Oracle® Cloud

Developing Applications with Oracle Visual Builder in Oracle Integration Generation 2

24.10.0
F94592-03
August 2024

Oracle Cloud Developing Applications with Oracle Visual Builder in Oracle Integration Generation 2, 24.10.0

F94592-03

Copyright © 2018, 2024, Oracle and/or its affiliates.

Primary Author: Oracle Corporation

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

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

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

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

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

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Part I Get Started with Oracle Visual Builder

1 Welcome to Oracle Visual Builder

What is Oracle Visual Builder? 1-1
Access Oracle Visual Builder 1-2

2 Create Visual Applications

Typical Visual Application Workflow 2-1
Create a Visual Application 2-2
- Create a New Visual Application 2-3
- Create a Copy of an Application 2-7

Export and Import Visual Applications 2-8
- Export a Visual Application 2-9
- Import a Visual Application 2-10
- About Classic Applications 2-12
- Import Classic Applications 2-13

Add Web (and Mobile) Apps to Your Visual Application 2-15
- Create a New Web Application 2-16
- Import an Existing Mobile Application 2-21

Tour the Designer 2-21
Common Tasks for Visual Applications 2-29
- Manage Visual Application Settings 2-30
- Add Team Members 2-31
- Export and Import Application Resources 2-33
  - Export Application Resources 2-33
  - Import Application Resources 2-34
  - Upgrade Imported Resources 2-35

3 Anatomy of Visual Applications

Understand Variables 3-2
- Variables and Parameter Passing 3-3

Expressions 3-4
Variables and Lifecycles 3-5
Variables and Events 3-5
Understand Actions and Action Chains 3-6
Action Chain Context and Contract 3-6
Built-in Actions 3-7
Event Handling for Action Chains 3-7
Understand Page Flows and Lifecycles 3-7
The Lifecycle of a Page 3-8
Page Navigation 3-9
Understand UI Components 3-9
The Component Contract 3-9
Properties 3-10
Events 3-10
Child Slots 3-10
Methods 3-10
Component IDs and Styles 3-10
Understand Data Access Through REST 3-10
Data Binding 3-11
Mapping to and from REST 3-11

Part II Connect Applications to Data

4 Work With Business Objects

About the Business Objects Pane 4-1
Create and Edit Business Objects 4-2
Create a Business Object 4-3
Add Fields to Business Objects 4-4
Edit Business Object Fields 4-4
Change a Field's Data Type 4-8
Set a Default Value for a Field 4-9
Add a Formula to a Field 4-10
Add a Field for Aggregating Data 4-12
Index a Field 4-15
View, Create, and Edit Business Object Relationships 4-15
Create a Many-To-One or One-To-One Relationship 4-15
Create a Many-to-Many Relationship 4-18
Edit a Business Object Relationship 4-19
Secure Business Objects 4-21
Create Rules for Business Objects 4-21

About Adding Business Rules 4-22
Access the Current User's Details in Your Groovy Script 4-23
Triggers for Business Objects 4-23
Object Triggers 4-23
Field Triggers 4-29
Add an Action to Send Email Notifications 4-31
Convert a Trigger to Editable Code 4-34
Build Conditions for Triggers 4-35
Object Validators for Business Objects 4-37
Field Validators for Business Objects 4-38
Object Functions for Business Objects 4-40
Log Diagnostic Messages From Your Scripts 4-42
Work with Endpoints to Access Business Objects 4-42
Access a Business Object's Resource APIs and Endpoints 4-43
Work With Query Parameters 4-45
Add or Remove Exposed Endpoints 4-46
View and Edit Data in Business Objects 4-48
Edit the Data in Business Objects 4-48
Import Data to a Business Object 4-49
Reload Data from Application Sources 4-50
Work with the Data Manager 4-51
Manage Data During the Development Lifecycle 4-51
Import Data From a File Using the Data Manager 4-52
Import Data From a Database 4-54
Export the Data to a File from the Data Manager 4-55
Resolve Problems When Importing Data 4-55
Import and Export Data From the Command Line 4-56
Import Data from the Command Line 4-56
Export Data from the Command Line 4-58
Create Business Objects From a File 4-59
Set Your Own Audit Fields For Imported Business Objects 4-63
Optimize Business Object Performance 4-64
Override Default Timeout for Groovy Scripts 4-64
Enable Polling for Endpoint Requests 4-65
Control Data Caching for Business Objects 4-66
Data Caching Options 4-66
Define a Data Caching Strategy 4-67
Index a Field to Speed Up Searches and Sorts 4-68
Work with Business Object Diagrams 4-68
Create Business Objects with the Diagrammer 4-70
Create Relationships with the Diagrammer 4-71
Switch to Your Own Database Schema for Business Objects 4-72

Create a Business Object Based on a DB Table or View 4-73
Add Fields to a Business Object Based on a DB Table or View 4-80
Change the Data Type of a Field Based on a DB Column 4-81
Create Calculated Fields for a Business Object Based on a DB Table or View 4-82
Set a Field for Auditing 4-82
Use a Sequence for a Primary Key Field 4-84
Switch Schemas Used During an App's Lifecycle 4-84

5 Work With Services

Manage Backends in Your Visual Application 5-1
What Are Backends? 5-3
What Are Application Profiles? 5-4
Set the Backend's Authentication Method and Connection Type 5-6
How Does the Identity Propagation Authentication Method Work? 5-8
How Does the Fixed Credentials Authentication Method Work? 5-9
What is CORS? 5-12
Use an Appropriate Connection Type to Handle CORS for REST Services 5-12
Work with HTTP-based Endpoints 5-14
Allow Anonymous Access to the Service Data 5-14
OAuth and JWT Token Validity 5-15
Create a Backend 5-15
Add Server Variables for Backends 5-16
Create a Custom Backend 5-18
Create a Child Backend 5-21
Edit a Backend 5-23
Add Transforms 5-24
Manage Service Connections 5-34
What Are Service Connections? 5-34
Service Connections: Static Versus Dynamic 5-35
Create a Service Connection 5-39
Create a Service Connection from a Catalog 5-40
Create a Service Connection from a Service Specification 5-47
Create a Service Connection from an Endpoint 5-51
Edit a Service Connection 5-56
Add a Server to a Service Connection 5-57
Edit Service Endpoints for a Static Service Connection 5-58
Retrieve Service Metadata for a Dynamic Service Connection 5-60
Add Server Variables for Service Connections 5-61
Add More Endpoints to a Service Connection 5-62
Edit a Server's Authentication Details 5-66
Add Transforms to a Service Connection or an Endpoint 5-67

Convert a Service Connection (Static to Dynamic or Dynamic to Static) 5-67
Test Service Connection Responses 5-70
Update Schema of the Request or Response 5-74
Connect to Oracle Cloud Services 5-76
Connect to Oracle Cloud Applications APIs 5-76
Connect to Oracle Cloud Applications APIs With User Propagation for Authenticated Flows 5-76
Connect to Oracle Cloud Applications APIs Not in the Catalog Using Fixed Credentials 5-77
Connect to Oracle Integration APIs 5-78
Connect to Oracle Integration APIs Using Identity Propagation 5-78
Connect to Oracle Integration APIs Using Fixed Credentials 5-79
Connect to Oracle Cloud Infrastructure Process Automation APIs 5-81
Connect to Process Cloud APIs 5-83
Connect to Process APIs Using Identity Propagation 5-83
Connect to Process APIs Using Fixed Credentials 5-84
Connect to Oracle Content Management REST APIs 5-85
Connect to Oracle Content Management REST APIs Using Identity Propagation 5-85
Connect to Oracle Content Management REST APIs Using Fixed Credentials 5-86
Access Data in an Existing Oracle Database Using ORDS 5-87
Map ORDS Endpoints in Visual Builder 5-88
Sort, Query, and Paginate ORDS Data 5-88
Connect to ORDS APIs Using Fixed Credentials 5-89

