns to incidentsResponse
// chain variable,
// (4) calls an actionChain to fire a data provider event to refresh the
// SDP page variable
// (5) an update event payload passed to the action chain
"saveIncidentChain": {
    "variables": {                                                // (1)
        "incidentId": {
            "type": "string",
            "description": "the ID of the incident to update",
            "input": "fromCaller",
            "required": true
        },
        "incidentPayload": {
            "type": "object",
            "description": "the payload of the incident data",
            "input": "fromCaller",
            "required": true
        },
        "incidentsResponse": {
            "type": "application:incidentsResponse"
        }
    }
}
```
"root": {
  "id": "saveIncidentToRest",       // (2)
  "module": "vb/action/builtin/restAction",
  "parameters": {
    "endpoint": "ifixitfast-service/putIncident",
    "uriParams": {
      "id": "{{ $variables.incidentId }}"
    },
    "body": "{{ $variables.incidentPayload }}"
  },
  "outcomes": {
    "success": "assignVariables_incidentsResponse"
  }
},
"assignVariables_incidentsResponse": {
  "module": "vb/action/builtin/assignVariablesAction",
  "parameters": {
    "$variables.incidentsResponse.result": {
      "source": "{{ $chain.results.saveIncidentToRest.body }}" // (3)
    }
  },
  "outcomes": {
    "success": "updateIncidentList"
  }
},
"updateIncidentList": {
  "module": "vb/action/builtin/callChainAction",
  "parameters": {
    "id": "fireDataProviderMutationEventActionChain",       // (4)
    "params": {
      "payload": {
        "update": {
          "data": "{{ $variables.incidentsResponse }}"
        }
      }
    }
  }
},
"fireDataProviderMutationEventActionChain": {
  "variables": {
    "payload": {
      "type": "application:dataProviderMutationEventDetail",
      "input": "fromCaller"
    }
  },
  "root": "fireEventOnDataProvider",
  "actions": {
    "fireEventOnDataProvider": {
      "module": "vb/action/builtin/fireDataProviderEventAction",
      "parameters": {
        "payload": {
          "type": "application:dataProviderMutationEventDetail",
          "input": "fromCaller"
        }
      }
    }
  }
}
"target": "{{ $page.variables.incidentListDataProvider }}", // SDP variable on which the event is fired
"add": "{{ $variables.payload.add }}",
"remove": "{{ $variables.payload.remove }}",
"update": "{{ $variables.payload.update }}" // has the updated record details
}
}
}

Example 8-7  Using the vbDataProviderNotification

"vbDataProviderNotification": {
  "chains": [
    {
      "chainId": "someChainX"
    }
  ]
}

MultiService Data Provider

vb/MultiServiceDataProvider

Often components that bind to data providers, like oj-combobox-one /oj-select-one (or the -many variants), require or use different ‘fetch’ capabilities on the data provider implementation.

Design Time Assumptions

- A service author can identify different endpoints that provide the fetchByKeys capability, in addition to the current fetch all (i.e. fetchFirst capability).

Type Definition

A variable of the new built-in type vb/MultiServiceDataProvider can be configured with the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Sub-property</th>
<th>Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataProviders</td>
<td>fetchFirst</td>
<td>&quot;vb/ServiceDataProvider&quot;</td>
<td>{</td>
<td>A MultiService Data Provider is needed only when more than one fetch capability needs to be configured.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;variables&quot;:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;activitiesMultiSDP&quot;:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;type&quot;: &quot;vb/MultiServiceProvider&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;defaultValue&quot;:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;dataProvider&quot;:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;fetchFirst&quot;: &quot;{{ $variables.listSDP }}&quot;</td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Property</th>
<th>Sub-property</th>
<th>Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchByKeys</td>
<td>&quot;vb/ServiceDataProvider&quot;</td>
<td></td>
<td>{</td>
<td>A reference to the vb/SDP variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;variables&quot;:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;activitiesMultiSDP&quot;: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;type&quot;: &quot;vb/MultiServiceDataProvider&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;defaultValue&quot;: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;dataProviders&quot;: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;fetchFirst&quot;:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;{{ $variables.listSDP }}&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;fetchByKeys&quot;:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;{{ $variables.detailSDP }}&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Sub-property</td>
<td>Type</td>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| fetchByOffset          |              | "vb/ServiceDataProvi
der" | {                                                                      |             |
|                        |              |                    | "variables": {                                                        |             |
|                        |              |                    |                            {                                              |             |
|                        |              |                    | "activitiesMultiSDP": {                                              |             |
|                        |              |                    |                            "type": "vb/MultiServiceDataProvider",                   |             |
|                        |              |                    |                            "defaultValue": {                                 |             |
|                        |              |                    |                            "dataProviders": {                                  |             |
|                        |              |                    |                            "fetchFirst": {                                   |             |
|                        |              |                    |                            "{{ $variables.listSDP }}"                        |             |
|                        |              |                    |                            "fetchByOffset": {                                 |             |
|                        |              |                    |                            "{{ $variables.listSDP }}"                        |             |

**Behavior**

A variable of type vb/MultiServiceDataAuthProvider must have at least one fetch capability defined, otherwise an error is flagged. When a fetchFirst capability is not defined, a noop fetchFirst capability is used. The JET DataProvider contract requires a fetchFirst implementation to be provided. All fetch capabilities must point to a variable of type vb/ServiceDataAuthProvider. A Multiservice Data Provider cannot reference another Multiservice Data Provider variable.

**vb/MultiServiceDataAuthProvider Configuration**

In design time, a variable using this new type will be created that looks like below:

```json
1  {
2    "variables": {
3      "countriesMultiSDP": {
```
Line 3: countriesMultiSDP is a variable of the new type vb/MultiServiceDataProvider. This defines two properties: ‘fetchFirst’ and ‘fetchByKeys’.

Line 7: The fetchFirst property allows the MultiService Data Provider to call fetchFirst() on the referenced Service Data Provider variable.

Line 8: fetchByKeys property allows the MultiService Data Provider to call fetchByKeys() on the Service Data Provider.

**vb/ServiceDataProvider Variables Configuration**

For the above use case the referenced Service Data Provider variables will be configured as follows:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 3: defines the Service Data Provider variable with a fetchFirst capability.</td>
<td></td>
</tr>
<tr>
<td>• when a capabilities property is not specified then it's assumed that the Service Data Provider supports a fetchFirst capability. or</td>
<td></td>
</tr>
<tr>
<td>• when a capabilities property is present but no fetch capability is defined, iow, just the filter and sort capabilities are defined, again fetchFirst is assumed</td>
<td></td>
</tr>
<tr>
<td>Line 6: defines the endpoint to use the getAllCountries operation to fetch all countries</td>
<td></td>
</tr>
</tbody>
</table>

```json
{  "variables": {    "allCountriesSDP": {      "type": "vb/serviceDataProvider",      "defaultValue": {        "endpoint": "rest-service/getAllCountries",        "idAttribute": "alpha3Code"      }    },    "countriesByCodesSDP": {      "defaultValue": {      }    }  }
```
Line 4: defines the Service Data Provider variable that supports a fetchByKeys capability.

Line 7: uses the getCountriesByCodes operation to fetch a list countries by their codes

Line 9: a new ‘capabilities’ property is added to Service Data Provider that has a ‘fetchByKeys’ property object. See next section for details

• ‘implementation’ property is set to “lookup”,
• ‘multiKeyLookup’ property set to “no”

Line 15: the ‘mergeTransformOptions’ property is set to a page function.

• this is needed so page author can map the keys set programmatically 3 to be turned into the query parameters ‘?codes=’

Note:

Normally fetchByKeys() is called by a JET component programmatically with one or more keys.

• when keys are provided programmatically, Service Data Provider will use a best guess heuristic to map keys to the appropriate transform options. But when this is not easily decipherable by Service Data Provider 4, page authors can use a ‘mergeTransformOptions’ property that maps to a function, to fix up the list of the ‘query’ options. This function will be passed in all the info it needs to merge the final transform options.

Note:

In this example the keys needs to map to the codes uriParameters and such a mapping cannot be represented in the page model using a VB expression.

• when no keys are provided, Service Data Provider will throw an error
### Configuration

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>fix up the query transform options.</td>
</tr>
<tr>
<td>3</td>
<td>When the fetchByKeys capability is set, the 'keys' provided via the fetch call can be looked up via the configuration.fetchParameters.</td>
</tr>
<tr>
<td>4</td>
<td>This can be used to set a 'codes' property on the 'query' transform options whose value is the keys provided via a fetch call.</td>
</tr>
<tr>
<td>8</td>
<td>@param configuration a map of key values, The keys are</td>
</tr>
<tr>
<td>9</td>
<td>- fetchParameters: parameters passed to a fetch call</td>
</tr>
<tr>
<td>10</td>
<td>- capability: 'fetchByKeys'</td>
</tr>
<tr>
<td>11</td>
<td>context: the context of the SDP when the fetch was initiated.</td>
</tr>
<tr>
<td>12</td>
<td>@param transformOptions a map of key values, where the keys are the names of the transform functions.</td>
</tr>
<tr>
<td>13</td>
<td>@returns {*}</td>
</tr>
</tbody>
</table>

### Variables

Line 17: function that fixes up the transforms options that will be sent to the transforms functions.

Line 33: set a new 'codes' query parameter, whose value is a ';' separated list of country alpha codes.
```javascript
28     to = to || {};
29     to.query = to.query || {};
30
31     // ignore codes set on the
query options and instead use ones
passed in
32     // by fetchByKeys call
33     to.query.codes =
keysToFetchStr;
34   }
35
36   return to;
37 };
```

### Configuring a JET Combo/Select in Design Time

To configure an LOV field that uses the above design time, three variables need to be created:

- 1 vb/MultiServiceDataProvider variable
- 2 vb/ServiceDataProvider variables

The MultiService Data Provider variables are what are bound to the combo/select components:

<table>
<thead>
<tr>
<th>sample-page.html</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 &lt;oj-combobox-one id=&quot;so11&quot; value=&quot;{{ $variables.selectedActivities }}&quot;</td>
<td>• Line 2 points to a variable of type vb/MultiServiceDataProvider</td>
</tr>
<tr>
<td>2 options=&quot;[[ $variables.countriesMultiSDP ]&quot;]&quot;</td>
<td></td>
</tr>
<tr>
<td>3 options-keys.label='[[ &quot;name&quot; ]]'</td>
<td></td>
</tr>
<tr>
<td>4 options-keys.value='[[ &quot;alpha3Code&quot; ]]'</td>
<td></td>
</tr>
<tr>
<td>5 &lt;/oj-combobox-one&gt;</td>
<td></td>
</tr>
</tbody>
</table>

A distinct vb/SDP variable is needed for each unique service/endpoint.

Any individual vb/ServiceDataProvider variables might externalize its fetch, or allow an actionChain to assign values to its properties directly via expressions. They can also allow a fireDataProviderEventAction to reference the Service Data Provider variable directly. First class variables are the easiest way to give page authors access.

**When a service provides unique endpoints for different fetch capabilities, but the fetchByKeys endpoint only supports a single key based lookup**
In this use case, the services supports a fetchFirst capability that fetches all rows, and a fetchByKeys capability that returns a single row by its key. There is no endpoint that can return rows by multiple keys.

vb/MultiServiceDataProvider Variable Configuration

The configuration for the vb/MultiServiceDataProvider variable is similar to the previous examples.

```json
1  {
2    "variables": {
3      "countriesMultiSDP": {
4        "type": "vb/MultiServiceDataProvider",
5        "defaultValue": {
6          "dataProviders": {
7            "fetchFirst": "{{ $page.variables.allIncidentsSDP }}"
8            "fetchByKeys": "{{ $page.variables.incidentBySingleKeySDP }}"
9          }
10      }
11    }
12  }
```

vb/ServiceDataProvider Variables Configuration

For the previous use case, the referenced Service Data Provider variables will be configured as follows:
Configuration | Description
--- | ---
some-page.json | - Line 2 points to a variable of type vb/MultiServiceDataProvider