Part III Develop Applications

6 Develop Your Application

How Are Applications Structured? 6-2
Which Editor Do I Use? 6-4
What Are Scopes? 6-6
Manage App Settings 6-6

7 Work With Pages and Flows

Use the Page Designer 7-1
The Components Palette 7-6
The Structure View 7-8
The Data Palette 7-10
The Properties Pane 7-14
Variables Picker 7-15
Expression Editor 7-16

Create and Manage Pages 7-19
Manage Page Settings 7-20
Customize Page Headers 7-21
Set a Page's Layout 7-24
Create Pages From Templates 7-28
Create Pages From Patterns 7-28
Create Pages From Fragments 7-30
Change Page Templates 7-32
Create and Manage Flows 7-33
Manage Flow Settings 7-34
Embed a Flow Within a Page 7-36
Customize Your App's Root Page 7-37
Edit an App's Header, Footer, and Navigation Items 7-38
Add a Navigation Item for Navigation Drawer Apps 7-42
Navigate Between Pages and Flows 7-44
Navigate Between Pages in the Same Flow 7-44
Navigate Between Pages in Different Flows 7-46
Navigate Between Flows in the Root Page 7-48

8 Work With Components

Add a Component to a Page 8-1
Opt In to JET Core Pack Components 8-5
Add a Component Using Code Completion 8-6
Bind a Component to Data 8-8
How Do Quick Starts Work? 8-9
Add Data to a Table or List 8-11
Add a Create Page With a Quick Start 8-13
Add an Edit Page With the Quick Start 8-15
Add a Details Page With the Quick Start 8-18
Quick Starts for Dynamic Forms and Tables 8-19
Add an Image to a Page 8-20
Add an Icon Component to a Page 8-21
Add a Camera Component to a Page 8-22
Filter Data Displayed in a Component 8-27
Filter Data by Filter Criteria 8-28
Filter Component Data by Text 8-30
Filter Component Data by URL 8-32
Sort Data Displayed in a Component 8-34
Use Conditions to Show or Hide Components 8-37
Use Converters to Change Date and Time Formats and Time Zones 8-41
Add Dynamic Components to Pages 8-46

What are Dynamic Components? 8-46
How to Create Layouts With Dynamic Components 8-47
Add a Dynamic Table to a Page 8-48
Add a Dynamic Form to a Page 8-52
Add Display Logic to Determine What's Displayed at Runtime 8-56
Create a Layout for a Dynamic Table or Form 8-61
Preview Different Layouts 8-65
Responsive App Display Logic Example 8-67
Define Custom Contexts for Components in a Layout 8-68
Use Field and Form Templates 8-71
Control How a Field is Rendered with Field Templates 8-72
Apply a Template to a Field 8-74
Start an Action Chain from a Field 8-77
Control How a Form Layout is Rendered 8-80
Apply a Template to a Form 8-84
Add and Group Fields in Dynamic Form Layouts 8-86
Use Business Rules With Your Rule Sets 8-88
What Are Business Rules? 8-89
Work with Business Rules 8-91
Display Messages When Conditions Are Met 8-120
Edit a Field's Properties 8-126
Set a Field to be Read Only 8-126
Set How User Assistance is Rendered in a Layout 8-128
Set a Field as Required 8-130
Use Conditions to Show or Hide Fields in a Layout 8-133
Configure How Columns Render in a Dynamic Table's Layout 8-134
Set a Field to Display as a Text Area in a Form 8-135
Add Converters and Validators to a Field 8-138
Add a Dynamic Container to a Page 8-141
Add Fragments as Sections in a Dynamic Container 8-151
Re-Order a Dynamic Container's Content 8-156
Guidelines for Working with Sections 8-156
Change a Dynamic Container's Layout 8-156
Create Fields For a Layout 8-159
Create a Calculated Field 8-160
Create a Virtual Field 8-163
Work With Custom Web Components 8-165
Work with the Component Exchange 8-167
Get Components From the Component Exchange 8-168
Update a Component from the Component Exchange 8-169
Uninstall a Component 8-170
Import a Web Component Archive 8-170

Create a Web Component 8-172

9 Work with Variables and Types

What are Variables and Types? 9-1
Create Variables in Artifacts 9-6
Enable Variables as Input Parameters 9-9
Track Variables to Detect Unsaved Changes 9-11
Create Variables to Temporarily Store Data Changes in a Buffer 9-13
Create Types 9-15
Create a Custom Object or Array 9-16
Create a Type From an Endpoint 9-18
Create a Type From Code 9-19
Service Data Provider 9-24
Creating a Custom Fetch Action Chain - An Example 9-24
Delay Display of SDP Data 9-33
Speed Up Sorts on Business Object Fields 9-36

10 Work With JavaScript Action Chains

About Action Chains 10-3
About the Action Chain Editor 10-5
Create Action Chains in Design Mode 10-7
Create Action Chains in Code Mode 10-12
About the Action Chain Code 10-14
Visually Create an Action Chain 10-17
Built-In Actions 10-31
Add an Array Operation Action 10-33
Add an Assign Variable Action 10-34
Use Filter Builder to Create Filter Criteria for an SDP 10-36
Filter Builder's Code Editor 10-39
Add a Call Action Chain Action 10-39
Add a Call Component Action 10-42
Add a Call Function Action 10-42
Add a Call REST Action 10-44
Service Definitions 10-48
Transform Functions 10-48
Add a Call Variable Action 10-48
Add a Code Action 10-49
Add a Fire Data Provider Event Action 10-49
Add a Fire Event Action 10-52
Add a Fire Notification Action 10-54

Add a For Each Action 10-56
Add a Get Dirty Data Status Action 10-57
Add a Get Location Action 10-65
Add an If Action 10-66
Add a Login Action 10-67
Add a Logout Action 10-68
Add a Navigate Back Action 10-69
Add a Navigate To Flow Action 10-69
Add a Navigate To Page Action 10-71
Add an Open URL Action 10-73
Add a Process Action 10-74
Add a Reset Dirty Data Status Action 10-74
Add a Reset Variables Action 10-75
Add a Return Action 10-75
Add a Run In Parallel Action 10-76
Add a Scan Barcode Action 10-78
Add a Share Action 10-80
Add a String Operation Action 10-81
Add a Switch Action 10-82
Add a Try-Catch Action 10-84
Custom Actions 10-86
Create a Custom Action 10-86
Create the Action Files 10-86
Add the Metadata 10-88
Add the Code 10-92
Specify Path to Code 10-94
Start an Action Chain 10-94
Start an Action Chain From a Component 10-94
Start an Action Chain When a Variable Changes 10-98
Start an Action Chain From a Lifecycle Event 10-99
Start an Action Chain By Firing a Custom Event 10-102
Test Action Chains 10-108
Create a Test for a Test Case 10-110
Run the Tests 10-115
Use the Tests Footer in a Visual Application 10-116
Test Action Chains Using the vb-test Grunt Task 10-117

11 Work With JSON Action Chains

About Action Chains 11-2
Create an Action Chain 11-3
Built-in Actions 11-10

Add an Assign Variables Action 11-12
Filter Builder's Code Editor 11-16
Add a Call Action Chain Action 11-17
Add a Call Component Action 11-19
Add a Call Function Action 11-21
Add a Call REST Action 11-22
Add a Call Variable Action 11-26
Add a Fire Data Provider Event Action 11-27
Add a Fire Event Action 11-29
Add a Fire Notification Action 11-29
Add a Get Location Action 11-31
Add a Reset Variables Action 11-32
Add a Login Action 11-33
Add a Logout Action 11-33
Add a Scan Barcode Action 11-34
Add a Take Photo Action 11-37
Add a Share Action 11-38
Add a For Each Action 11-39
Add an If Action 11-42
Add a Return Action 11-44
Add a Run In Parallel Action 11-45
Add a Switch Action 11-46
Add a Navigate Action 11-47
Add a Navigate Back Action 11-50
Add an Open URL Action 11-50
Custom Actions 11-51
Create a Custom Action 11-51
Create the Action Files 11-52
Add the Metadata 11-53
Add the Code 11-57
Specify Path to Code 11-59
Test Action Chains 11-59
Manage All Tests in a Visual Application 11-61
Test Action Chains Using the vb-test Grunt Task 11-62
Start an Action Chain 11-63
Start an Action Chain From a Component 11-63
Start an Action Chain When a Variable Changes 11-67
Start an Action Chain From a Lifecycle Event 11-68
Start an Action Chain By Firing a Custom Event 11-71

12 Work With Events and Event Listeners

Define Events in Your Application 12-1
Create Event Listeners for Events 12-3
Choose How Custom Events Call Event Listeners 12-7
Raise Fragment or Layout Events that Emit to the Parent Container 12-8

13 Work With Application Resources

Import Resources 13-2
Manage Custom Component, CSS, and Module Imports 13-4
Work with the Image Gallery 13-6

14 Work With Code

Work With JavaScript 14-1
- Add a Custom JavaScript Function 14-2
- Use Variables with a JavaScript Module 14-6
- Work With Third-Party JavaScript Libraries 14-6
- Use RequireJS to Reference Third-Party JavaScript Libraries 14-7

Work With JSON 14-8
Trigger Code Insight 14-10
Manage Code Editor Settings 14-10

15 Work With the Diagram View

View a Flow's Navigation in Diagram View 15-2
Add Pages and Action Chains to a Flow in Diagram View 15-3
- Add a Page in the Flow Diagram 15-4
- Create an Action Chain in the Flow Diagram 15-6
- Bind an Action Chain in the Flow Diagram to an Existing Event Listener 15-9
- Show or Hide an Action Chain in the Flow Diagram 15-10

16 Work With Fragments

Create and Add a Fragment to a Page 16-1
Manage Fragment Settings 16-7
Reuse a Fragment 16-9
Pass Data Between a Fragment and Its Parent Container 16-10
- Enable Fragment Variables as Input Parameters 16-11
- Enable Page Variables to Provide Initial Values for a Fragment's Input Parameters 16-15
- Automatically Write Back a Fragment Variable's Value to Its Container Variable 16-19
- Automatically Create and Wire a Fragment Variable on Its Container 16-21

Sample Scenario: Create a Fragment and Pass Values 16-24
Create Custom Events that Emit to a Fragment's Parent Container 16-36
Set the Binding Type for Variables in Dynamic Components 16-45
Pass a Fragment's Context to VDOM or Custom Web Components 16-46
Defer Rendering of a Fragment's Content 16-47
Add Slots to a Fragment 16-50
Add Default Content to a Fragment Slot 16-56
Set Data Context for a Fragment Slot 16-56
Customize How Fragment Properties Display in the Properties Pane 16-59
Customize How a Fragment Variable is Displayed in the Properties Pane 16-59
Section Fragment Properties for Display in the Properties Pane 16-82

Part IV Augment Applications

17 Enable Progressive Web App Support

Guidelines for Using PWA Support 17-1
Configure Progressive Web App Support 17-2
Deep Linking on Android 17-6
Run Mobile Applications as PWAs 17-7
Configure Mobile Application Settings 17-8
Build a Mobile Application as a PWA 17-9
Convert a Mobile PWA to a Web PWA 17-12

18 Secure the Application

Security for Web Apps 18-1
Authentication Roles Versus User Roles 18-2
Manage User Roles and Access 18-3
Test Role-Based Access 18-6
Access and Secure Business Objects 18-7
Secure Business Objects 18-7
Allow External Access to Your Business Objects 18-10
Get an Access Token for Authentication 18-12
Allow Anonymous Access 18-13
Embed a Web Application 18-16

19 Add Offline Capabilities to Your Application

Add Offline Support Using the Oracle Offline Persistence Kit 19-1

20 Optimize Your App for Search Engines

Create a Sitemap for a Web App 20-1
Add a Sitemap to a Web App's Resources 20-2
Warm the Cache for URLs in a Sitemap 20-5
Move Your Sitemap to a Visual Application's Root Directory 20-6

21 Work With Business Processes

About Using Processes in Your Application 21-1
Add a Business Process to an Application 21-2
Modify a Business Process Alias 21-3
Navigate to Your Process Instance 21-4
Add a Process to a Page 21-5
Add a Process to an Action Chain 21-7
Start and Complete Processes from Visual Builder 21-10
Configure the Connection to a Process Server 21-10
Process Server Authentication Options 21-13

22 Work With Translations

About Translation Resources 22-1
Understand the Structure of Translation Bundles 22-1
Understand Translation Keys for Display Texts 22-3
    Generate Translation Keys for Display Texts 22-4
Download Bundles for Translation 22-6
Use Translation Strings in JavaScript Files 22-8
Upload Translated Files 22-10
Create Translation Bundles 22-12

Part V Manage Applications

23 Manage Your Visual Application

View an Application's Status 23-2
Create a New Version of an Application 23-3
Delete a Visual Application 23-5
Restore a Deleted Application 23-7
Manage Applications Created in Visual Builder Studio 23-8

24 Integrate Your Visual Application With a Git Repository

Add Credentials for Your Oracle Visual Builder Studio Account 24-1
Link Your Visual Application to a Git Repository 24-2
Pull Files From Your Git Repository 24-2
Push Your Changes to Your Git Repository 24-3
Change the Local Branch HEAD in a Linked Git Repository 24-3

25 Test and Debug Applications

Audit Application Code 25-2
Preview an App in Debug Mode 25-6
Troubleshoot Build Issues 25-7
Debug Business Objects 25-7
- Enable Tracing to Monitor Endpoint Calls 25-8
  - View Trace Details 25-9
  - Manage Tracing to Control Disk Usage 25-11
  - Export and Import a Trace File 25-11
- Enable Logging for Scripting Events 25-12

Change an Application's Log Level 25-13

26 Stage and Publish Visual Applications

What Happens When You Stage and Publish Visual Applications? 26-1
Stage a Visual Application 26-2
Publish a Visual Application 26-5
View Database Schemas Used During an App's Lifecycle 26-7
Specify a Custom App URL 26-9
Update a Published Visual Application 26-11
Roll Back Application to the Previously Published Version 26-13

27 Manage Runtime Dependencies for Visual Applications

Upgrade Your App 27-2
- After Upgrading 27-4
- Set a Custom Version 27-5
- Understand What's Happening in visual-application.json 27-6

Resolve Upgrade Issues 27-7
What Happens During Software Maintenance? 27-8

28 Optimize Your Builds and Audit Your Code Using Grunt

Overview 28-1
Build Your Application Using Oracle Visual Builder Studio 28-2
Build Your Application Locally 28-2
- Build and Deploy Your Application 28-2
- Authentication 28-4

Grunt Tasks to Build Your Visual Application 28-5
- About Visual Builder Grunt Build Tasks 28-5
- vb-clean 28-6
- vb-process-local 28-6
- vb-deploy 28-12
- vb-optimize-cdn 28-14
- vb-optimize 28-15
- vb-prepare 28-15
- vb-prerender-cache-warm 28-16
- vb-test 28-18
- vb-require-bundle 28-19
- vb-require-bundle-clean 28-21
- vb-css-minify 28-21
- vb-image-minify 28-22
- vb-json-minify 28-22
- vb-export 28-22
- vb-manifest 28-23
- vb-package 28-24
- vb-archive 28-25
- vb-process-raw 28-26
- vb-process-raw-index-html 28-26
- vb-application 28-26
- vb-serve 28-28
- vb-pwa 28-29
- vb-fa-generate-base-app-config 28-30
- vb-pwa-splashscreen 28-31
- vb-watch 28-32