```json
1  {
2     "variables": {
3         "allIncidentsSDP": {
4             "type": "vb/ServiceDataProvider",
5             "defaultValue": {
6             "endpoint": "fixitfast-service/getAllIncidents",
7                 "idAttribute": "id",
8             "itemsPath": "result",
9             "uriParameters": {
10            "technician": "{{ $application.user.userId }}"
11          } }
12     }
13 },
14     "incidentBySingleKeySDP": {
15       ...
16     }
```
Line 4: defines the Service Data Provider variable with the fetchByKeys capability. The Service Data Provider variable is configured for an implicit fetch.

Line 7: uses the getIncident operation to fetch a single incident by its id

Line 9: maps the ‘id’ key in the ‘uriParameters’.
- At runtime the ‘id’ key value is substituted in the path parameter of the url
- example, if ‘id’ value is "inc-101" then the request url goes from https://.../incidents/{id} → http://.../incidents/inc-101

Line 12: a new ‘capabilities’ property is added to Service Data Provider that has a ‘fetchByKeys’ key object
- ‘implementation’ property is set to "lookup",
- ‘multiKeyLookup’ property set to "no", as endpoint only supports lookup using a single key at a time

Notice that a ‘mergeTransformOptions’ property is not set.
- this is because Service Data Provider uses a simple heuristic to map the ‘keys’ provided programmatically, to the ‘id’ sub-property of the ‘uriParameters’
  - it can do this because Service Data Provider sees that the idAttribute "id", is the same attribute key set on ‘uriParameters’
  - also, this is only possible when Service Data Provider is configured to use implicit fetch, iow, it does not use an external action chain to do a fetch

- In some cases Service Data Provider cannot easily decipher the mapping (as seen in the previous example), and this is when page authors can use a ‘mergeTransformOptions’ property to fixup / map the keys to the right transform options.
- when multiple keys are provided by caller, Service Data Provider as an optimization calls the single endpoint a single key at a time, assembles the result and returns this to caller.
Line 4: defines the Service Data Provider variable with a fetchByKeys capability.
- the Service Data Provider variable uses an action chain to fetch data. See the next section for the action chain configuration.

Line 9: sets a mergeTransformOptions function
- this function is used by page author to fix up the 'query' transform options to use the key passed in via the fetch call.
/**
 * Process the transform options.
 * When Service Data Provider uses
 * external fetch chain, it doesn't
 * have all the information to build
 * the final transform options
 * to use with the transform
 * functions. In such cases the page
 * author can use this method to build
 * the final
 * list of options. Replaces id set
 * via configuration with the value
 * passed in by caller.
 *
 * @param configuration an Object
 * with the following properties
 *   - capability: `fetchByKeys` | `fetchFirst` | `fetchByOffset`
 *   - fetchParameters: parameters
 *     passed to the fetch call
 *   - context: the context of the
 *     Service Data Provider variable at
 *     the time the fetch call was made
 *
 * @param transformOptions a map of
 * key values, where the keys are the
 * names of the transform functions.
 *
 * @returns {*} the
 * transformOptions either the same
 * one passed in or the final fixed up
 * transform options
 */

PageModule.prototype.fixupTransformOptions = function (configuration, transformOptions) {
  var c = configuration;
  var to = transformOptions || {};
  var fetchByKeys = !!(c &&
    c.capability === 'fetchByKeys');

  if (fetchByKeys) {
    var key =
      c.fetchParameters.keys[0];
    if (key &&
      (!to.query || (to.query &&
        to.query.id !==
        c.fetchParameters.keys[0]))) {
      to.query = to.query || {};
      to.query.id = key;
    }
  }
}
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>return to;</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
The external fetch action chain is configured as follows:

Line 4: the action chain used by the Service Data Provider

Line 23: the RestAction, the chain calls to fetch a single incident by id

Line 28: the 'uriParams' property of the RestAction is set to the page variable "incidentId"

• the value of the "incidentId" variable might be different from what the caller passes in.

• the mergeTransformOptions function above builds the query options containing the final id value

Line 31: the requestTransform Function.query maps to a query transform function that substitutes the endpoint url with the final id value
```javascript
/**
 * query transform function that takes the id provided in the options and expands the url.
 * @param configuration
 * @param options
 * @returns {*} */
PageModule.prototype.queryIncidentById = function (configuration, options) {
const c = configuration;
if (options && options.id) {
  var result = URI.expand(c.endpointDefinition.url, {
    id: options.id
  });
  var newUrl = result.toString();
  if (newUrl !== c.url) {
    console.log(`typesDemo sample: replacing ${c.url} with ${newUrl}`);
  }
  c.url = newUrl;
}
return c;
};
```

**query transforms function**

When the same endpoint supports multiple fetch capabilities

Most lovs fall into this category. For example to fetch both a list of territories and to fetch a subset of territories by their ids, the same endpoint is used:

- **fetchFirst capability:**
  - service/endpoint: fa-crm-service/getTerritories
  - GET /findTerritories?finder=EnabledFlagFinder;BindEnabledFlag=Y
- **fetchByKeys capability:**

---

**Configuration**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>&quot;failure&quot;:</td>
</tr>
<tr>
<td>37</td>
<td>&quot;returnFailureResponse&quot;</td>
</tr>
<tr>
<td>38</td>
<td>}</td>
</tr>
<tr>
<td>39</td>
<td>},</td>
</tr>
<tr>
<td>40</td>
<td>}</td>
</tr>
<tr>
<td>41</td>
<td>}</td>
</tr>
<tr>
<td>42</td>
<td>}</td>
</tr>
</tbody>
</table>
GET /fndTerritories?
finder=EnabledFlagFinder;BindEnabledFlag=Y&q=TerritoryCode IN ('AE', 'AD', 'US')

In this case, though it might be desirable to have one definition for the variable of type vb/ServiceDataProvider that multiplexes different fetch capabilities, the recommended approach is to have one Service Data Provider per fetch capability, then combine them using the vb/MultiServiceDataProvider. The latter will be used to bind to components.

vb/ServiceDataProvider Variables Configuration

The data returned by the service endpoint will look like this:

```json
{
  "items": [
    {
      "TerritoryCode": "AE",
      "AlternateTerritoryCode": "ar-AE",
      "TerritoryShortName": "United Arab Emirates",
      "CurrencyCode": "AED"
    },
    ...
  ],
  "count": 25,
  "hasMore": false,
  "limit": 25,
  "offset": 0,
}
```

The ServiceDataProvider variables for the fetchFirst and fetchByKeys capabilities will be configured as follows:

```html
sample-page.html

"territoriesSDPVar": {
  "type": "vb/ServiceDataProvider",
  "defaultValue": {
    "endpoint": "fa-crm-service/getTerritories",
    "idAttribute": "TerritoryCode",
    "itemsPath": "items",
    "uriParameters": {
      "finder": "EnabledFlagFinder;BindEnabledFlag=Y"
    }
  }
}
```

A finder query parameter is applied to all queries going against the endpoint.

When no capabilities are set the Service Data Provider variable is assumed to support a fetchFirst capability.


```json
  "territoriesByCodesSDPVar": {
    "type": "vb/ServiceDataProvider",
    "defaultValue": {
      "endpoint": "fa-crm-service/getTerritories",
      "idAttribute": "TerritoryCode",
      "itemsPath": "items",
      "uriParameters": {
        "finder": "EnabledFlagFinder;BindEnabledFlag=Y"
      },
      "capabilities": {
        "fetchByKeys": {
          "implementation": "lookup"
        }
      },
      "mergeTransformOptions": "{{ $functions.fixupTransformOptions }}"
    }
  }
```

- Line 10: the 'capabilities' property is set to implementation "lookup"
- Line 13: the 'mergeTransformOptions' property is bound to a method that can be used to fix up keys to map to filter options
  - when the fetchByKeys() method is called with the 'keys' parameter, then configuration parameters provided to the 'mergeTransformOptions' function can be used to lookup the key values, to set on the 'filter' transform options.
  - the filter transform function can then generate the "q=" query param using the filter transform options.
    Example &q=TerritoryCode IN ('AE', 'AD', 'US')
- Note: A filter request transforms mapping is not shown here.
The code on the right is the function referenced by the 'mergeTransformOptions' property of the Service Data Provider.

- Line 8 implements a query transform for a RAMP endpoint to handle the keys that might need to be appended as a query param.

Configuring the MultiServiceDataProvider Variable

A page author can configure a variable of type vb/MultiServiceDataProvider, with two dataProviders.
### sample-page.html

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>&quot;territoriesMultiSDP&quot;:</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>&quot;type&quot;: &quot;vb/MultiServiceDataProvider&quot;,</code></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><code>&quot;defaultValue&quot;: {</code></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>&quot;dataProviders&quot;: {</code></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><code>&quot;fetchFirst&quot;:</code></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><code>&quot;fetchByKeys&quot;:</code></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><code>&quot;{{ $page.variables.territoriesSDPVar }}&quot;</code></td>
<td>Line 5: 'fetchFirst' configuration, has the 'variable' property set to the Service Data Provider variable - territoriesSDPVar</td>
</tr>
<tr>
<td>8</td>
<td><code>&quot;{{ $page.variables.territoriesByCodesSDPVar }}&quot;</code></td>
<td>Line 6: 'fetchByKeys' configuration, has the 'variable' property set to the same Service Data Provider variable - territoriesByCodesSDPVar</td>
</tr>
<tr>
<td>9</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

### Configuring a JET Combo/Select in Design Time

- Line 1: the value is bound to a variable that is an array of selected TerritoryCode keys
- Line 2: the options attribute is bound to the multiSDP variable

```html
<oj-combobox-many id="so11" value="{{ $variables.selectedTerritories }}" options="[[ $variables.territoriesMultiSDP ]]"
  options-keys.label="[[ "TerritoryShortName" ]]
  options-keys.value="[[ "TerritoryCode" ]]
</oj-combobox-many>
```

### When a service provides a fetchByKeys capability, andDataProvider.containsKeys is called

The containsKeys() method can be called by components bound to an Service Data Provider variable that supports the 'fetchByKeys' capability.

### Built-in Array Data Provider

A data provider that can be bound to collection components.

This built-in type is a data provider implementation where the data is static. A static source of data does not change frequently and only allows infrequent adds/updates and removes. This data provider can be bound to collection components such as listView and table components. Operations on the data, such as sorts, adds, removes, or updates are managed by the vb/ArrayDataProvider itself. This is different than the vb/ServiceDataProvider, where all operations generally are processed in the back end via REST calls.

A variable of this type is generally defined on the page, using the built-in type `vb/ArrayDataProvider`. 
vb/ArrayDataProvider

{
 "pageModelVersion": "0.9.9",

 "variables": {
 "productListADP": {
 "type": "vb/ArrayDataProvider",
 "defaultValue": {
 "itemType": "application:productSummary"
 }
 }
 }
}

The Array Data Provider has several properties available.

data
The static array of data that the Array Data Provider wraps. The data property is set once when the page or component loads. The implicitSort criteria that the data is pre-sorted with is also set once the page or component loads.

idAttribute
A string or array that is the field or fields in the response data for each row, that represents the 'id' or key field. Can be one of:

- a property name - the key, in various contexts, will also be a string.
- an array of property names - the key will also be an array, of values.
- @value, use all properties - the key will also be an array, of values.
- @index use the index as the key - the key will be an integer.

implicitSort
The implicit sort criteria by which the data is pre-sorted. This is an array of objects, where each object is an atomic sort expression of the form:

{
 "attribute": "<name of the field>",
 "direction": "<'ascending' (default) or 'descending'>"
}

itemType
The type of each item in the data array. This is usually a string that points to an application type or to a definition.

The Array Data Provider provides a sort feature:

- {capabilityName: 'full', attributes: 'multiple} means the endpoint has support for sorting results by one or more fields.
- null means the endpoint has no support for sorting

Events
vb/ADP raises events to notify components listening to changes in the data provider they are bound to. The events currently supported by any iterating data providers are the ‘add’, ‘remove’, ‘update’ and ‘refresh’ events.

To make it easier for users to force a variable of type vb/ADP to dispatch the above events, a new action - fireDataProviderEventAction - has been introduced. When the mutation events ‘add’, ‘remove’ and ‘update’ are called, the vb/ArrayDataProvider implementation will automatically mutate the underlying data. Users are not required to mutate the data prior to raising this event, using an assignVariables action.

Variable Events

All variables including vb/ADP raise the variable onValueChanged event when any of its properties change. ADP in particular will detect if its data has changed and will fire a DataProviderEvent notifying subscribers of the change. Typically, these would be components that are bound to the ADP variable and have registered a listener.

Assigning Data

The data property of the vb/ADP variable is set once, when the page or component loads. The implicitSort criteria that the data is pre-sorted with is also set once the page or component loads.

After the initial load, a page author can mutate the data by directly manipulating the data array using the assignVariablesAction action. Typically, the mutation to the data is triggered by the UI or some other app logic. In either circumstance, the ADP data needs to be manually updated. When the data property mutates, ADP automatically detects the change and notifies all listeners/components of the change, so that they can re-render. If the data is mutated directly, it’s not required to use the fireDataProviderEvent action with the ADP.

Example 8-8   Where the data refers to a constant

Here the ADP variable productADP1 gets its initial data from a constant, **productsConstant**. The ADP data array is initialized with one item.

```
"constants": {
    "productsConstant": {
        "type": "ProductType[]",
        "defaultValue": [{
            "Amount": 10,
            "CurrencyCode": "USD",
            "Quantity": 1,
            "RegisteredPrice": 10,
            "Type": "Constant",
            "Product": "Product-C1",
            "id": 10
        }]
    }
},

"productsADP1": {
    "type": "vb/ArrayDataProvider",
    "description": "mutations on data has to be done directly to the 'data' property",
    "defaultValue": {
        "data": "{{ $page.constants.productsConstant }}",
        "itemType": "ProductType",
        "idAttribute": "id"
    }
}
```
In order to add a new item to the above ADP data you can use an *assignVariablesAction*:

- Line 12: action that generates a new product item
- Line 22: assigns a new array with the new item appended to the existing data

It is currently not possible to add to a specific index of the array using *assignVariablesAction*, when the array references a constants expression.

```json
1 "addProductsADP1": {
2  "description": "adds the generated product to the end",
3  "variables": {
4    "detail": {
5      "required": true,
6      "type": "any",
7      "input": "fromCaller"
8    }
9  },
10  "root": "generateNewProduct",
11  "actions": {
12    "generateNewProduct": {
13      "module": "vb/action/builtin/callModuleFunctionAction",
14      "parameters": {
15        "module": "{{ $page.functions }}",
16        "functionName": "generateNewProduct"
17      },
18      "outcomes": {
19        "success": "assignToADPData"
20      }
21    },
22    "assignToADPData": {
23      "module": "vb/action/builtin/assignVariablesAction",
24      "parameters": {
25        "$page.variables.productsADP1.data": {
26          "source": "{{ $page.variables.productsADP1.data.concat([\$chain.results.generateNewProduct]) }}",
27          "reset": "empty"
28        }
29      }
30    }
31  }
32 }
```

**Example 8-9 Where the data refers to another variable**

In this example the ADP variable `productADP2` gets its initial data from the variable `products`. The ADP data array is initialized with one item.

```json
"variables": {
  "products": {
```
In order to update an item of the above ADP data you can use an `assignVariablesAction`:

- Line 5: action chain gets the updated product item
- Line 22: assign a new array to `productsADP2` with the updated product

```javascript
updateProductsADP2: {
  root: "assignToADPData",
  description: "",
  variables: {
    updatedProduct: {
      type: "page:ProductType",
      required: true,
      input: "fromCaller"
    },
    key: {
      type: "number",
      required: true,
      input: "fromCaller"
    }
  },
  actions: {
    assignToADPData: {
      module: "vb/action/builtin/assignVariablesAction",
      description: "assigning to specific item in ADP.data is not possible - BUFP-27240, so we replace entire array",
      parameters: {
        $page.variables.productsADP2.data: {
          source: "{{ $page.variables.productsADP2.data.map(p => (p.id === $chain.variables.key ? $chain.variables.updatedProduct : p)) }}",
          reset: "empty"
        }
      }
    }
  }
}
```
Example 8-10  Where the data is literally inlined

In this example the ADP variable productADP3 has its initial data inlined.

```
"variables": {
  "productsADP3": {
    "type": "vb/ArrayDataProvider",
    "description": "any mutations are done on ‘data’ property directly",
    "defaultValue": {
      "itemType": "ProductType",
      "idAttribute": "id",
      "data": [{
        "Amount": 30,
        "CurrencyCode": "USD",
        "Quantity": 3,
        "RegisteredPrice": 30,
        "Type": "Literal",
        "Product": "Product-Literal",
        "id": 30
      }]
    }
  }
}
```

In order to remove an item from the above ADP data you can use an `assignVariablesAction`. Line 16 filters the data array of productsADP3 by removing the item with the matching key.