Customize Your Grunt Build Process 28-33
- Add Custom Functionality to Existing Tasks 28-33
- Override Existing Grunt Tasks 28-33
- Optimize a Specific Web Application 28-34
- Host an Application on a Content Delivery Network (CDN) 28-35
- Run and Configure a Multitask 28-35
- Customize Bundle Modules 28-36
- Exclude Resources from a Bundle 28-37

Specify Options of Non-multitasks 28-41
Specify Options for All Tasks 28-41
Audit Your Application Using the vb-audit Grunt Task 28-42
Configure Audit Options in Gruntfile.js 28-43
Override Configuration Options in Gruntfile.js 28-45

Part VI Use Cases & Troubleshooting

29 Common Use Cases

Work With Code Samples 29-1
Change an Application's Logo 29-2
Style and Theme Visual Builder Applications 29-3
Transition a Web (or Mobile) App's Theme to Redwood 29-4
Customize a Web App's Appearance 29-5
Override the Redwood Theme for a Mobile Application 29-9
Add a Custom Style to a Component 29-10
Add Login and Logout Capabilities to an Application 29-11
Create a Custom Lock Page 29-14
Apply a Custom Lock Page to a Visual Application 29-15
Apply a Custom Lock Page to a Web Application 29-16
Use a SOAP Web Service With Visual Builder 29-18
Run Visual Builder Applications On Other Servers 29-18
Embed a Web App in an Oracle Cloud Application 29-19
Make Your Web App Ready for Embedding 29-19
Embed a Web App Using Page Composer 29-21
Embed a Web App Using Application Composer 29-24
Call Server-side Functionality from Visual Builder 29-25
Add the Oracle Digital Assistant to Your Web App 29-25
Abort Pending REST Calls in Visual Builder 29-26
Forms 29-32
Enable Client-Side Validation for a Form 29-32
Validate Dates in Forms 29-34
Tables 29-34
Modify a Table's Default Display 29-35
Reorder Columns in a Table 29-42
Sort Data in Table Columns 29-44
Enable Resizing of a Table Column 29-47
Wrap Table Text 29-47
Add Columns to an Existing Table 29-48
Format Row Values in a Table Conditionally 29-49

Create a Search Filter for a Table 29-50
Create an Editable Table 29-51
Update Pagination Behavior for a Table 29-56
Enable Text Selection in a Table 29-60
Pages and Flows 29-61
Restrict User Access to an Application, Flow, or Page 29-62
Components 29-63
Conditionally Show or Hide UI Components 29-63
Enable Time Zone Specification 29-63
Validate the Length of an Entry in an Input Text Field 29-66
Filter Multiple Attributes in a Search 29-66
Set an Initial Value for the Select (Single) Component 29-67
Business Objects 29-67
Format a Date Field 29-68
Apply an Aggregate Function to a Calculated Field From a Child Business Object 29-68

30 Troubleshooting & FAQs

How Do I Find the URL of My Visual Builder Instance? 30-1
How Do I Find My Application's Runtime Version? 30-2
How Do I Clear My App's Resource Cache? 30-2
How Do I View Details of Client Apps in IDCS? 30-2
How Do I Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null? 30-3
How Do I Resolve Web Component Loader Issues? 30-4
How Do I Resolve a 'Method Not Allowed' Error? 30-4
How Do I Resolve a 'No Such File' Error for the URI.js File? 30-5
How Do I Exclude a JavaScript Library from a Packaging Job 30-5
How Do I Fix a Missing Scroll Bar in a Table? 30-6
How Do I Access Components After Upgrading? 30-7
How Do I Set a Custom Content-Security-Policy Header? 30-7
Troubleshooting Access Issues 30-7
How Do I Control the Session Duration For My Visual App? 30-8
Why Does a Live App That Allowed Anonymous Access Prompt for Login? 30-8
How Can I Recover Apps Linked to a Deleted User Account? 30-9
Troubleshooting Service Connections 30-9
Why Was a Certificate in the Remote Path Reported as Invalid? 30-9
How Do I Resolve an "Unknown Host" Error? 30-10
How Do I Resolve a "Cannot Process Service Scope" Error? 30-10
How Do Timeout Settings Affect External REST Calls in a Visual App? 30-11
How Do I Resolve a CORS Error? 30-11
Troubleshooting Business Objects 30-12

What Is The Maximum Data Limit for Business Objects? 30-12
How Do I Resolve Database Connection Problems? 30-12
How Do I Resolve a "Failed to verify the target database" Error When Switching the Tenant's Database 30-13
How Can I Access Business Object Data? 30-13
How Do I Resolve a "Maximum Number of Sessions Exceeded" Error? 30-14

Preface

This guide describes how to use a web-based visual development tool to create and publish custom applications that can integrate business objects and Oracle Cloud Applications REST services to extend SaaS services.

Topics:

- Audience
- Documentation Accessibility
- Diversity and Inclusion
- Related Resources
- Conventions
- Information About Cookies

Audience

This guide 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.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `https://www.oracle.com/corporate/accessibility/`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `https://support.oracle.com/portal/` or visit `Oracle Accessibility Learning and Support` if you are hearing impaired.

Keyboard Shortcuts

When working in the Page Designer, you can use these keyboard shortcuts to help you move around quickly:

To do this:	Use this on Mac:	Use this on Windows:
Find and open a file in an app	Command-P	Ctrl+P
Find a given string in a file (or in the Code editor)	Command-Shift-F	Ctrl+F
Undo/redo	Command-Shift-Z	Ctrl+Z Except for common Windows shortcuts such as cut, copy, and paste, Visual Builder does *not* support shortcuts like the ones described here.

Keyboard Shortcuts for Code Editors

To help you work efficiently, Visual Builder supports the following keyboard shortcuts when working with code editors in the Designer (HTML, JSON, JavaScript, and Code view in the Page Designer).

Tab behavior shortcut

To do this:	Use this on Mac:	Use this on Windows:
Change the code editor's default behavior for the Tab key; pressing Tab adds four spaces in the editor by default	Control-Shift-M	Ctrl+M

Show Command Palette and Save shortcuts

To do this:	Use this on Mac:	Use this on Windows:
Show Command Palette	F1	F1
Save	Command-S	Ctrl+S

Basic Editing shortcuts

To do this:	Use this on Mac:	Use this on Windows:
Cut line if there is no selection; cut selection if there is one	Command-X	Ctrl+X
Copy line if there is no selection; copy selection if there is one	Command-C	Ctrl+C
Move line up	Option-Up	Alt+Up
Move line down	Option-Down	Alt+Down
Copy line up	Shift-Option-Up	Shift+Alt+Up
Copy line down	Shift-Option-Down	Shift+Alt+Down
Delete line	Shift-Command-K	Ctrl+Shift+K
Insert line below	Command-Enter	Ctrl+Enter
Insert line above	Shift-Command-Enter	Ctrl+Shift+Enter
Jump to matching bracket	Shift-Command-\	Ctrl+Shift+\
Indent line	Command-]	Ctrl+]
Outdent line	Command-[	Ctrl+[
Go to beginning of line	Home or Fn-Left	Home
Go to end of line	End or Fn-Right	End
Go to beginning of file	Command-Home	Ctrl+Home
Go to end of file	Command-End	Ctrl+End
Scroll line up	Command-Up	Ctrl+Up
Scroll line down	Command-Down	Ctrl+Down
Scroll page up	Option-PgUp	Alt+PgUp
Scroll page down	Option-PgDn	Alt+PgDn
Fold (collapse) region	Command-Shift-[	Ctrl+Shift+[
Unfold region	Command-Shift-]	Ctrl+Shift+]
Toggle line comment	Command-/	Ctrl+/

To do this:	Use this on Mac:	Use this on Windows:
Toggle block comment	Shift-Option-A	Shift+Alt+A
Toggle word wrap	Option-Z	Alt+Z

Navigation shortcuts

To do this:	Use this on Mac:	Use this on Windows:
Go to line ...	Command-G	Ctrl+G
Go to symbol ...	Shift-Command-O	Ctrl+Shift+O
Go to next error or warning	F8	F8
Go to previous error or warning	Shift+F8	Shift+F8
Go back	Control--	Alt+Left
Go forward	Control-Shift--	Alt+Right

Search and Replace shortcuts

To do this:	Use this on Mac:	Use this on Windows:
Find	Command-F	Ctrl+F
Replace	Option-Command-F	Ctrl+H
Find next	Command-G	F3
Find previous	Shift-Command-G	Shift+F3
Select all occurrences of Find match	Option-Enter	Alt+Enter
Add selection to find matches	Command-D	Ctrl+D

Multi-cursor and Selection shortcuts

To do this:	Use this on Mac:	Use this on Windows:
Insert cursor	Option-Click	Alt+Click
Insert cursor above	Option-Command-Up	Ctrl+Alt+Up
Insert cursor below	Option-Command-Down	Ctrl+Alt+Down
Undo last cursor operation	Command-U	Ctrl+U
Insert cursor at end of each line selected	Shift-Option-I	Shift+Alt+I
Select current line	Command-L	Ctrl+L
Select all occurrences of current selection	Shift-Command-L	Ctrl+Shift+L
Select all occurrences of current word	Command-F2	Ctrl+F2
Expand selection	Control-Shift-Command-Right	Shift+Alt+Right
Shrink selection	Control-Shift-Command-Left	Shift+Alt+Left
Column (box) selection	Shift-Option- (drag mouse)	Shift+Alt + (drag mouse) Ctrl+Shift+Alt+(arrow key)
Column (box) selection page up	Shift-Option-Command-PgUp	Ctrl+Shift+Alt+PgUp
Column (box) selection page down	Shift-Option-Command-PgDn	Ctrl+Shift+Alt+PgDn

Rich Languages Editing shortcuts

To do this:	Use this on Mac:	Use this on Windows:
Trigger suggestion	Command-Space	Ctrl+Space
Format document	Shift-Option-F	Shift+Alt+F
Go to definition	F12	F12
Peek definition	Option-F12	Alt+F12
Show references	Shift+F12	Shift+F12
Rename symbol	F2	F2

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Resources

For more information, see these Oracle resources:

- Oracle Public Cloud

  `http://cloud.oracle.com`
- Manage Instance Settings in *Administering Oracle Visual Builder in Oracle Integration Generation 2*

Conventions

The following text conventions are used in this document.

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

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:

Name	Description
JSESSIONID	The JSESSIONID cookie is a transient cookie used for session management. It only has a session identifier and does not contain any personal details.
OAMAuthnCookie	The OAMAuthnCookie cookie is generated by Oracle Access Manager for all clients using an Oracle Cloud service. A valid OAMAuthnCookie is required for a session. • 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 Removing the cookie will cause the user to be logged out. The user will need to re-authenticate the next time they request 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.
VBCS_METRICS_<*app name*>_<*app version*>	The VBCS_METRICS cookie is a persistent cookie that expires at midnight on the day it is created and contains a time stamp. The purpose is to 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.

Name	Description
VBCS_HOURLY_METRICS_<*tenant id*>	The VBCS_HOURLY_METRICS is a persistent cookie that expires at midnight on the day it is created. The purpose is to count unique visits per hour, 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.

Part I

Get Started with Oracle Visual Builder

Look here for everything you need to know to start using Oracle Visual Builder.

Topics:

- Welcome to Oracle Visual Builder
- Create Visual Applications
- Anatomy of Visual Applications

1

Welcome to Oracle Visual Builder

With increased demand for modern applications that serve specific business needs, quickly turning ideas into powerful apps that help run your business is critical. Oracle Visual Builder helps you do just that, by providing a cloud-hosted solution that empowers you to create and host applications with ease.

What is Oracle Visual Builder?

Oracle Visual Builder is an intuitive development experience on top of a development and hosting platform that empowers you to create engaging responsive applications. Focusing on ease of use and a visual development approach, it provides an easy way for you to create applications that are hosted in Oracle's secure and scalable cloud platform.

Visual Development Experience

Visual Builder provides simple but powerful visual development tools to create responsive apps—all without the need to install any additional software. This rich set of visual tools help you quickly design your app by dragging and dropping UI components and customizing their attributes to define behavior. While these tools lend themselves to low-code developers, experienced developers can just as easily access the underlying source code, even extend it using standard HTML5, JavaScript, and CSS techniques for complex needs.

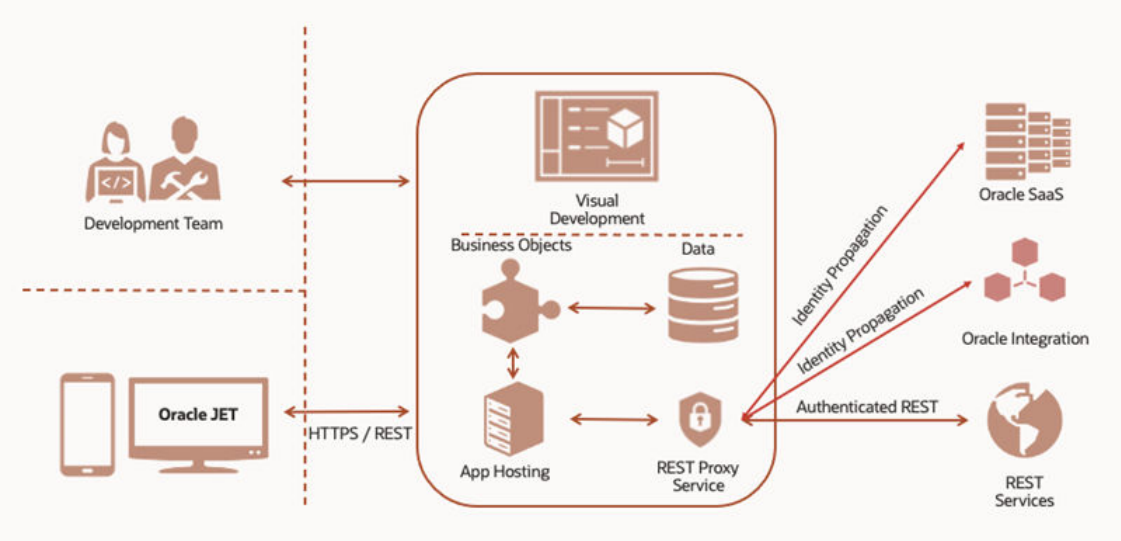
Easy Access to Data

Visual Builder makes it easy to access your app's data through REST-based services. So you can create reusable business objects to implement your app's business logic and store its data, which can then be managed through REST endpoints that Visual Builder generates for you. Or you can pick data objects exposed by Oracle SaaS or Oracle Integration applications in an integrated catalog of REST services. You can also access data from any external REST service with just a few clicks.

Development and Hosting Platform

Visual Builder is a complete development tool as well as a hosting platform, which means you can manage your application's lifecycle right from development to test and final publishing. Version management and data migration are built into an app's lifecycle, making it easy for you to stage and publish your app and manage its data in every phase.

What's more, Visual Builder is a managed service. This means that once you provision a Visual Builder instance, there's very little you need to do beyond developing and publishing your app. Everything the app needs to run successfully (including a web server to host your application and to secure data access) is taken care of. Thus, as a development team, you can take your app from development to stage and publish it in a very short time. Here's a high-level walkthrough of how you'd go about developing an app using Visual Builder:



Your Visual Builder instance (represented by the square in the middle of the image) provides capabilities for your visual application both as a *visual development* tool (at the top) as well as an *app hosting* platform with a built-in web server (indicated by server-side components at the bottom):

- As a visual development tool, Visual Builder provides access to UI components and WYSIWYG interfaces that leverage the open-source Oracle JavaScript Extension Toolkit (JET). This visual environment, known as the Designer, features several visual editors that a development team can use to collaboratively build rich UIs that span multiple devices. It also supports Redwood, the Oracle standard for user experience, that lets you develop apps that provide the same look and feel as apps delivered from Oracle.