```
1  "removeProductsADP3": {
2    "root": "removeFromProductsADP3",
3    "description": "",
4    "variables": {
5      "key": {
6        "type": "number",
7        "required": true,
8        "input": "fromCaller"
9      }
10    },
11    "actions": {
12      "removeFromProductsADP3": {
13        "module": "vb/action/builtin/assignVariablesAction",
14        "description": "splice returns the removed item, so filter is used instead, which mutates and returns the original array",
15        "parameters": {
16          "$page.variables.productsADP3.data": {
17            "source": "(( $page.variables.productsADP3.data.filter((p) => p.id !== $chain.variables.key) ))",
18            "reset": "empty",
19          }
20        }
21      }
22    }
23  }
```
When the data property is a literal value, to add or update items to the array it is possible to assign to a specific item of the array:

- Line 1: shows an example action where the product is updated directly
- Line 12: shows an example action where the new product is added to the tail end of the data array

```
1 "updateProductsADP3": {
2   "module": "vb/action/builtin/assignVariablesAction",
3   "description": "directly updating ADP.data item is possible when data has no expression",
4   "parameters": {
5     "$page.variables.productsADP3.data[\$page.variables.productsADP3.data.findIndex(p => p.id === $chain.variables.key)]: {
6       "source": "{{ $chain.variables.product }}",
7       "auto": "always",
8       "reset": "empty"
9     }
10   }
11 }
12 "addToProductsADP3Tail": {
13   "module": "vb/action/builtin/assignVariablesAction",
14   "parameters": {
15     "$page.variables.productsADP3.data[\$page.variables.productsADP3.data.length]: {
16       "source": "{{ $chain.results.generateNewProduct }}"
17     }
18   }
19 }
```

Default Values

Variables may have default values.

To specify a default value:

```
"nameOfVariable": {
  "type": "string",
  "defaultValue": "someString"
},
"someOtherVariable": {
  "type": "boolean",
  "defaultValue": true
},
"yetAnotherVariable": {
  "type": "number",
```
Object variables can also have default values:

```json
"nameOfVariable": {
  "type": {
    "foo": "string",
    "bar": "number"
  },
  "defaultValue": {
    "foo": "myDefaultFoo"
  }
}
```

Object variables that reference an application type can also have a default value for their properties:

```json
"nameOfVariable": {
  "type": "application:myType",
  "defaultValue": {
    "foo": "myDefaultValue"
  }
}
```

Arrays can also have a default value for their properties:

```json
"nameOfVariable": {
  "type": "application:myArrType",
  "defaultValue": {
    "foo": "myDefaultValue"
  }
}
```

Expressions in Default Values

Default values may contain expressions.

When a default value contains an expression, note that expressions can also use other variables. You can reference a variable with the following syntax:

<table>
<thead>
<tr>
<th>Scope</th>
<th>Variable Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>$application.variables.&lt;variableName&gt;</td>
</tr>
<tr>
<td>Page</td>
<td>$page.variables.&lt;variableName&gt;</td>
</tr>
<tr>
<td>Action Chain</td>
<td>$chain.variables.&lt;variableName&gt;</td>
</tr>
</tbody>
</table>

Expressions must be wrapped in expression syntax `{{ expr }}`, and the expression must be the entire value. Expressions can also call external functions via the page function module.

To reference another variable in a default value, you can do the following:
"nameOfVariable": {
    "type": "application:myType",
    "defaultValue": {
        "foo": "{{ $application.variables.someOtherVariable }}"
    }
}

Since these are expressions, you can also add simple Javascript code to the values:

"myOtherVariable": {
    "type": {
        "someBoolProperty": "boolean"
    },
    "defaultValue": {
        "someBoolProperty": {{ $application.variables.someOtherVariable === true }}
    }
}

Input Variables

Variables can also be inputs to the page.

There are two types of input. The first consists of inputs that come from the URL. The second type consists of inputs that are passed internally by the framework. To mark a variable as an input, you can use the following properties:

"nameOfVariable": {
    "type": "string",
    "input": "fromCaller/fromUrl",
    "required": true
}

Here the input is either "fromCaller" or "fromUrl". If it is `fromCaller`, it will be passed internally. If it is fromURL, it will be passed via the URL. If the "required" property is true, the variable value will be required to be passed during a navigation or page load.

The implicit object $parameters is used to retrieve the input parameter values inside the vbBeforeEnter event handler. Input variables do not exist until the vbEnter event.

The input regionName is retrieved using $parameters.regionName in the vbBeforeEnter handler and $page.variables.regionName in the enter handler.

"eventListeners": {
    "vbBeforeEnter": {
        "chains": [
        {
            "chainId": "checkForRegionName",
            "parameters": {
                "regionName": "{{ $parameters.regionName }}"
            }
        },
        
        "vbEnter": {
            "chains": [
            {
                "chainId": "initializeVariables",
                "parameters": {
                    "regionName": "{{ $page.variables.regionName }}",
                    "facilityId": "{{ $page.variables.facilityId }}"
                }
            }
        ]
    }
}
Persisted Variables

The value of a variable can be persisted on the history, for the current session or across sessions.

To remove a variable from the storage, sets its value to null.

**Example 8-14   Using a Persisted Variable**

```json
"variables": {
  "sessionToken": {
    "type": "string",
    "persisted": "session"
  }
}
```

**Actions**

A list of built-in actions available in Oracle Visual Builder for applications

---

**Note:**

Action definitions minimally have a "module" property that specifies the action implementation. Actions can also have an optional "label" property, which is user-friendly.

**Topics:**

- Assign Variables
- Call Component Method
- Call Chain Action
- Call Module Function
- Fire Data Provider Event Action
- Call Module Function
- ForEach Action
- Fork Action
- If Action
- Login Action
- Logout Action
- Navigate Back Action
- Navigate to Page Action
- Open URL Action
Assign Variables

This action is used to assign values to a set of variables.

This action has two forms. The first is metadata-driven, where you can specify how assignment should be performed by using metadata. The second supports calling out to a custom assign variable function. This custom assign variable function can perform a transformation on the source value before assignment.

```
"myActionChain": {
  "root": "myAssignVariableAction",
  "actions": {
    "myAssignVariableAction": {
      "module": "vb/action/builtin/assignVariablesAction",
      "parameters": {
        "$page.variables.target1": { "source": "{{ $page.variables.source1 }}" },
        "$page.variables.target2": { "source": "{{ $page.variables.source2 }}" }
      }
    }
  }
}
```

Topics:
- Metadata-driven Variable Assignment
- Custom Assign Variable Function

Metadata-Driven Variable Assignment

This action is used to assign values to a set of variables using metadata.

Metadata-driven variable assignment lets you use metadata to specify how assignment should be performed.

This form takes a map of target expression and assignment metadata pairs. For example, if the target expression is a structure, it has to resolve to a variable or to a variable's property. The target expression has to be prefixed with one of the following:

- `$application.variables`
- `$page.variables`
- `$chain.variables`
- `$variables`

This should be followed by a variable name or a path to a variable property, such as the following:

- `application.variables.a`
• $page.variables.a.b
• $variables.a.b.c

Note that $variables.a.b.c is a shortened form of $chain.variables.a.b.c

The expression can be arbitrarily complex as long as it is a valid JavaScript expression and satisfies the above constraints.

The assignment metadata has the following format:

```
{
    "source": "some expression",
    "reset": "none", // default to "toDefault"
    "mapping": { ... }
}
```

The "source" expression can be an arbitrary expression that evaluates to a primitive value, an object or an array.

The "reset" option can be one of the following:

• "toDefault" - reset the target to its default value before assignment.
• "empty" - clear the target before assignment. If the target has an object type, the target will be reset to an empty object of that type. If the target is an array, the target will be reset to an empty array.
• "none" - overwrite the existing target value

The "mapping" is a piece of metadata used to provide fine-grained control over what gets assigned from the source to the target. When no "mapping" is used to control the assignment, there are two possible schemes for assignment depending on the target type, auto and direct.

**Auto Assign Source to Target**

If the target has a concrete type, the assign action will auto-assign the source to the target. If the target type is an object type, auto-assignment will recursively assign each property in the source object to the corresponding property in the target object based on the target type. If the target is an array, the source will be treated as an array if it is not one already. For each item of the source array, an empty item will be created using the target's array item type and appended to the target array. The source item is then auto-assigned to the target item.

If the target property is an object and the source property is a primitive or vice versa, no assignment will be made. For primitive types, the source value will be coerced into the target type before assignment. For boolean type, the coercion will be based on whether the source value is truthy except for "false" (case-insensitive) and "0" which will be coerced to false.

**Direct Assign Source to Target**

If the target has a wildcard type, e.g., any, any[], object or object[], direct assignment will be performed. The behavior may differ depending on the wildcard type:

• any - the source value is directly assigned to the target
• any[] - the source value is turned into an array if not one already and then directly assigned to the target
• object - same as any except the source value has to be an object. Otherwise, no assignment is performed.
• object[] - same as any[] except the items in the source array have to be objects. Otherwise, no assignment is performed.

**Example**: metadata-driven assignment takes a map of target expression and assignment metadata pairs.

```json
"myActionChain": {
    "root": "myAssignVariableAction",
    "actions": {
        "myAssignVariableAction": {
            "module": "vb/action/builtin/assignVariablesAction",
            "parameters": {
                "$page.variables.target1": { "source": "{{ $page.variables.source1 }}" },
                "$page.variables.target2": { "source": "{{ $page.variables.source2 }}" }
            }
        }
    }
}
```

**Example**

```json
"$page.variables.target": {
    "source": "{{ $page.variables.source }}",
    "mapping": {
        "$target.a": "$source.b",
        "$target.b.c": "$source.c.b"
    }
}
```

**Example**

```json
"$page.variables.target": {
    "source": "{{ $page.variables.source }}",
    "mapping": {
        "$target.a": "$source.b",
        "$target.b": {
            "source": "$source.c"
            "mapping": {
                "$target.c": "$source.b"
            }
        }
    }
}
```

### Assign Variables With a Custom Function

This action uses a custom function to assign values to a set of variables.

A custom assign variable function can perform a transformation on the source value before assignment.

The AssignVariablesAction will first look up the function referenced by "functionName" from the page's functions module and call it with the current available scopes. It will then assign the return value of the function call to the target variable. The custom function should have the following signature:

```javascript
PageModule.prototype.myAssignVariableFunction = function (helper, targetDefaultValue)
```

The "targetDefaultValue" is the default value for the target which can be used to emulate the "toDefault" reset option.
The "helper" is an utility object that can be used to retrieve values for variables within
the current scope and perform auto-assignment. It has the following interface:

class AssignmentHelper {
    /**
    * Gets a variable from a scope by its string representation, e.g.,
    * helper.get("$page.variables.myVar")
    */
    get(expr);

    /**
    * Assigns properties from one or more sources to the target if and
    * only if the property already exists on the target. The sources
    * are processed in the order they are defined.
    * If target is null, any empty target value will be created based
    * on the target's type. If the target is not null, it will be cloned
    * and the sources will be assigned into the clone. In either case,
    * this value will be returned as the result.
    */
    pick(target, ...sources) {
    }
}

**Example**: an assign variable function that resets the target value to its default value and auto-assign the source to the target:

PageModule.prototype.myAssignVariableFunction = function (helper, targetDefaultValue) {
    var source = helper.get("$page.variables.source");
    var result = helper.pick(targetDefaultValue, source); return result;
}

## Call Component Method

The action module for this action is "vb/action/builtin/callComponentMethodAction". This provides a declarative way to call methods on JET components.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>The actual page component. Utility methods are provided in the $page scope to get JET components:</td>
</tr>
<tr>
<td></td>
<td>$page.components.byId('myCard')</td>
</tr>
<tr>
<td></td>
<td>$page.components.bySelector('#myCompId')</td>
</tr>
<tr>
<td>method</td>
<td>The name of the component method to call.</td>
</tr>
</tbody>
</table>

**Note:**

These methods will return `null` if no element is found, or if the element is not part of a JET component.
For this sample composite component, the 'flipCard' method takes two parameters: 'model' which is unused (null below), and 'event', which we construct with a 'type' property:

```
"myActionChain": {
   "root": "myAction",
   "actions": {
      "flipCardMethodCall": {
         "label": "Flip the Card",
         "module": "vb/action/builtin/callComponentMethodAction",
         "parameters": {
            "component": "{{ $page.components.byId('myCard') }}",
            "method": "flipCard",
            "params": ["{{ null }}", { "type": "click" }]
         }
      }
   }
}
```

### Call Chain Action

The action module for this action is "vb/action/builtin/callChainAction".

To call an action chain, you need to pass the following parameters:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| id             | The ID of the action chain to call. Application action chains need to be prefixed with "application:"
| params         | An expression that maps to an array of parameters. |

### Call Module Function

Function and function payloads.

Call module parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>The module to call the function on. At the moment, this could be &quot;$page.functions&quot; or &quot;$application.functions&quot;.</td>
</tr>
<tr>
<td>functionName</td>
<td>The name of the function to call.</td>
</tr>
<tr>
<td>params</td>
<td>An array of parameters. Note that a single parameter must still be passed as a single item array.</td>
</tr>
</tbody>
</table>
The outcome is either 'success' if the function call was successful, or 'error' otherwise. The result payload is equivalent to whatever the function returns (which may be undefined if there is no return). If the function returns a promise, the result payload will be whatever is resolved in the promise.

If there is a function defined in the page functions module as follows:

```javascript
PageModule.prototype.sum = function(one, two) {
    return one + two;
}
```

You can call that function with the following action:

```javascript
"myActionChain": {
    "root": "myAction",
    "actions": {
        "myAction": {
            "label": "call my sum function",
            "module": "vb/action/builtin/callModuleFunction",
            "parameters": {
                "module": "{{$page.functions}}",
                "functionName": "sum",
                "params": ["3", "4"]
            }
        }
    }
}
```

After this action call, $chain.results.mySumAction should be set to 7.

**Call REST Action**

The action module for this action is "vb/action/builtin/restAction".

The call REST action is used to make a REST call in conjunction with the service definitions. Internally, this action uses the Rest Helper, which is a public utility. Its parameters are as follows:

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpoint</td>
<td>The endpoint ID as defined in the service configuration.</td>
</tr>
<tr>
<td>uriParams</td>
<td>A key/value pair map that will be used to override path and query parameters as defined in the service endpoint.</td>
</tr>
<tr>
<td>body</td>
<td>A structured object that will be sent as the body.</td>
</tr>
</tbody>
</table>
Parameter Name | Description
--- | ---
requestType | The content-type of the request, either ‘json’, ‘form’, or ‘url’.

**Note:**
Note that this is deprecated. Instead, use ‘contentType’ and ‘fileType’.

headers | An object; each property name is a header name and value that will be sent with the request.

contentType | An optional string value with an actual MIME type, which will be used for the “content-type” header. When used with "fileType", this is also used as the type for the File blob.

responseType | If a responseType is set, coerce the REST response body to this type.

filePath | An optional path to a file to send with the request. If "contentType" is set, that is used as the type for the File contents. If "contentType" is not set, a lookup of common file extensions will be used.

fileContentType | An optional string, used in combination with "contentType", “multipart/form-data”, and “filePath”.

hookHandler | Used primarily by vb/ServiceDataProvider when externalizing data fetches. See ServiceDataProvider for details.

requestTransformOptions | A map of values to pass to the corresponding transform, as the “options” parameter.

requestTransformFunctions | A map of named transform functions, called before making the request, where the function is: fn(configuration, options)

responseTransformFunctions | A map of named transform functions, called before making the response, where the function is: fn(configuration, options)

responseBodyFormat | A string that allows an override of the standard Rest behavior which looks for a “content-type” header to determine how to read and parse the response. Possible values are “text”, “json”, “blob”, “arrayBuffer”, ”base64”, ”base64Url”, and “formData”.

**Using multipart/form-data**
If you have set "contentType" to "multipart/form-data", the Action will interpret your request “body” object as the form parts. Each property of the body object will be a form part. If “filePath” is also set, it will be added as an additional part using the lookup of common file extension types.
If “filePath” is also set, it will be added as an additional part using the sample simple file extension type association. The name of this part will be “file”, or can be specified using “filePartName”.

You may optionally override the file type by using “fileContentType” for the file part.

**Transforms**

The requestTransformOptions, requestTransformFunctions, and responseTransformFunctions can be used to modify the request and response. Some built-in service endpoints have transform functions for 'sort', 'filter', 'paginate', 'select'. Options for these transform functions can be defined using the same name via the requestTransformOptions property. For third party services, the options set are based on the type of transform functions supported.

When using the Rest Action the transforms names have no semantic meaning, and all request and response transforms are called.

**Table 8-1 Request and Response Transforms**

<table>
<thead>
<tr>
<th>Transform type</th>
<th>Parameters</th>
<th>Return Value</th>
</tr>
</thead>
</table>
| Request       | */**       | Configuration object; see "Parameters".  
|               | configuration: {  
|               |   url:  
|               |     initConfig: {  
|               |     method: // string with http method  
|               |     body: // request body, if any  
|               |     credentials: // string see (fetch)  
|               |     headers: // object, map of strings  
|               |   }  
|               | },  
|               | },  
|               | options: provided by the application  
|               | },  
|               | context: an empty object, which exists for the lifetime of one REST call, a set of transforms share this.  
|               | **/  
|               | mytransform(configuration, options, context) |
Table 8-1  (Cont.) Request and Response Transforms

<table>
<thead>
<tr>
<th>Transform type</th>
<th>Parameters</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>/* response: { body, headers }</td>
<td>The return value is application-defined. The value is returned as the 'transformResults' of the REST call result:</td>
</tr>
<tr>
<td></td>
<td>* context: an empty object, see &quot;Request transforms&quot;</td>
<td>/*</td>
</tr>
<tr>
<td></td>
<td>*/</td>
<td>myresponsetransform(response, context);</td>
</tr>
</tbody>
</table>

Table 8-2  Outcomes

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Description</th>
<th>Result Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>If the response code is within the 200 range (or 'ok' in fetch API terms).</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>• status: number</td>
<td>status: &lt;responseCode&gt;,</td>
</tr>
<tr>
<td></td>
<td>• headers: Headers object</td>
<td>headers:</td>
</tr>
<tr>
<td></td>
<td>• body: the result of the call (scalar, object, array, etc.)</td>
<td>&lt;responseHeaders&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>body: &lt;result body&gt;</td>
</tr>
</tbody>
</table>
Table 8-2  (Cont.) Outcomes

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Description</th>
<th>Result Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>failure</td>
<td>If the response code is outside of the 200 range (an error response).</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>• message</td>
<td>message: {</td>
</tr>
<tr>
<td></td>
<td>• summary: string</td>
<td>summary: &lt;rt message&gt;</td>
</tr>
<tr>
<td></td>
<td>• error: Error object, or null</td>
<td>error: &lt;Error, or null&gt;,</td>
</tr>
<tr>
<td></td>
<td>• payload</td>
<td>payload: {</td>
</tr>
<tr>
<td></td>
<td>• status: number</td>
<td>status: &lt;responseCode&gt;,</td>
</tr>
<tr>
<td></td>
<td>• headers: Headers object</td>
<td>headers: &lt;responseHeaders&gt;,</td>
</tr>
<tr>
<td></td>
<td>• body: the result of the call (scalar, object, array, etc).</td>
<td>body: &lt;result body&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Example 8-15  Defining Services

In order to use a REST API, it should be first defined.

In this example, the following endpoint is registered for the 'foo' service:

```json
{
    "swagger": "2.0",
    "info": {
        "version": "1.1",
        "title": "ifixitfast",
        "description": "FIF",
    },
    "host": "domain.com",
    "basePath": "/services/root",
    "schemes": [
        "http"
    ],
    "paths": {
        "/foo/{id}": {
            "get": {
                "summary": "get a specific Foo object",
                "operationId": "getBar",
                "parameters": [
                    {
                        "name": "id",
                        "in": "path",
                        "required": true,
                        "type": "string"
                    }
                ],
                "responses": {
                    "200": {
                        "description": "",
                        "schema": {}                
                    }
                }
            }
        }
    }
}
```
You can invoke that endpoint with the following, passing in a value for the 'id' path parameter from a page parameter:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/restAction",
      "parameters": {
        "endpoint": "foo/getBar",
        "uriParams": {
          "id": "{{ $page.variables.myId }}"
        }
      }
    }
  }
}
```

**Example 8-16  Declaring Services in the Application**

Service definitions are referenced in declarations in the application or in flows. The service name and path are defined by a "services" section in an app-flo.json or xxx-flow.json model. Service declarations support two syntaxes: a string (path), or an object with "path" and "headers":

```
"services": {
  "fooService": "/demo-data-service.json",
  "barService": {
    "path": "/service-def.json",
    "headers": {
      "Accept": "application/vnd.oracle.openapi3+json"
    }
  }
}
```

**Example 8-17  Referencing Endpoints**

You can invoke the endpoint with the following, passing a value for the 'id' path parameter from a page parameter:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/restAction",
      "parameters": {
        "endpoint": "fooService/getBar",
        "uriParams": {
          "id": "{{ $page.variables.myId }}"
        }
      }
    }
  }
}
```

**Example 8-18  A Simple Transform Function**

One request transform function, and one response transform function for a third party service or endpoint, might look like this example. Here, the transform functions are defined in the page module and are configured on the RestAction directly. More
commonly, transform functions are defined in the service definition and do not need to be mapped on the RestAction.

"fetchIncidentList": {
  "module": "vb/action/builtin/restAction",
  "parameters": {
    "endpoint": "ifixitfast-service/getIncidents",
    "requestTransformOptions": {
      "sort": "{{ $page.variables.sortExpression }}",
    },
    "requestTransformFunctions": {
      "sort": "{{ $page.functions.sort }}"
    },
    "responseTransformFunctions": {
      "paginate": "{{ $page.functions.paginateResponse }}"
    },
  },
  "outcomes": {
    "success": "returnSuccessResponse",
    "failure": "returnFailureResponse"
  }
},

The corresponding module functions would be:

PageModule.prototype.sort = function (configuration, options) {
  /// some code here to modify 'configuration'
  return configuration;
}

PageModule.prototype.paginateResponse = function (configuration) {
  /// some code here to modify 'configuration'
  return configuration;
}

Fire Data Provider Event Action

The action module for this action is "vb/action/builtin/fireDataProviderEventAction"

This dispatches an event on a data provider of type oj.IteratingDataProvider. The action can be called either with a mutation event or a refresh but not both. A mutation event can include multiple mutation operations as long as the id values between operations do not intersect. This behavior is enforced by JET components. For example, you cannot add a record and remove it in the same event, because the order of operations cannot be guaranteed.

The action can return either success or failure. Success returns null, while failure returns the error string.

### Table 8-3 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>string</td>
<td>Target of the event, usually a variable of type vb/SDP or vb/ADP</td>
<td>target: &quot;{{ $page.variables.incidentList }}&quot;</td>
</tr>
</tbody>
</table>
### Table 8-3  (Cont.) Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh</td>
<td>null</td>
<td>Indicates a data provider refresh event needs to be dispatched to the data provider identified by the target. A null value is specified because the refresh event does not require a payload.</td>
<td>refresh: null</td>
</tr>
<tr>
<td>add</td>
<td>object</td>
<td>The following properties may be present in the payload:</td>
<td><em>add</em>: {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>data</strong>: Array&lt;Object&gt;; the results of the 'add' operation. Note there can be more than one rows added. If data alone is present in the payload, and the target has an idAttribute specified then the 'keys' is built for you. The structure of the data returned must be similar to the responseType specified on the target variable of type vb/ServiceDataProvider (respecting the &quot;itemsPath&quot;, if any), or the itemType specified on the vb/ArrayDataProvider</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>keys</strong>: optional Set&lt;&gt;., the keys for the rows that were added. If SDP variable is configured with idAttribute this can be determined by SDP itself from the data, if data is present.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>metadata</strong>: optional Array&lt;ItemMetadata&lt;Object&gt;&gt;. Since the SDP variable is configured with 'idAttribute', this can be determined by SDP itself.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>afterKeys</strong>: optional Set&lt;&gt;; a Set that is the keys of items located after the items involved in the operation. If null and index not specified then insert at the end.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>indexes</strong>: optional Array&lt;number&gt;, identifying insertion point.</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
<td>The payload for the remove event is similar to add above except 'afterKeys' is not present.</td>
<td><em>remove</em>: {</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>keys</em>:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;{{ [ $page.variables.productId ] }}&quot;</td>
</tr>
</tbody>
</table>

An example with ServiceDataProvider, where "itemsPath": "items":

```json
"updateList": {
  "module": "vb/action/builtin/fireDataProviderEventAction",
  "parameters": {
    "target": "{{ $page.variables.personList }}",
    "add": {
      "data": {
        "items": "{{ $chain.results.createPersonPost.body }}"
      }
    }
  }
}
```

An example with ServiceDataProvider, where "itemsPath": "items":

```json
"remove": {
  "keys": "{{ $page.variables.productId }}"
}
```
Table 8-3  (Cont.) Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>Same as</td>
<td>update.</td>
<td>&quot;update&quot;: { &quot;data&quot;: &quot;{{ $page.variables.currentIncidentResponse }}&quot; }</td>
</tr>
</tbody>
</table>

Example 8-19  Example 1

Configuring a refresh event to be dispatched to a ServiceDataProvider:

(1) activityListDataProvider is the name of the page variable that is of type vb/ServiceDataProvider
(2) refresh has a null value

```
"fireDataProviderRefreshEventActionChain": {
  "variables": {
    "payload": {
      "type": {
        "target": "activityListDataProvider"  "fireDataProviderRefreshEventActionChain": {
                                                  (1)                                                 |
    }
  }
},
"root": "fireEventOnDataProvider",
"actions": {
  "fireEventOnDataProvider": {
    "module": "vb/action/builtin/fireDataProviderEventAction",
    "parameters": {
      "target": "$page.variables.$variables.payload.target  "fireDataProviderRefreshEventActionChain": {
                                                (2)                                                 |
      "refresh": null                                               |
    }
  }
},
},
```

Example 8-20  Example 2

Configuring a remove event to be dispatched to a ArrayDataProvider:

(1) deleteProductChain deletes a product and ends up calling another chain that fires a remove event on an ArrayDataProvider
(2) calls fireDataProviderEventAction
(3) on a variable of type vb/ArrayDataProvider
(4) with a remove payload

```
"variables": {
  "variablesDynamic": {
    "type": "vb/ArrayDataProvider",
    "defaultValue": {
      "idAttribute": "id",
      "implicitSort": "{{ $page.variables.productCategoryImplicitSortCriteria }}",
      "itemType": "application:productSummary"
    }
  }
},
```
"chains": {
  "deleteProductChain": {
    "variables": {
      "productId": {
        "type": "string",
        "description": "delete a single product",
        "input": "fromCaller",
        "required": true
      }
    },
    "root": "deleteProduct",
    "actions": {
      "deleteProduct": {
        "module": "vb/action/builtin/restAction",
        "parameters": {
          "endpoint": "ifixitfast-service/deleteProduct",
          "uriParams": {
            "productId": "{{ $page.variables.productId }}"
          }
        },
        "outcomes": {
          "success": "refreshProductList"
        }
      },
      "refreshProductList": {
        "module": "vb/action/builtin/callChainAction",
        "parameters": {
          "id": "fireDataProviderMutationEventActionChain",
          "params": {
            "payload": {
              "remove": {
                "keys": "{{ [ $page.variables.productId ] }}"
              }
            }
          }
        }
      }
    }
  },
  "fireDataProviderMutationEventActionChain": {
    "variables": {
      "payload": {
        "type": "application:dataProviderMutationEventDetail",
        "input": "fromCaller"
      }
    },
    "root": "fireEventOnDataProvider",
    "actions": {
      "fireEventOnDataProvider": {
        "module": "vb/action/builtin/fireDataProviderEventAction",
        "parameters": {
          "target": "{{ $page.variables.productListArrayDataProviderDynamic }}",
          "remove": "{{ $variables.payload.remove }}"
        }
      }
    }
  }
}
ForEach Action

This action lets you execute another action for each item in an array.

The ForEach action takes an 'items' and 'actionId', and adds a $current context variable for the called action, or 'Callee', in order to access the current item. The parameters are as follows:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>An optional alias for $current. Used to name the context so that it can be referenced in nested Callees.</td>
</tr>
<tr>
<td>actionId</td>
<td>An ID in the current action chain.</td>
</tr>
<tr>
<td>items</td>
<td>An expression that evaluates to an array.</td>
</tr>
</tbody>
</table>

By default, the ForEach Action ID in the declaration will be used for the alias to $current. Additional properties are injected into the available contexts that the called action ('callee') can reference in its parameter expressions:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$current.data</td>
<td>The current array item.</td>
</tr>
<tr>
<td>$current.index</td>
<td>The current array index.</td>
</tr>
<tr>
<td>alias.data</td>
<td>An alternate syntax for $current.data, which allows a reference to the $current from nested contexts.</td>
</tr>
<tr>
<td>alias.index</td>
<td>An alternate syntax for $current.index, which allows a reference to the $current from nested contexts.</td>
</tr>
</tbody>
</table>

Note that if an action has an "as" alias, then the value will be used as the alias instead. For example, for as="foo", you can also create expressions that reference "foo.data" and "foo.index".

**Example 8-21  Example 1**

In this example, $current.data and forEachCurrent.data are equivalent.

```json
actions: {
  "forEach": {
    "module": "vb/action/builtin/forEachAction",
    "parameters": {
      "items": "{{ $variables.testArray }}",
      "actionId": "someAction",
      "as": "forEachCurrent",
    },
  },
  "someAction": {
    "module": "someRandomAction",
    "parameters": {
      "outcome": "{{ $current.data.foo }}",
      "payload": {
        "text": "{{ forEachCurrent.data.bar }}",
        "index": "{{ $current.index }}"
      }
    }
  }
}
```
Example 8-22  Example 2

This example demonstrates the use of "as".

```json
"actions": {
    "forEachOuter": {
        "label": "the outer-most action, a ForEach",
        "module": "vb/action/builtin/forEachAction",
        "parameters": {
            "items": ["a", "b"],
            "actionId": "forEachInner"
        }
    },
    "forEachInner": {
        "label": "the inner-most action, a ForEach, called by a ForEach",
        "module": "vb/action/builtin/forEachAction",
        "as": "inner",
        "parameters": {
            "items": [1, 2],
            "actionId": "someAction"
        }
    },
    "someAction": {
        "label": "a custom action",
        "module": "countToTwoAction",
        "parameters": {
            "someParam": "{{ forEachOuter.data }}",
            "anotherParam": "{{ inner.data }}"
        }
    }
}
```

Fork Action

The action module for this action is "vb/action/builtin/forkAction".

This action allows multiple action chain paths to run in parallel, then wait for their responses and produce a combined result. Normally, if you do not care what your action chains return, you can chain multiple action chains on the event handler. If you want to wait for the result, and take action once everything is complete, you can use this action instead.

A fork action has an arbitrary set of actions whose action sub-chains will run in parallel. A special outcome, 'join', will be followed once all the sub-chains complete processing. The outcome of the fork action is always 'join', and the result is a mapping from the outcome id's of the sub-chains to their outcome/result payload.

This action does not have any parameters.

Example 8-23  Example

To make two REST calls, then do some assignments only after they both complete:

```json
"myActionChains": {
    "root": "myAction",
    "actions": {
        "myForkAction": {
            "module": "vb/action/builtin/forkAction",
```
If Action

The action module for this action is "vb/action/builtin/ifAction".

This action will evaluate an expression and return a 'true' outcome if the expression evaluates to true, and a 'false' outcome otherwise.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>The expression to evaluate.</td>
</tr>
</tbody>
</table>

For example:

"myActionChain": {  
  "root": "myAction",  
  "actions": {  
    "myAction": {  
      "module": "vb/action/builtin/ifAction",  
      "parameters": {  
        "condition": "{{ $chain.results.myRestAction.code === 404 }}"  
      },  
      "outcomes": {  
        "true": "...",  
        "false": "..."  
      }  
    }  
  }  
}
Geolocation Action

The action module for this action is "vb/action/builtin/geolocationAction".

This action provides a declarative access to geographical location information associated with the hosting device. This action requires user's consent. As a best practice, it should only be fired on user gesture. Doing so will allow users to more easily associate the system permission prompt for access with the action they just initiated.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximumAge</td>
<td>A positive long value indicating the maximum age in milliseconds of a possible cached position that is acceptable to return. If set to 0, it means that the device cannot use a cached position and must attempt to retrieve the real current position. If set to Infinity, the device must return a cached position regardless of its age.</td>
</tr>
<tr>
<td>timeout</td>
<td>A positive long value representing the maximum length of time, in milliseconds, that the device is allowed to take in order to return a position. The default value is Infinity, meaning that getCurrentPosition() won't return until the position is available.</td>
</tr>
<tr>
<td>enableHighAccuracy</td>
<td>A boolean that indicates the application would like to receive the best possible results. If true, and if the device is able to provide a more accurate position, it will do so. This can result in slower response times or increased power consumption. If false (the default value), the device can save resources by responding more quickly or using less power. On mobile devices, enableHighAccuracy should be set to true in order to use GPS sensors.</td>
</tr>
</tbody>
</table>

If geolocation API is supported in the browser, geolocationAction returns a JSON Position object that represents the position of the device at a given time:
Return Type | Description | Example
--- | --- | ---
Object | The Position interface represents the position of the concerned device at a given time. The position, represented by a Coordinates object, comprehends the 2D position of the device, on a spheroid representing the Earth, but also its altitude and its speed. | Latitude and longitude can be accessed from the Position's coordinates as follows:
```
[[chain.results.getCurrentLocation.coords.latitude ]]
[[chain.results.getCurrentLocation.coords.longitude ]]
```
where `getCurrentLocation` is a geolocationAction.

Latitud and longitude can be accessed from the Position's coordinates as follows:
```
[[chain.results.getCurrentLocation.coords.latitude ]]
[[chain.results.getCurrentLocation.coords.longitude ]]
```
where `getCurrentLocation` is a geolocationAction.

If geolocation is not supported by the browser, or a parameter with a wrong type is detected, a failure outcome is returned. If a PositionError occurs when obtaining geolocation, a failure outcome with PositionError.code payload is returned. Possible PositionError.code values are:

1. PositionError.PERMISSION_DENIED
2. PositionError.POSITION_UNAVAILABLE
3. PositionError.TIMEOUT

For every failure, a descriptive error message can be obtained from the action chain, such as `[[ $chain.results.getCurrentLocation.error.message ]]`.

An example of using geolocation action:
```
"chains": {
  "getCurrentLocation": {
    "root": "geolocation1",
    "description": "",
    "actions": {
      "geolocation1": {
        "module": "vb/action/builtin/geolocationAction",
        "parameters": {
          "timeout": 50000,
          "maximumAge": "{{Infinity}}"
        },
        "outcomes": {
          "failure": "fireNotification1",
          "success": "assignVariables1"
        }
      },
      "fireNotification1": {
        "module": "vb/action/builtin/fireNotificationEventAction",
        "parameters": {
          "summary": "[[ $chain.results.geolocation1.error.message ]]",
          "type": "error",
          "displayMode": "persist"
        }
      }
    }
  }
}
```
Login Action

This action launches the login process as defined in the Security Provider implementation.

The action module for this action is `vb/action/builtin/loginAction`. It invokes the `handleLogin` function on the Security Provider with the `returnPath` argument.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnPath</td>
<td>The path of the page or flow to go to when login is successful.</td>
</tr>
</tbody>
</table>

The behavior of the default implementation of the Security Provider `handleLogin` function is:

- Navigate to the login URL specified by the Security Provider configuration.
- If `returnPath` is not defined, use the default page of the application.
- Convert the page `returnPath` to a URL path and add it to the login URL.

**Example 8-24** Example

An example of a chain using the `loginAction`:

```
"signInChain": {
  "root": "signInAction",
  "actions": {
    "signInAction": {
      "module": "vb/action/builtin/loginAction"
    }
  }
}
```

Logout Action

This action launches the logout process as defined in the Security Provider implementation.

The action module for this action is `vb/action/builtin/logoutAction`. It invokes the `handleLogout` function on the Security Provider with the `logoutUrl` argument.
### Parameter Name | Description
--- | ---
logoutUrl | The URL to navigate to in order to log out.

The behavior of the default implementation of the Security Provider handleLogout function is:

- Navigate to the URL defined by the logoutURL parameter.
- If the logoutUrl parameter is not defined, uses the logout URL of the Security Provider configuration.

**Example 8-25  Example**

An example of a chain using the logoutAction:

```json
"logoutChain": {
  "root": "logout",
  "actions": {
    "logout": {
      "module": "vb/action/builtin/logoutAction"
    }
  }
}
```

**Navigate Back Action**

The action module for this action is "vb/action/builtin/navigateBackAction".

This action will go back one step in browser history. It has a single 'success' outcome and can return a payload by specifying values for the input parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>params</td>
<td>An optional key/value pair map that will be used to pass parameters to a page.</td>
</tr>
</tbody>
</table>

When a parameter is not specified, the original value of the input parameter on the destination page is used. When a parameter is specified, it has precedence over fromUrl parameters.

**Navigate to Page Action**

The action module for this action is "vb/action/builtin/navigateToPageAction".

This action will navigate the user and also perform any parameter passing to activate that page. Parameters for this action are as follows:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>The page path (as defined in the page model) of the page to navigate to (required)</td>
</tr>
<tr>
<td>params</td>
<td>A key/value pair map that will be used to pass parameters to a page (optional)</td>
</tr>
</tbody>
</table>
Parameter Name | Description
---------------|---------------------------------------------
history        | Defines the effect on the browser history. Allowed values are 'replace', 'skip' or 'push'. If the value is 'replace', the current browser history entry is replaced, meaning that back button will not go back to it. If the value is 'skip', the URL is not modified. (optional and default is 'push')

Page parameters are the same as variables that are marked input. When calling this action it doesn't matter if the destination page's variables are passed internally or via the URL. The framework will arrange that the parameters end up in the correct place, and that they are passed to the destination page.

Example 8-26  Example

To navigate to another page:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/navigateToPageAction",
      "parameters": {
        "page": "myOtherPage",
        "params": {
          "id": "{{ $page.variables.myId }}"
        }
      }
    }
  }
}
```

This returns the outcome 'success' if there was no error during navigation. If navigation completed successfully, returns the action result true, otherwise false. Returns the outcome fail with the error in the payload if there was an error.

Open URL Action

The action module for this action is "vb/action/builtin/openUrlAction".

In a web app, this action open the specified URL in the current window or in a new window using the window.open() API. In a mobile app, Cordova's plugin (cordova-plugin-inappbrowser) opens the specified URL.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The url to navigate to (required)</td>
</tr>
<tr>
<td>params</td>
<td>A key/value pair map that will be used as query parameters to the url (optional)</td>
</tr>
<tr>
<td>hash</td>
<td>The hash entry to append to the URL. (optional)</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
**history** | Defines the effect on the browser history. Allowed values are 'replace' or 'push'. If the value is 'replace', the current browser history entry is replaced, meaning that back button will not go back to it. (optional and default is 'push')

**windowName** | A name identifying the window as defined in the window.open() API (optional). If not defined, the URL opens in the current window. Otherwise, refer to the window.open() API documentation. In a mobile app, there are 3 possible values: _self, _blank, or _system. The default is _self. Refer to the documentation for cordova-plugin-inappbrowser.

Once on the URL location, the browser back button will re-enter the last page if you specified a value for the windowName parameter that opens the URL in the current window and the page input parameters will be remembered, even if their type is 'fromCaller'.

**Example 8-27**  
**Example**

To open a URL:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/openUrlAction",
      "parameters": {
        "url": "http://www.oracle.com",
        "params": {
          "id": "{{ $page.variables.myId }}"
        },
        "windowName": "myOtherWindow"
      }
    }
  }
}
```

**Return Action**

The action module for this action is "vb/action/builtin/returnAction".

This action (which should be the terminal action of a chain) allows you to control the outcome and payload of that chain when necessary. Parameters for this action are as follows:

### Parameter Name | Description
--- | ---
**payload** | The payload to return from this action. Useful in a 'callChainAction' to control the resulting payload from calling that action chain. This can be an expression.
An example that uses the return action on a chain that makes a REST call, but returns a simpler value:

```
"myActionChain": {
    "root": "myAction",
    "actions": {
        "someRestCall": {
            "module": "vb/action/builtin/callRestAction",
            "parameters": {...},
            "outcomes": {
                "success": "myReturnAction"
            }
        }
    },
    "myReturnAction": {
        "module": "vb/action/builtin/returnAction",
        "parameters": {
            "outcome": "success",
            "payload": "{{ $chain.results.someRestCall.body.somewhere.in.the.payload.isa.string }}"
        }
    }
}
```

This will return a simple string on a successful REST call if this action chain was called via the `callChainAction`.

## Switch Action

The action module for this action is `vb/action/builtin/switchAction`.

This action will evaluate an expression and create an outcome with that value as the outcome name. An outcome of "default" is used when the expression does not evaluate to a usable string.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>caseValue</td>
<td>This value is used as the outcome value. If null or undefined, the outcome is &quot;default&quot;.</td>
</tr>
<tr>
<td>possibleValues</td>
<td>Optional. Array of strings, representing the allowed outcomes. If caseValue evaluates to something not in this array, the outcome is &quot;default&quot;.</td>
</tr>
</tbody>
</table>

Example:

```
"myActionChain": {
    "root": "myAction",
    "actions": {
        "myAction": {
            "module": "vb/action/builtin/switchAction",
```
"parameters": {
   "caseValue": "{{ $chain.variables.myCase }}",
   "possibleValues": ["case1", "case2"]
},
"outcomes": {
   "case1": "...",
   "case2": "...",
   "default": "..."
}
}

Take Photo Action

Use this action in your application to take photos or choose images from the system's image library.

The action module for this action is "vb/action/builtin/takePhotoAction".

The behavior of this action depends on the type of application that you use it in:

- Web application: Opens a File selection dialog to select a file
- iOS application: Prompts user with multiple options, such as Camera, Browse, or Like
- Android application: Prompts user with options, such as Camera, Browse, or Cancel
- Progressive web apps on Android and iOS: Prompts user with multiple options, such as Camera, Browse, or Like

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mediaType</td>
<td>Set to image by default. Clear the image input value from the Media Type dropdown list if you want your mobile application to use the deprecated Take Photo action implementation from pre-19.1.3 releases. The pre-19.1.3 Take Photo action can only be used in Android and iOS applications.</td>
</tr>
</tbody>
</table>

Example 8-28 Example

The outcome of this action is a binary data object (blob) duck-typed as File. The outcome name is file.

```javascript
// To use the outcome file in images, use the URL.createObjectURL and URL.revokeObjectURL
// methods, as in the following example
const blobURL = URL.createObjectURL(fileBlob);

// Release the BLOB after it loads.
document.getElementById("img-712450837-1").onload = function () {
   URL.revokeObjectURL(blobURL);
};

// Set the image source to the BLOB URL
```
document.getElementById("img-712450837-1").src = blobURL;

// To upload the selected/captured image, use restAction and set the body of
// restAction to the outcome file of takePhotoAction.
"takePhoto1": {
    "module": "vb/action/builtin/takePhotoAction",
    "parameters": {
        "mediaType": "image"
    },
    "outcomes": {
        "success": "callTakePhotoSuccess",
        "failure": "callTakePhotoFailed"
    }
},
"callRestEndpoint1": {
    "module": "vb/action/builtin/restAction",
    "parameters": {
        "endpoint": "OracleCom/postUpload",
        "body": "{{ $chain.results.takePhoto1.file }}", // <- File is
        "contentType": "image/jpeg"
    },
    "outcomes": {
        "success": "callUploadSuccess",
        "failure": "callUploadFailed"
    }
},
"callUploadFailed": {
    "module": "vb/action/builtin/callModuleFunctionAction",
    "parameters": {
        "module": "{{$page.functions}}",
        "functionName": "uploadFailed",
        "params": "{{ $chain.results.callRestEndpoint1.body }}"
    }
},
"callUploadSuccess": {
    "module": "vb/action/builtin/callModuleFunctionAction",
    "parameters": {
        "module": "{{$page.functions}}",
        "functionName": "uploadSuccess",
        "params": "{{ $chain.results.callRestEndpoint1.body }}"
    }
},

Transform Chart Data Action

The action module for this action is "vb/action/builtin/transformChartDataAction".
Transforms a JSON array with a particular structure into a JSON object containing (array) properties that JET chart component expects.

Page Authors can use this action to take the response from a REST action, turn into a format that this action expects, and use the result returned by this action to assign to a variable bound to the chart component.

The example below shows a chain "fetchTechnicianStatsChain" with 4 actions chained together to take a REST response and turn the JSON response into a form that can be used by the chart UI component. The 4 actions are

1. Call 'REST action' to fetch technician stats
2. Use 'Assign Variables action' to map the response from (1) to a form that the 'Transform Chart Data action' expects. If the REST response is deeply nested that a simple transformation of source to target using Assign Variables action is not possible then page authors can use a page function (using a Call Module Function action) to transform the data into a form that the Transform Chart Data action expects.
3. Call 'Transform Chart Data action' to take the response from (2) and turn into a form that a chart component can consume
4. Use 'Assign Variables action' to take the return value from (3) to store in page variable.