  Within this environment, you can develop browser-based responsive apps, including progressive web apps, which combine the on-device mobile experience with a web app's ease of distribution—eliminating the need to download updates from app stores.

- As an app hosting platform, Visual Builder provides various capabilities to publish and run your app in the cloud, including an embedded database that stores your app's *business objects*—essentially Oracle tables with business logic exposed through REST APIs—and their *data*.

  It also includes a *REST proxy service* to manage access to external REST endpoints. When your app's data comes from REST APIs in Oracle catalogs such as Oracle SaaS or Oracle Integration, the proxy service uses server-side integration with the Oracle Identity Cloud Service (IDCS) to manage authentication and authorization (by default) through *identity propagation*. When your app's data comes from other REST endpoints, *authenticated REST* mechanisms are used to manage credentials.

  Together, these components provide the resources required to host your visual app and manage its data.

When your apps are published, they become available to your users in the cloud, from any desktop or mobile device, with communication to the app's underlying JET components secured over HTTPS and REST.

Access Oracle Visual Builder

To develop applications using Oracle Visual Builder, you access the service from the Oracle Integration Home page.

1. When you sign in to Oracle Integration, click **Show/Hide Navigation menu** in the top corner of the Home page.
2. In the navigation pane, click **Visual Builder** to open the Visual Applications Home page.

   You can now use Visual Builder to create and develop visual applications.

> **Note:**
>
> If you see a message that Visual Builder is not enabled for your Oracle Integration instance, talk to your administrator. See Enable Visual Builder in Oracle Integration.

3. To return to Oracle Integration, click the header menu (next to **ORACLE Visual Builder**), then click **Navigate to Home**.

   Once you create apps, they also become available to you on the Oracle Integration Home page. Click the **Visual Applications** tile to access your apps in Visual Builder.

2

Create Visual Applications

As a developer, you use Visual Builder to create *visual applications*, each of which contains web (and mobile) applications that can be deployed to desktop as well as mobile devices. Let's take a look at how you can create these visual applications and add web (and mobile) apps to them.

Typical Visual Application Workflow

A visual application is a collection of resources that you use to develop web (and mobile) apps. It includes metadata in JSON files that describe data sources (business objects and services) as well as the HTML and JavaScript files of your web (and mobile) apps.

To develop your application, you define its data sources and design an interface for users to work with. Visual Builder does not impose any specific order for building your application. 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. However you decide to proceed, you always start with a visual application, which is your ticket into the Designer.

This table provides a high-level description of the tasks that you typically perform when building your application:

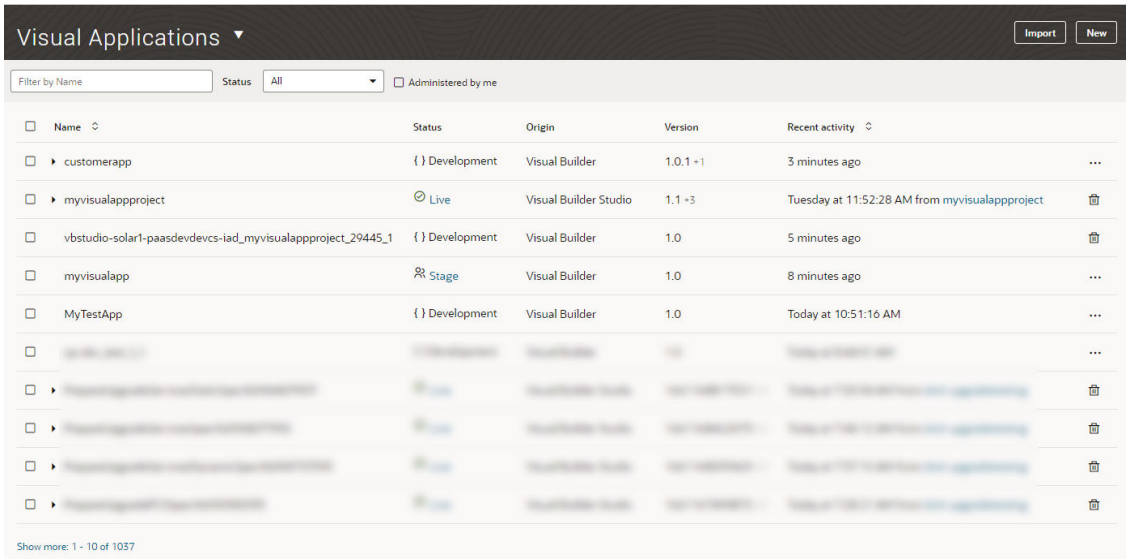
Task	Description	More Information
Create a new visual application	Use the wizard on the Home page to create a new visual application, import an existing visual application as the basis for a new one, or create a new version of an existing visual application	Create a Visual Application Export and Import Visual Applications 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.	What Are Service Connections?
Create business objects	Define your own custom REST endpoints for data in your database based on the needs of your applications.	Create a Business Object
Add web (and mobile) apps	Add web (and mobile) apps to your visual application	Add Web (and Mobile) Apps to Your Visual Application
Develop the web (or mobile) app	Use the Designer's tools to lay out pages and develop your web (or mobile) app.	Develop Your Application
Secure the application	Create application roles and configure the permissions for business objects.	Secure the Application

Task	Description	More Information
Stage and test the application	Use the Application Options menu to stage the application when you are ready to do more thorough testing. You can share staged application's URL with other people for testing.	Stage a Visual Application
Import real data and check the schema	Use the Data Manager to import data into your databases from a file or from the live database.	Manage Data During the Development Lifecycle
Publish the application	Publish the staged version and either import data or use your live database.	Publish a Visual Application