```json
"actions": {
    "fetchTechnicianStatsChain": {
        "variables": {
            "flattenedArray": {
                "type": [
                    {
                        "group": "string",
                        "series": "string",
                        "value": "string"
                    }
                ],
                "description": "array of data points",
                "input": "none"
            }
        },
        "root": "fetchTechnicianStats",
        "actions": {
            "fetchTechnicianStats": { // (1)
                "module": "vb/action/builtin/restAction",
                "parameters": {
                    "endpoint": "ifixitfast-service/getTechnicianStats",
                    "uriParams": {
                        "technician": "{{ $page.variables.technician }}"
                    }
                },
                "outcomes": {
                    "success": "flattenDataForBar"
                }
            },
            "flattenDataForBar": { // (2)
                "module": "vb/action/builtin/assignVariablesAction",
                "parameters": {
                    "$chain.variables.flattenedArray": {
                        "source": "{{ $chain.results.fetchTechnicianStats.body.metrics }}"
                    }
                }
            }
        }
    }
}
```
Fire Custom Event Action

This action allows to fire application-defined events.

This action allows you to fire application-defined events.

"actions": {
  "fireEvent": {
    "module": "vb/action/builtin/fireCustomEventAction",
    "parameters": {
      "name": "{{ $variables.name }}",
      "payload": "{{ $variables.payload }}"
    }
  }
}

Fire Notification Event Action

This action is used to fire "vbNotification" events.

"vbNotification" events are just like custom events, except they have a defined 'name', and a suggested use. Notifications are generally intended to help implement a flexible 'show message', but its specific use can be defined by the application.

"actions": {
  "fireNotification": {

Action Chains

Action chains are defined under the 'chains' property of the page model.

Topics:
- Chain Actions
- Reference Variables in Action Chains
- Action Chain Variables
- Action Results

Chain Actions

An action chain has two properties: the set of variables it can use, and the root action.

Action chains are defined under the 'chains' property of the page model. An action chain always has a root action. This root action will always be called when the action chain is invoked.

This action chain will call the 'myAction' action:

```
"chains": {
  "myActionChain": {
    "root": "myAction",
    "actions": {
      "myAction": {
        "label": "My action!",
        "module": "vb/action/builtin/someAction",
        "parameters": {
          "key": "value"
        }
      }
    }
  }
}
```

Each action has an outcome. Usually, an action supports the "success" or "error" outcomes. Some actions may also support other outcomes. Actions can be chained by connecting an additional action to a previous action's outcome.

To perform another action if the previous action succeeds, and handle error cases if it does not succeed, you could do the following:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/fireNotificationEventAction",
      "parameters": {
        "message": "{{ $variables.message }}",
        "type": "{{ $variables.type }}",
        "displayMode": "{{ $variables.displayMode }}"
      }
    }
  }
}
```
Reference Variables in Action Chains

Variables can be referenced for the parameter values of an action.

The runtime will automatically evaluate parameter values as expressions. Similar to the default value syntax of variables, variables can be referenced directly into an action parameter's value:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "label": "some action",
      "module": "vb/action/builtin/someAction",
      "parameters": {
        "key": "{{ $page.variables.myVariable }}"
      }
    }
  }
}
```

Simple Javascript code can be added to the values:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "label": "some action",
      "module": "vb/action/builtin/someAction",
      "parameters": {
        "key": "{{ $page.variables.myVariable === 'yellow' }}"
      }
    }
  }
}
```

Non-expressions are entered in JSON:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/someAction",
```

---

**Chapter 8**

**Action Chains**

8-86

---
"parameters": {
  "myString": "somestaticvalue",
  "myNumber": 1
  "myBoolean": true
}
}
}

Map and array values are also expressed in JSON:

"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/someAction",
      "parameters": {
        "key": {
          "key1": "static value",
          "key2": "{{ $page.variables.something }}"
        }
      }
    }
  }
}

Action Chain Variables

An action chain can also have variables. These are defined and used in the same way as page parameters.

Unlike page parameters, input variables only support the ‘fromCaller’ or ‘none’ type. Input variables must be specified by event handlers calling into action chains.

Action chain variables can be assigned to or read from using the syntax: $chain.variables.<varName> and are only accessible within an action chain. They can also be referenced by the shorthand $variables.<varName> within the chain.

Example 8-29  Example

(Optional) Enter an example here.

"myActionChain": {
  "variables": {
    "id": {
      "type": "string",
      "description": "the ID of something to update",
      "input": "fromCaller",
      "required": true
    }
  },
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/someAction"
    }
  }
}
Action Results

Actions in an action chain can return a result that can be used by subsequent actions in the chain.

After an action implementation is run, it may return a result. The type of these results are specific to an implementation of an action. This result will be stored in a special variable, $chain.results. The results of a previous action are contained within $chain.results.<actionId>.

Example 8-30 Accessing a Previous Action`s Results

To access a previous action's results:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/someAction",
      "outcomes": {
        "success": "someOtherAction"
      }
    },
    "someOtherAction": {
      "module": "vb/action/builtin/someAction",
      "parameters": {
        "myKey": "\$$chain.results.myAction\"
      }
    }
  }
}
```

Example 8-31 Action Chain Return Type and Outcomes

You can specify a return type and an array of outcomes:

```
"myActionChain": {
  "root": "myAction",
  "actions": {
    "myAction": {
      "module": "vb/action/builtin/someAction"
    },
    "returnType": "application:someType",
    "outcomes": ["success", "failure"]
  }
}
```

Components

Components are written as an HTML file, using standard HTML syntax.

Topics:
- HTML Source
- Component Events
- Custom Events
• Import Custom Components

HTML Source

Components are written as standard HTML files.

The HTML file for a page is located as a peer to the page model, as name-page.html. This html source can be edited as a normal Jet page.

Currently, Jet 7.1.0 is used, including the custom element syntax. You can find documentation on the specific version of Jet in the Jet Cookbook.

There are some expression changes. There are currently two expressions, write-back or no write-back. This can be seen in the component properties.

Example 8-32  Example:

<oj-input-text
placeholder="[[$variables.searchText]]" value="[[$variables.searchVariable]]"><oj-input-text>

Component Events

Component event listeners are defined within the eventsListeners section of the page model, along with Page lifecycle events.

Also known as DOM Events, these are similar to Page events, except that they are fired by components on a page.

An event listener can have any name, but must be bound to a component event. Component event listeners are defined in the Page module under the 'eventListeners' property. Custom events also propagate up the page's container hierarchy, executing any handlers found in parent containers or their shells. Event propagation can be stopped.

Event listeners are defined within the eventsListeners section of the page model, along with page lifecycle events.

Referencing Event Listeners From Components

Component event listeners are called in the same way as page lifecycle event listeners. There can be more than one listener. When there is more than one, they run in parallel).

A new implicit object, $event, can be used in this context. This corresponds to the event payload sent by the component.

To reference an event listener from a component, you can use the $listeners.<eventListenerName> implicit object.

Example 8-33  eventListeners Example:

"eventListeners": {
  "onSelectionChange": {
    "chains": [
      {"chainId": "respondToChange",
       "parameters": {
        "text": "{{ $event.detail.value }}"}}
    ]
  }
}
Custom Events

Custom events are similar to page events, except that they are not limited to lifecycles. Their event listeners can be defined in a page, flow, or application.

An event name is defined by the user, and are explicitly fired by the application, using the event Actions provided, in the context of a page.

Custom event listeners are defined in the page, shell, or flow under the `eventListeners` property.

One difference between custom events and page events is that they 'bubble' up the containment hierarchy. Any event listeners in a given flow or page for the event are executed before looking for listeners in the container's parent. The order of container processing is:

- The page from where the event is fired.
- The flow containing the page.
- The page containing the flow.
- Recursively up the containment, ending with the application.

Custom and system event behavior can be modified using the `stopPropagation` and `preventDefault` properties:

- `stopPropagation` can be used to prevent the event from bubbling to this event listener's container's parents.
- `preventDefault` can be used to prevent the 'normal' DOM event handling from being executed.

Example 8-35  stopPropagation Example

```
"eventListeners": {  
  "customEventTwo": {  
    "stopPropagation": "{{ $event.type === 'info' }}"  
  }  
},
```

Example 8-36  preventDefault Example

```
"eventsListeners": {  
  "customEventTwo": {  
    "chains": [  
    
  },
```

Example 8-34  Referencing an Event Listener From a Component

```html
<oj-select-one ... on-selection-change="[[$listeners.onSelectionChange]]"
```
Import Custom Components

JET Custom Components can be loaded using the "imports" section in a shell or page. The 'components' section contains a map of component IDs to objects which contain a (requireJS) path to the JET Custom Components loader javascript. The ID should match the component tag.

Example 8-37  Example:

```
"imports": {
  "components": {
    "demo-card": {
      "path": "resources/components/democard/loader"
    }
  }
}
```

Security

The **security** entry provides certain access limits.

The **security** entry provides a way to limit access to UI level artifacts, such as pages, flows, or applications. These artifacts can require either a specific role or a specific permission in order to enter and display the resource. If the user does not have the correct role or permission, the runtime will refuse entry into that UI artifact. Currently the application, flows, and individual pages can be protected in this manner.

- Security Configuration
- Security Provider
- userInfo
- Error Handling

Security Configuration

The security configuration is managed in several resources.

The configuration for security resides in the model for each of these resources: `app-flow.json`, `<n>-flow.json`, `<n>-page.json`. If `requiresAuthentication` is false, specifying roles or permissions results in an error. By default an artifact inherits the `requiresAuthentication` from its parent. If this is not present in the application configuration, it defaults to true. The configuration follows the format seen in this example:

```
"security": {
  "access": {
    "requiresAuthentication": true/false,
```
When an anonymous user navigates to an artifact (page, flow or application) and the artifact is secure, the user is prompted to login, and is then redirected to the artifact. This functionality is provided by the default implementation of the Security Provider.

Security Provider

Security for an application is enabled using a pluggable mechanism called Security Providers.

In the application model, app-flow.json, you can specify a "userConfig" element. The userConfig element selects which Security Provider to use and how to configure it.

A Security Provider takes a configuration object with a url. The url property should point to a Rest API. It must be possible to retrieve the current Security Provider configuration via this Rest API. The configuration contains user information and configuration information such as loginUrl and logoutUrl.

A Security Provider performs the following:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchCurrentUser(config)</td>
<td>Fetch the configuration from the url and initialize the userInfo property as well as the loginUrl and logoutUrl properties.</td>
</tr>
<tr>
<td>static getUserInfoType()</td>
<td>Return an object describing the type of the user info.</td>
</tr>
<tr>
<td>isAccessAllowed(type, path, accessInfo)</td>
<td>Check if the current user can access a resource with the given access info. If the user is not authenticated, this method returns false. Otherwise, if the user role is one of the roles in accessInfo, or if the user permission is one of the permissions in accessInfo, then the method returns true.</td>
</tr>
<tr>
<td>handleLoadError(error, returnPath)</td>
<td>This function is called by the client when an error occurs while loading a page. It attempts to handle the load error for a Visual Builder artifact, and returns true if it does.</td>
</tr>
<tr>
<td>headers</td>
<td>Handle the user login process. Redirect to the login page using the login URL given by the security provider configuration. If defined, the return path is added to the login URL using the query parameter name. This is defined in the 'returnPathQueryParam' property of the SecurityProvider class.</td>
</tr>
<tr>
<td>handleLogout(logoutUrl)</td>
<td>Handle the user logout process. The default implementation navigates to the URL defined by the logoutUrl argument. If the logoutUrl argument is not defined, it uses the logoutUrl of the SecurityProvider configuration.</td>
</tr>
</tbody>
</table>

Example 8-38   Example of an entry in app-flow.json to specify the Security Provider

```
"userConfig": {
    "type": "vb/DefaultSecurityProvider",
    "configuration": {
        "url": "url to some security api"
    }
}
```
User Information

The `userInfo` contains the user information fetched by the Security Provider.

For the default implementation, the `userInfo` has the following type:

```json
{
    "userId": "string",
    "fullName": "string",
    "email": "string",
    "roles": "string[]",
    "permissions": "string[]",
    "isAuthenticated": "boolean"
}
```

The `userInfo` is made available to the application with the help of the `$application.user` built-in variable. This allows content in the page to be rendered conditionally.

### Example 8-39  Example of conditional content rendering

```html
<!-- Render 'I am a manager' if manager is a role of the current user -->
<oj-bind-if test="[[!$application.user.roles.manager]]">
    I am a manager
</oj-bind-if>

<!-- Render the 'Sign In' button if the current user is not authenticated -->
<oj-bind-if test="[[!$application.user.isAuthenticated]]">
    <oj-button id='signIn' on-oj-action="[[listeners.onSignIn]]">Sign In</oj-button>
</oj-bind-if>
```

Error Handling

Support for unauthorized error handling is provided by several functions.

When loading an artifact returns an error, the function `handleLoadError` is called with an error object that has a `statusCode` property. If the artifact is secure and the roles and permissions of the current user do not match the ones required by the artifact, the error statusCode is 403. The default implementation of the `handleLoadError` will check if the user is authenticated, and if not, will call the `handleLogin` function. This redirects to the loginUrl provided by the Security Provider configuration.