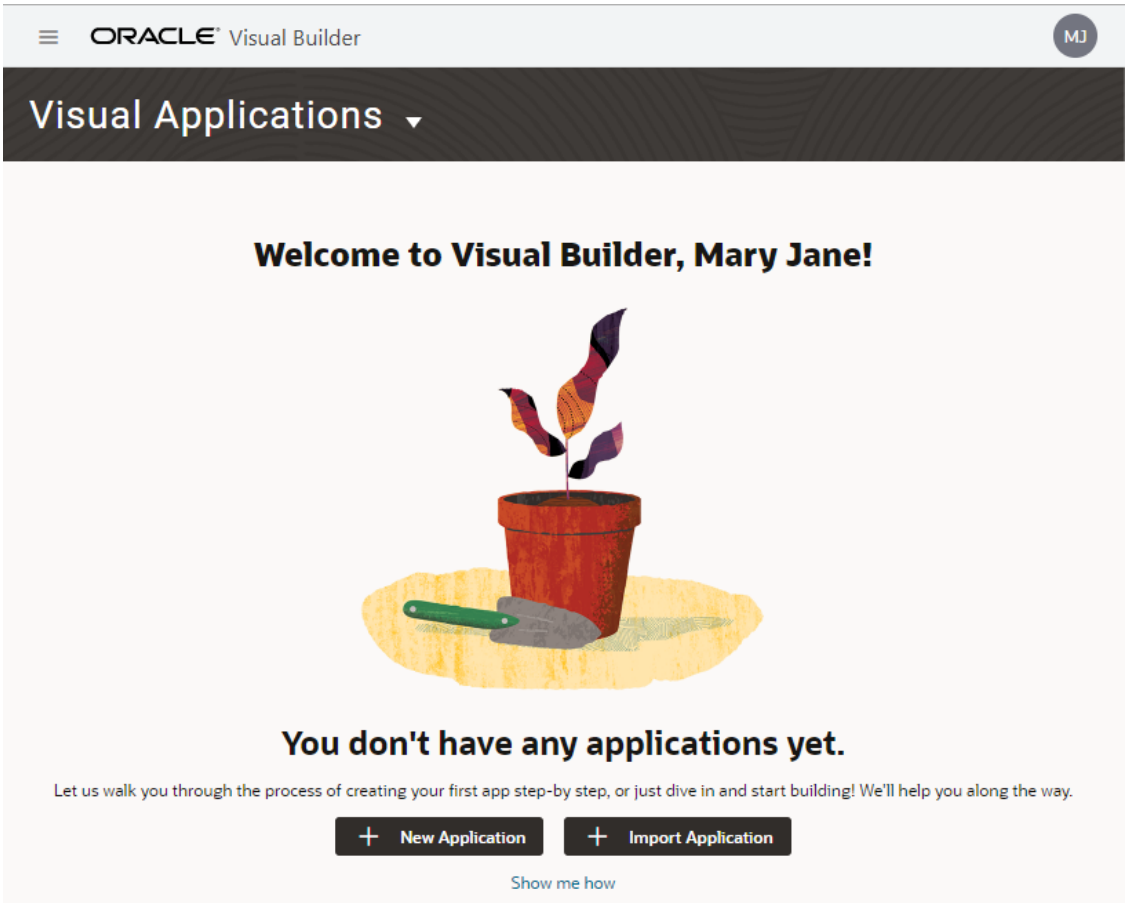
Create a Visual Application

You can create a new visual application and build it from the ground up, or start with an existing application that someone else has already worked on. Either way, you start with a visual application on the Visual Applications Home page.

The Home page is the primary console for creating and managing your applications. It serves as the entry point for your applications and is the first page you see whenever you log in to Visual Builder.



If you don't have any applications when you log in, you'll see the landing page as shown here (you might see additional options if all your apps were previously deleted or if you're an admin user):

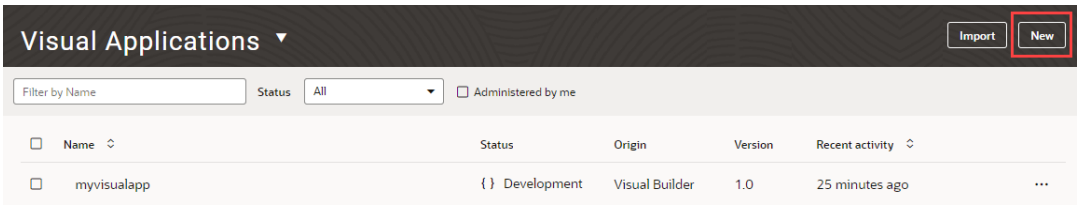


To get started, you need to create a new application, import an existing one, or have someone add you as a team member to their application. You can then manage the application from the Home page.

Create a New Visual Application

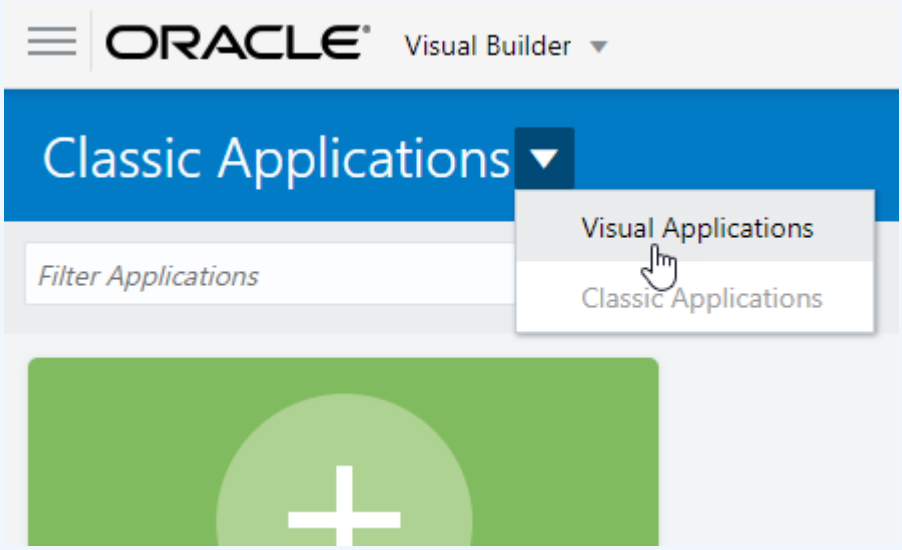
Create a visual application when you want to start building an application from scratch.

1. On the Visual Applications Home page, click **New**:



Note:

If you don't see the Visual Applications Home page, select **Visual Applications** in the menu on the Home page:



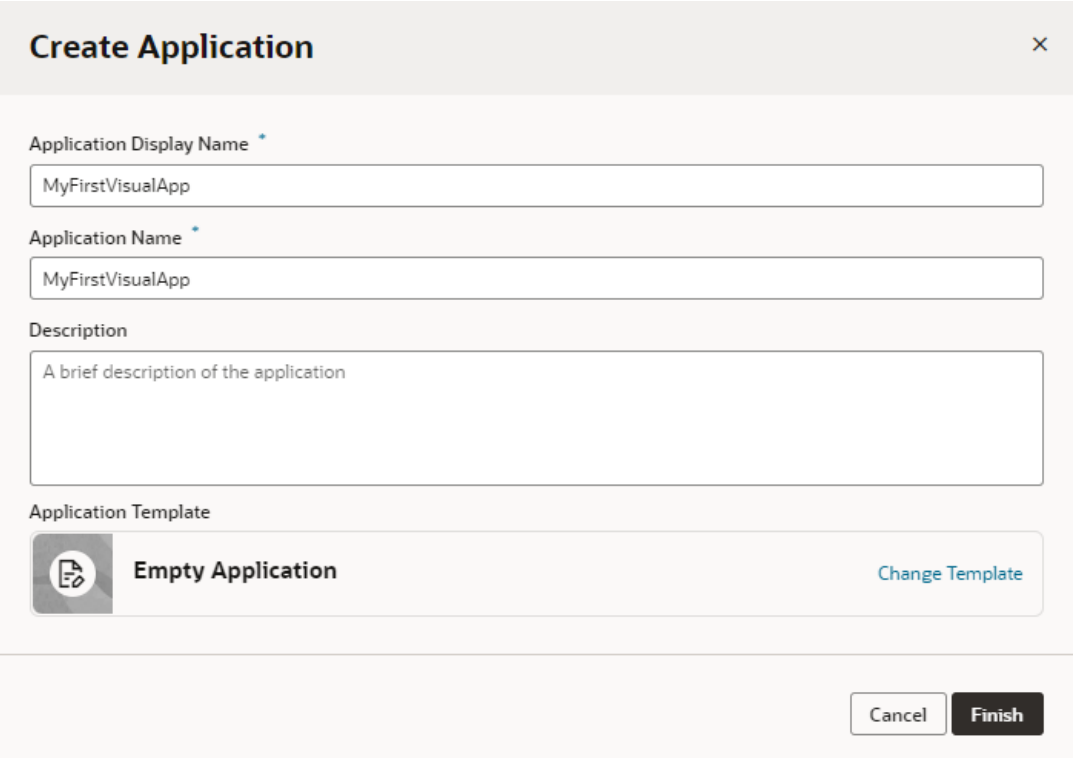
2. Enter the Application Display Name (for example, `MyFirstVisualApp`) in the Create Application dialog. The Application Name is automatically populated based on the display name you provide. You can change this if you want, but the name must be unique in your identity domain.

   A starting version is added to the application name behind the scenes to construct the complete application ID, something like `MyFirstVisualApp-1.0`, which you can view later on in the Settings editor.