The default implementation of the Security Provider handles status 401 and 403 errors. Other security schemes will need to implement their own security provider and specify it in the UserConfig section of the application descriptor. To implement your own security provider:

1. Create your own class extending `vb/types/securityProvider` and override any method necessary.

2. If the user information is different, make sure to match the content of the userInfo property and the type information returned by `getUserInfoType()`, since this determines what information is exposed in the `$application.user` variable.

3. Enter your new type in the "type" section of the userConfig in app-flow.json as well as the URL to retrieve the Security Provider configuration.
**Example 8-40  Example of a custom Security Provider**

```javascript
define(['vb/types/securityProvider'],
    (SecurityProvider) => {
        class TestSecurityProvider extends SecurityProvider {
            handleLogin(returnPath) {
                // implement your own login mechanism here
            }
        }

        return TestSecurityProvider;
    });
```

---

**Translations**

An API available for getting localized strings using `$<container>.translations`. Translations bundles may now be defined declaratively in Application, Flow, or Page containers. The properties of the "translations" object are the names of the bundle, and the value must contain a "path" property that is the path to the bundle.

When declaring bundles in the Application level, there is an optional "merge" property which allows you to specify an existing bundle path, which this bundle should merge with, and override. This allows overriding existing bundles in JET, or JET CCs, with Application-level bundles. Expressions for "merge" are supported, but cannot references to Application artifacts, as this evaluation happens before the creation of the Application.

The bundle must be in a folder named 'nls': the path can be any depth, but the last folder in the path must be 'nls', such that the root bundle is in the nls/ folder. Translation bundles are the standard JET bundle format. String resolution uses the JET oj.Config.getLocale() to get the current locale for the context.

The following paths are supported for "path":

- **container relative**: a path fragment prefixed by "./" (dot-slash) will be appended to the declaring container's (flow, page) path. Note that flows and pages are not allowed to reach outside of its container (the path cannot reference parent folders). This means that "./" is not allowed anywhere in the path.
- **application relative**: a path fragment without a "./" prefix will be relative to the application root. This is discouraged for Flows or Pages, except where a RequireJS path mapping is being used.
- **absolute**: paths that contain a host are used as-is.

Applications that used translations prior to 18.2.3 **must** manually migrate their translations. Translations previously used the JET configuration, and therefore had one bundle for the entire app. You have several options:

- Declare the bundle; you can chose to break the bundle up logically, but the simplest migration would be to use the exact example above in app-flow.json, which uses the path for the existing bundle provided for new apps.
- Change the expression syntax to the new syntax. Assuming you declared your single bundle in the same manner as Example 1.
  - For just the translated string, change `$application.translations.get(<key>)` to `$application.translations.app.<key>`
– For Strings that require replacement, change $application.translations.get( <key>, <arguments>) to $application.translations.format( 'app', <key>, <arguments> )

By default, VB defers to JET to determine the current locale for the client. This is typically done by first looking at the <html> tag 'lang' attribute, and then falling back to some browser settings.

There is a new "localization" declaration section in the Application model (app-flow.json) that contains a "locale" property. This allows the developer to specify an alternate locale. This configures the JET ojL10n plugin to use this locale.

Expressions may be used, but the application is not created at this point, and therefore no application functions or variables are available. Instead, the developer must provide the necessary Javascript. The developer should also set the 'lang' attribute on the <html> tag, so that JET, and anything that uses JET, will also use this locale.

**Example 8-41  Bundles**

Two bundles, translations.js and moreTranslations.js, defined in a Page model json, named "app" and "anotherBundle":

```json
"translations": {
    "app": {
        "path": "./resources/nls/translations"
    },
    "anotherBundle": {
        "path": "./resources/nls/moreTranslations"
    }
}
```

The corresponding expression syntax would be as follows, with one expression per bundle:

```html
<h4><oj-bind-text value="\[$page.translations.anotherBundle.description\]"</oj-bind-text></h4>
<span>
    <oj-bind-text value="\[$page.translations.format('app', 'info.instructions', {
        pageName: 'index.html' })\]"</oj-bind-text>
</span>
<br/>
```

**Example 8-42  Overriding**

Overriding both JET strings and a component's strings:

```json
{
    "applicationModelVersion": "18.3.1",
    "id": "demoCardDemo",
    "description": "Custom Component, Demo Card, with methods",
    "defaultPage": "shell",
    "translations": {
        "main": {
            "path": "/resources/nls/translations",
            "merge": "/ojtranslations/nls/ojtranslations"
        },
        "dcoverride": {
            "path": "/resources/nls/demo-card-overrides",
            "merge": "/resources/components/democard/resources/nls/demo-card-translations"
        }
    }
}
```
Helper Utilities

An API available for getting localized strings using $<container>.translations.

VB runtime provides public JavaScript helpers to help with implementing some features in JavaScript when a lower level of control is desired or needed. These can be imported in your Javascript module functions.

Topics:
  • Rest Helper
  • Module Function Event Helper

Rest Helper

The Rest helper utility allows calling REST endpoints, which are defined in the service definitions.

The Rest helper looks at the content-type header, if available, to try to determine how to read and parse the response. If no content-type is available, text is assumed. This helper is what Visual Builder runtime uses internally.

Content types that are currently understood, and the corresponding response method used:

Table 8-4  Rest helper content types

<table>
<thead>
<tr>
<th>Content type</th>
<th>Response method</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains &quot;json&quot;</td>
<td>Response.json()</td>
</tr>
<tr>
<td>starts with &quot;image/&quot;</td>
<td>Response.blob()</td>
</tr>
<tr>
<td>application/octet-stream</td>
<td>Response.blob()</td>
</tr>
</tbody>
</table>

This behavior can be overridden using the responseBodyFormat() method.
<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static get(endpointId)</td>
<td>endpointId: <code>&lt;server ID&gt;/&lt;operation ID&gt;</code>, same as RestAction, ServiceDataProvider</td>
<td>instance of Rest object</td>
<td>factory method</td>
</tr>
<tr>
<td>initConfiguration(initConfig)</td>
<td>initConfig: the initConfig of the fetch() Request object</td>
<td>Rest helper, to allow chaining of method calls.</td>
<td>Set the parameter for the call. Parameters defined as path parameters for the endpoint will be inserted in the URL as appropriate; the rest will be appended as query parameters.</td>
</tr>
<tr>
<td>parameters(parametersMap)</td>
<td>ParameterMap: object of key/value pairs, same as RestAction 'uriParams'</td>
<td>Rest helper</td>
<td>Set parameter for call. Parameters defined as path parameters for the endpoint will be inserted in the URL as appropriate; the rest will be appended as query parameters.</td>
</tr>
<tr>
<td>requestTransformation Functions</td>
<td>transformationFunctionMap: map of functions.</td>
<td>Rest helper</td>
<td>See Rest action.</td>
</tr>
<tr>
<td>(transformationFunctionMap)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>requestTransformation Options</td>
<td>transformationFunctionMap: map of request transform parameters</td>
<td>Rest helper</td>
<td>See Rest action.</td>
</tr>
<tr>
<td>(transformationOptionMap)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>responseTransformationFunctions</td>
<td>transformationFunctionMap: map of functions.</td>
<td>Rest helper</td>
<td>See Rest action.</td>
</tr>
<tr>
<td>(transformationFunctionMap)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>body(body)</td>
<td>body: actual payload to send with the request</td>
<td>Rest helper</td>
<td></td>
</tr>
<tr>
<td>hookHandler(handler)</td>
<td>handler: should extend RestHookHandler, and may override the following:</td>
<td>Rest helper</td>
<td>Allows installation of callbacks for various phases of the REST call, which may configure the Rest helpers, modify the request and response, or do special processing based on the result or result error.</td>
</tr>
<tr>
<td></td>
<td>handlePrefetchHook(rest)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>handleRequestHook(request)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>returns request</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>handleResponseHook(response)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>returns response</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>handlePostFetchHook(result)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>handlePostFetchErrorHook(result)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```cpp
class MyHandler extends RestHookHandler {
    define(['vb/helpers/rest', 'vb/helpers/rest'],
        {Rest, RestHookHandler} => {
            class MyHandler extends RestHookHandler {
```
Table 8-5   (Cont.) Rest helper methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>responseBodyFormat(format)</td>
<td>format: one of: text, json, blob, arrayBuffer, base64, or base64Url. The response body type is the same as the corresponding method for Response (except base64, which returns just the encoded portion of the base64 URL).</td>
<td>Rest helper</td>
<td>Overrides the default behavior, which looks at the &quot;content-type&quot; header to determine how to read (and parse) the response.</td>
</tr>
<tr>
<td>fetch()</td>
<td></td>
<td>Promise</td>
<td>Performs the configured fetch() call</td>
</tr>
<tr>
<td>toUrl()</td>
<td></td>
<td>Promise</td>
<td>Utility methods for building requests and responses that require the endpoint path. Resolves with the full (or relative) path of the endpoint, or empty string if the endpoint is not found.</td>
</tr>
<tr>
<td>toRelativeUrl()</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Rest helper fetch() call returns a Promise that resolves with an object that contains the following properties:

Table 8-6   fetch() call return value

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>response</td>
<td>The Response object from the native fetch() call, or the return from a HookHandler's handleResponseHook, if one is being used.</td>
</tr>
<tr>
<td>body</td>
<td>The body of the response object; the helper will attempt to call the appropriate Response method (json(), blob(), arrayBuffer(), etc) based on responseBodyFormat() and Content-Type.</td>
</tr>
</tbody>
</table>

Example 8-44   Overriding

Overriding Rest helper behavior:

```javascript
define(['vb/helpers/rest'], (Rest) => {
...
var rest = Rest.get('myservice/myendpoint').parameters(myparameters);
var promise = rest.fetch();
```

Module Function Event Builder

An API available for getting localized strings using $<container> translations.
Within the context of module functions including `main-page.js` and `app-flow.js`, there is an event helper available, to allow raising custom events, similar to Fire Custom Event Action. The helper is made available to the module function via a context passed to the Module classes constructor, and has two methods available.

### Table 8-7  Module function event helper methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fireCustomEvent(name, payload)</code></td>
<td>See Fire Custom Event Action.</td>
</tr>
<tr>
<td><code>fireNotificationEvent(options)</code></td>
<td>See Fire Notification Event Action.</td>
</tr>
</tbody>
</table>

### Example 8-45  Usage in a module function

```javascript
// use strict;

define(function () {
  function MainPageModule(context) {
    this.eventHelper = context.getEventHelper();
  }

  MainPageModule.prototype.fireCustom = function (name, payload) {
    return this.eventHelper.fireCustomEvent(name, payload);
  }

  MainPageModule.prototype.fireNotification = function (subject, message) {
    return this.eventHelper.fireNotificationEvent({ subject, message, type: 'info' });
  }

  return MainPageModule;
});
```

## Events

There are several types of events, all of which the application can react to, using the Event Listener syntax.

There are several types of Events in the runtime: Page Events, System Events, Custom or developer-defined System events, Component (DOM) Events, and Variable Events. Event types are all handled by executing action chains.

The application reacts to events through Event Listeners, which declaratively specify action chains to execute when the event occurs.

### Event topics:

- Page Events
- Component Events
- Custom Events
- System Events
- Variable 'onValueChanged' Events
Declared Events

Declared Events are events that are explicitly defined in the application model, to define a specific contract and type for the event.

Events can be declared at the Application, Flow, or Page. References to events use prefixes, just like variables and chains.

Events have a "payloadType" which declares the type of the event payload. This type is limited to simple scalar types, or objects and arrays composed of scalar types; you cannot define a "payloadType" that references other type definitions.

Example 8-46 Declaration

```
"events": {  
  "myPageEvent": {  
    "payloadType": {  
      "message": "string",  
      "code": "number"  
    }  
  }  
},
```

Example 8-47 Listener

```
"eventListeners": {  
  "page:myPageEvent": {  
    "chains": [  
      {  
        "chainId": "handleEvent",  
        "parameters": {  
          "payload": "{{ $event }}"  
        }  
      }  
    ]  
  }  
},
```

Page Events

Page events are events that can have event handlers. These are defined in the page module under the events property. A single event can have multiple event handlers.

Page events are lifecycle events, and are defined by the system. Event listeners are defined in a page, and share the name of the event. Page events have a 'vb' prefix. When a page event is raised, the framework calls the event listener with the name of the event, as defined in the page.

Page event listeners are defined in the page module under the "eventListeners" property. Like all event types, a single page event can have multiple event listeners. Event listeners call action chains and can pass parameters:

```
"eventListeners": {  
  "vbEnter": {  
```
"chains": [
  {
    "chainId": "handleEnterOne",
    "parameters": {
      "id": "1"
    }
  },
  {
    "chainId": "handleEnterTwo"
  }
],
}

The parameters are expressions, and are evaluated just like action parameters. The parameters must of course match the action chains input variables. If multiple event listeners are defined they will be called in parallel to each other.

The order of execution during navigation from page source to page target is:

1. vbBeforeExit is dispatched to the source page.
2. vbBeforeEnter is dispatched to the target page.
3. vbExit is dispatched to the source page.
4. vbEnter is dispatched to the target page.

Table 8-8  Page Event Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbBeforeEnter</td>
<td>Dispatched to a page before navigating to it. Three variable scopes are available:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• $application: All application variables</td>
<td>{cancelled: boolean}</td>
</tr>
<tr>
<td></td>
<td>• $flow: All parent flow variables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• $parameters: All page input parameters from the URL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page authors are able to cancel navigation to this page by returning an object with the property cancelled set to true. This is useful if the user does not have permission to this page or to redirect to an other page.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>vbEnter</td>
<td>Dispatched after all the page scoped variables have been added and initialized to their default values, values from URL or persisted values. Variable scopes available: • $application: All application variables • $flow: All parent flow variables • $page: All page variables Use this to trigger data fetches. These may occur concurrently.</td>
<td>None</td>
</tr>
<tr>
<td>vbBeforeExit</td>
<td>Dispatched to a page before exiting it. This event is can be cancelled, giving the opportunity to the page author to stop navigation away from the page. Cancellation is done by returning an object with the property cancelled set to true to the listener chain. This is useful when the page is dirty and leaving the page should not be allowed before saving.</td>
<td>{cancelled: boolean}</td>
</tr>
<tr>
<td>vbExit</td>
<td>Dispatched when exiting the page. This event can be used to clean up resources before leaving the page.</td>
<td></td>
</tr>
<tr>
<td>vbBeforeAppInstallPrompt</td>
<td>Dispatched when a PWA receives BeforeInstallPromptEvent from the browser. The event will be dispatched after vbBeforeEnter, but there is no guarantee that it will be dispatched after vbEnter. The vbBeforeAppInstallPrompt event can be used to display a native application install prompt by calling event.getInstallPromptEvent().prompt(). Currently, this is only supported in Chrome. For PWAs, the event will be handled automatically by the root page.</td>
<td>{getInstallPromptEvent() }</td>
</tr>
</tbody>
</table>

Table 8-8 (Cont.) Page Event Parameters
### Table 8-8  (Cont.) Page Event Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbPause</td>
<td>Dispatched as a response to Apache Cordova's pause event. The event fires when the native platform puts the native mobile application into the background, typically when the user switches to a different native mobile application. For more information, including differences in behavior between the Android and iOS platforms, please read Apache Cordova's documentation.</td>
<td>None</td>
</tr>
<tr>
<td>vbResume</td>
<td>Dispatched as a response to Apache Cordova's resume event. The event fires when the native mobile platform returns the native mobile application from the background. For more information, including differences in behavior between the Android and iOS platforms, please read Apache Cordova's documentation.</td>
<td>None</td>
</tr>
</tbody>
</table>

### Component Events

Component event listeners are defined within the eventsListeners section of the page model, along with Page lifecycle events.

Also known as DOM Events, these are similar to Page events, except that they are fired by components on a page.

An event listener can have any name, but must be bound to a component event. Component event listeners are defined in the Page module under the 'eventListeners' property. Custom events also propagate up the page’s container hierarchy, executing any handlers found in parent containers or their shells. Event propagation can be stopped.

Event listeners are defined within the eventsListeners section of the page model, along with page lifecycle events.

**Referencing Event Listeners From Components**

Component event listeners are called in the same way as page lifecycle event listeners. There can be more than one listener. When there is more than one, they run in parallel.)
A new implicit object, $event, can be used in this context. This corresponds to the event payload sent by the component.

To reference an event listener from a component, you can use the $listeners.<eventListenerName> implicit object.

Example 8-48  eventListeners Example:

```
"eventListeners": {
  "onSelectionChange": {
    "chains": [
      {
        "chainId": "respondToChange",
        "parameters": {
          "text": "{{ $event.detail.value }}"
        }
      }
    ]
  }
}
```

Example 8-49  Referencing an Event Listener From a Component

```
<oj-select-one ... on-selection-change="[${$listeners.onSelectionChange}]"
```

Custom Events

Custom events are similar to page events, except that they are not limited to lifecycles. Their event listeners can be defined in a page, flow, or application.

An event name is defined by the user, and are explicitly fired by the application, using the event Actions provided, in the context of a page.

Custom event listeners are defined in the page, shell, or flow under the eventListeners property.

One difference between custom events and page events is that they 'bubble' up the containment hierarchy. Any event listeners in a given flow or page for the event are executed before looking for listeners in the container's parent. The order of container processing is:

- The page from where the event is fired.
- The flow containing the page.
- The page containing the flow.
- Recursively up the containment, ending with the application.

Custom and system event behavior can be modified using the stopPropagation and preventDefault properties:

- stopPropagation can be used to prevent the event from bubbling to this event Listener's containers parents.
- preventDefault can be used to prevent the 'normal' DOM event handling from being executed.

Example 8-50  stopPropagation Example

```
"eventListeners": {
  "customEventTwo": {
    "stopPropagation": "{{ $event.type === 'info' }}"
  }
}
```
System Events

System events are identical to custom page events, except that the framework defines the event.

An event name is defined by the user, and are explicitly fired by the application, using the event Actions provided, in the context of a page.

System event listeners are defined in the page, shell, or flow under the `eventListeners` property.

System events also propagate or bubble up the page's container hierarchy, executing any listeners. Event bubbling can be stopped.

One difference between system events and page events is that they 'bubble' up the containment hierarchy. Any event listeners in a given flow or page for the event are executed before looking for listeners in the container's parent. The order of container processing is:

- The page from where the event is fired.
- The flow containing the page.
- The page containing the flow.
- Recursively up the containment, ending with the application.

Custom and system event behavior can be modified using the `stopPropagation` and `preventDefault` properties:

- `stopPropagation` can be used to prevent the event from bubbling to this event Listener's containers parents.
- `preventDefault` can be used to prevent the 'normal' DOM event handling from being executed.
Example 8-52  stopPropagation Example

```
"eventListeners": {
    "customEventTwo": {
        "stopPropagation": "{{ $event.type === 'info' }}"
    }
}
```

Example 8-53  preventDefault Example

```
"eventsListeners": {
    "customEventTwo": {
        "chains": [
            {
                "actionsId": "handleEventInMod2PageChain",
                "parameters": {
                    "eventPayload": "{{ $event }}"
                }
            }
        ],
        "preventDefault": "{{ $event.type === 'info' }}"
    }
}
```

Variable ‘onValueChanged’ Events

Specific to variables, the ‘onValueChanged’ event is raised by the framework when a variable’s value changes.

To add an event listener to an event, specify it in the ‘onValueChanged’ property of the variable. Event listeners can only be added to the root variable, not to any sub-objects of the variable structure. It uses the same syntax as other event listeners.

Old and new variable values are available in the $event implicit object.

- $event.oldValue provides the variable’s old value.
- $event.value provides the variable’s new value.
- $event.diff can be used for complex types, where it is necessary to know the properties within the variable that changed.

Example 8-54  Example:

```
"variables": {
    "incidentId": {
        "type": "string",
        "input": "fromCaller",
        "required": true,
        "onValuechanged": {
            "chains": [
                {
                    "chainId": "fetchIncidentChain",
                    "parameters": {
                        "incidentId": "{{ $event.value }}"
                    }
                }
            ]
        }
    }
}
```
Common Use Cases

Here are some common scenarios that Visual Builder users encounter.

Topics:

• Enable Client-Side Validation for a Form
• Format Row Values in a Table Conditionally
• Validate Dates in Forms
• Get The Label Value from a Selected Item in a List
• Run Visual Builder Applications On Other Servers
• Style Visual Builder Applications
• Use a Check Box to Select Rows in a Table

Enable Client-Side Validation for a Form

You may want to set up a form so that your application can check the validity of its contents before the user submits it.

To do this, surround the form with an oj-validation-group element, and add a custom isFormValid Javascript function that returns a boolean. You can then call that function before the form is submitted.

Suppose you have a form with three text fields. To set up a basic client-side validation for this form:

1. Open the page that contains the form.
2. Click the Code button to switch to the code view of the page.
3. Locate the div element that contains the oj-form-layout element. Enclose this div element in an oj-validation-group element that has an id attribute. For example:

   <oj-validation-group id="CreateForm">

   Make sure to add the closing tag.
4. Click the application node, then click the **Functions** tab to view `app-flow.js`.

```
var AppModule = function AppModule() {};

AppModule.prototype.isFormValid = function(form) {
    var tracker = document.getElementById(form);
    if (tracker.valid === "valid") {
        return true;
    } else {
        tracker.showMessages();
        tracker.focusOn("@firstInvalidShown");
        return false;
    }
};
```

5. Add the `isFormValid` function as shown. The function code appears in bold.

```
var AppModule = function AppModule() {};

AppModule.prototype.isFormValid = function(form) {
    var tracker = document.getElementById(form);
    if (tracker.valid === "valid") {
        return true;
    } else {
        tracker.showMessages();
        tracker.focusOn("@firstInvalidShown");
        return false;
    }
};
```

6. Go back to the page with the form. Click the **Save** button, then click the **Actions** tab for the button, and then click `createExpenseReportChain`.
7. Drag an If action after Start.

8. In the Condition field, enter the following. The argument to the `isFormValid` function is the id value for the `oj-validation-group` element.

   ```
   {{ $application.functions.isFormValid("CreateForm") }}
   ```

9. Move the Call businessObjects... node to the true branch of the If action.

You can now test the form validation.

**Format Row Values in a Table Conditionally**

You can use a column template to specify row-specific formatting for particular values in a table column.

Suppose, for instance, that your table has a Salary column and you want to display the values that fall above a certain level in bold. In your table, you can represent the Salary field of the business object as a separate column template, so that you can define the format for this field.
1. In the JavaScript for the page, define a `PageModule` function that determines the format you want to show. This code defines a `weight` function to set the font weight:

   ```javascript
   PageModule.prototype.weight = function(salary) {
       if (salary > 2000) {
           return "bold";
       }
       return "normal";
   };
   ```

2. To create the column template, drag and drop a `Text` component onto the existing field, then click the `Code` button for the page.

3. Surround the field with a `span` element within the `template` element. Make sure to put a colon in front of the `style` attribute.

   ```html
   <template slot="Salary">
   <span :style.font-weight="{$page.functions.weight($current.data)}">
       <oj-bind-text value="[[{$current.data}]]">
       </oj-bind-text>
   </span>
   </template>
   ```

When the page is displayed, all salary values above 2000 appear in bold.

**Validate Dates in Forms**

You can use the Expression Editor to validate a date you enter in a form.

Suppose you have a form for creating a business object instance that has a `Start Date` and an `End Date` field. You want to be sure that the end date can't be earlier than the start date. To do this:

1. In the Page Designer, select the `Input Date` component for `End Date` in the form.

2. In the `General` tab of the Property Inspector, click the `fx` icon for the `Min` property.

3. In the left panel, expand the business object and double-click `startDate`.

   The expression `variables.expenseReport.startDate` is displayed in the editor pane (where `expenseReport` is the name of the page variable).
4. Click **Save**.

The expression is displayed in the **Min** property, surrounded by double brackets. As a result, the DatePicker for the **End Date** field makes all dates before the **Start Date** unavailable. If you manually enter a date before the **Start Date**, you’ll see an error message.

You may also need to specify a format for dates. See [How to Format a Date Field](#) for information on how to format a date field of a business object.

---

### Get The Label Value from a Selected Item in a List

While it's easy to access the value of a selected item in a list, accessing the label is not quite as easy.

When you add a select list component such as **Select (One)** to a page, Visual Builder provides a wizard that helps you to bind the component to your data. You can then easily access the value that you select in the list. However, sometimes you might want to access the label of the list item rather than the value.

The **oj-select-one** component has an attribute called **value-option**. This contains an object with two fields, **value** and **label**, that are synchronized with the list. This attribute is not currently available in design time, but you can still use it. To do so:

1. Create a page variable of type **object**, then add two string fields called **label** and **value**. Here, the page variable is named **selectedVal**:

```
Variables  Types

Filter

Constants

No Constants defined.

Variables

**cityListSDP**

**selectedVal**

- label
- value
```
2. In the Page Designer, switch to the code editor view and add the `value-option` attribute to your component.

```html
<div class="oj-flex">
    <oj-label id="oj-label-627983464-1" for="oj-select-one-627983464-1" class="oj-flex-item oj-sm-12 oj-md-2">Select City</oj-label>
    <oj-select-one id="oj-select-one-627983464-1" class="oj-flex-item oj-sm-12 oj-md-4" value-option="{$page.variables.selectedVal}" options="[$page.variables.cityListSDP]" options-keys.value="id" options-keys.label="name"></oj-select-one>
</div>
```

You can now refer to the label using a syntax like this:

```javascript
{{ $page.variables.selectedVal.label }}
```

---

**Run Visual Builder Applications On Other Servers**

While it's possible to run Visual Builder web applications on other web servers, you do lose some functionality.

Specifically:

- Business objects won't run, because they require the Visual Builder back end.
- You can't use Identity Cloud Service to manage your users, roles, or authentication, so you'll have to manage these aspects of your app.
- The Visual Builder server authentication proxy manages connections to REST services, so you'll need to define your Visual Builder services to use a "Direct (Bypass Authentication Proxy)" connection. The calls are then made directly from the browser to the remote REST service. See Fixed Credentials Authentication Mechanisms for more information.

If these limitations are acceptable, then you can host your Visual Builder app on another server.

To modify your app to do this:

- Use the direct access to your REST services, and switch the set of services that the app is accessing. One way is by doing a global search and replace, to update the address of the server hosting the REST services that provides data to the app. This will allow the back end to be on-premise.
- Ensure that you've allowed anonymous access to the app. Identity Cloud Service won't be available to manage authentication.
- Create a zip file that contains the app ready to be deployed. See Optimize Your Builds for information on how to do this.

You can take this optimized version of the app and host it as a regular collection of HTML/JavaScript resources on a web server.

---

**Style Visual Builder Applications**

All styling in Visual Builder applications happens manually in CSS. There are no declarative features for changing the display of text or images. Because all Visual...
Builder applications are just JET applications, they use JET Themes to style the applications.

When you style your Visual Builder or JET applications, it's important to use theming correctly. Otherwise, you run the risk of finding that your re-styling breaks when you upgrade your platform versions. JET components provide a styling API through the use of Sass (Syntactically Awesome Style Sheets) only, and any directly styled \texttt{oj-*} component selectors or other approaches aren't supported. This is because the internal structure of JET components, and the styles that they use, are subject to change from release to release.

The published set of Sass variables constitutes a stable supported API that will internally re-map the configuration as required by internal changes within the JET implementation. See “Theming Applications” in Developing Applications with Oracle JET. To get the latest version, scroll to the bottom of https://www.oracle.com/webfolder/technetwork/jet/index.html and click Developer's Guide under Docs.

By default, all Visual Builder applications use the Alta UI Theme. Visual Builder applications have a built-in \texttt{app.css} file that you can use to define additional styling on top of the main Alta theme. For example, if you've made a \texttt{div} element clickable, you may want to add a class called \texttt{clickable} to the \texttt{div} and define the CSS for the class so that the element is highlighted, the cursor changes to a pointer when you hover over it, and so on.

### Use a Check Box to Select Rows in a Table

Although you can let your users select multiple rows in a table by employing a JET table's built-in multi-row selection capabilities, your users might find it easier to use a check box.

To add a checkbox, base your table on an \texttt{ArrayDataProvider} rather than a \texttt{ServiceDataProvider}. This allows you to access the values in the rows when you need to determine which ones were selected. If you are going to use the context, enter a standard lead-in sentence.

1. Add a boolean attribute to your \texttt{ArrayDataProvider} to track whether a row is selected. Selecting a row will set the value of this attribute to \texttt{true}. 

2. Use a `vbEnter` event to define an action chain to fetch the array data for your table.
3. For the check box, install the Boolean Checkbox component, available in the Component Exchange, in your application. Drag the component onto your table's empty Selected column, and connect the value of the new boolean field you introduced.
The HTML code looks like this:

```html
<oj-ext-checkbox-switch :id="['oj-ext-checkbox-switch-467799112-1-' + $current.index]" value="{{ $current.row.selected }}"></oj-ext-checkbox-switch>
```

4. Create a button to process the rows. Use another action chain with a **For Each** loop and an **If** logic action to loop over the records in the table and determine the value in the field, using an expression like this:

```javascript
{{ $page.variables.DeptADP.data[$current.index].selected }}
```

You can now test the application to see the notifications appear when you select check boxes and click the button to process the rows.