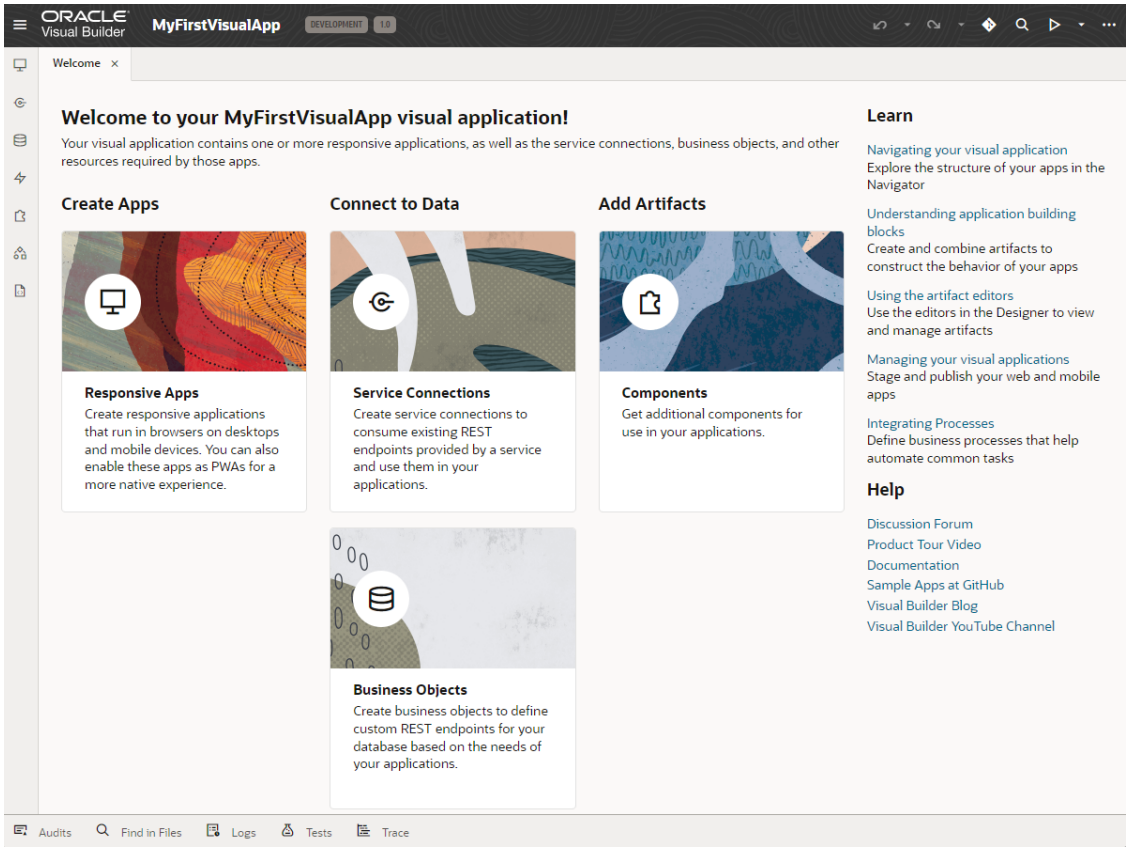
   The display name, name, and ID can't be changed once the application is created, so make sure you provide values you want to keep.

3. Optionally, provide a description.

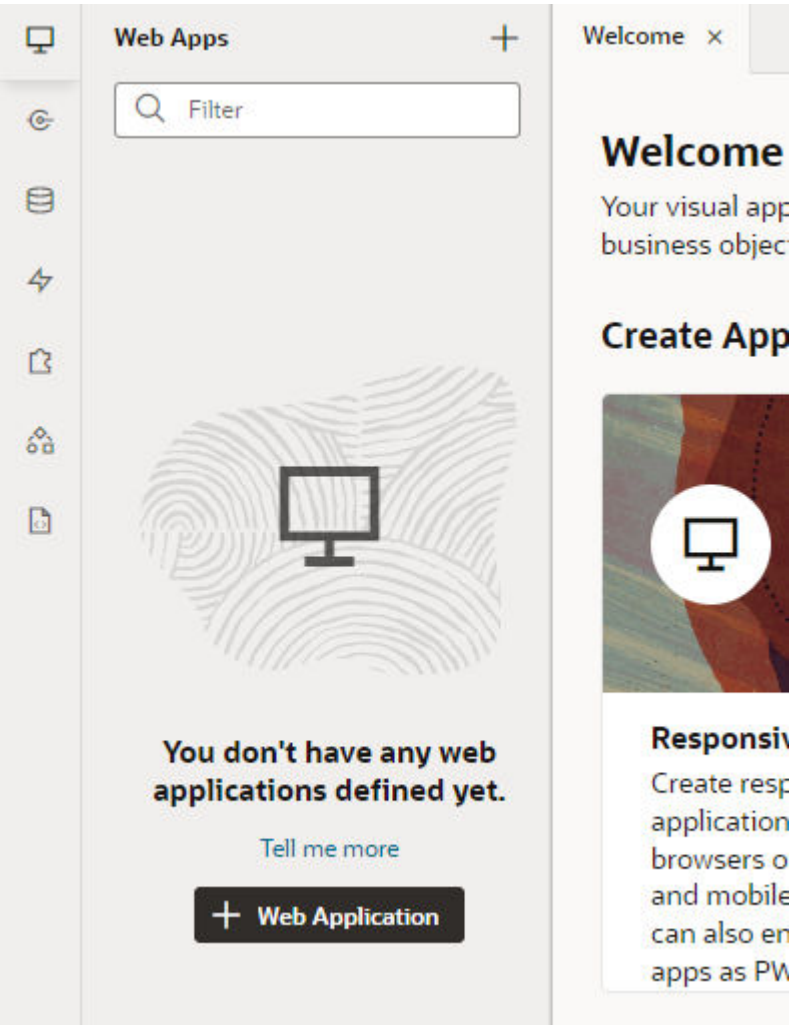
4. If you want to use the default Empty Application template that does not create any apps, artifacts, or other resources, click **Finish**. Otherwise, click **Change Template**, select the template you want to create resources or apps that already include artifacts, then click **Finish**.



Your new visual application opens in the Designer, on the Welcome screen. Your new application does not contain any artifacts, but the application's file structure and some resources are created for you by default. You can use the Welcome screen to help you decide which artifacts you want to create first:



Click any tile in the Welcome screen to open the corresponding panel in the Navigator, where you can create and manage the artifacts. For example, if you click Responsive Apps, the Web Apps tab opens in the Navigator:



> **Note:**
>
> All users who want to collaborate on the app, including admins, must be added to the app as a team member. See Add Team Members.

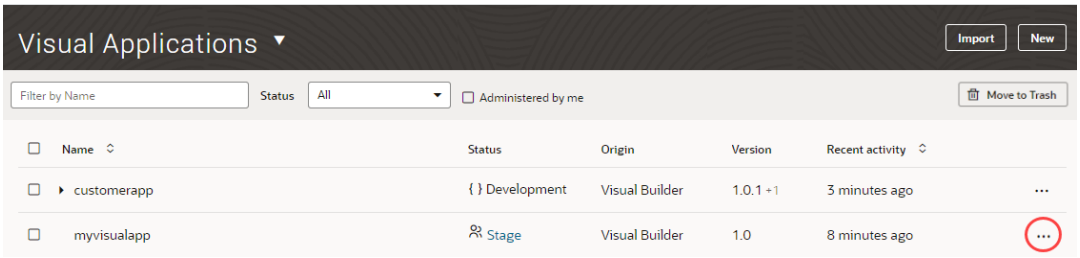
Create a Copy of an Application

You can duplicate an existing application to create a copy that you can work with. A application copy contains all the resources and database schema of the source, but will have a different name, application ID, and URI.

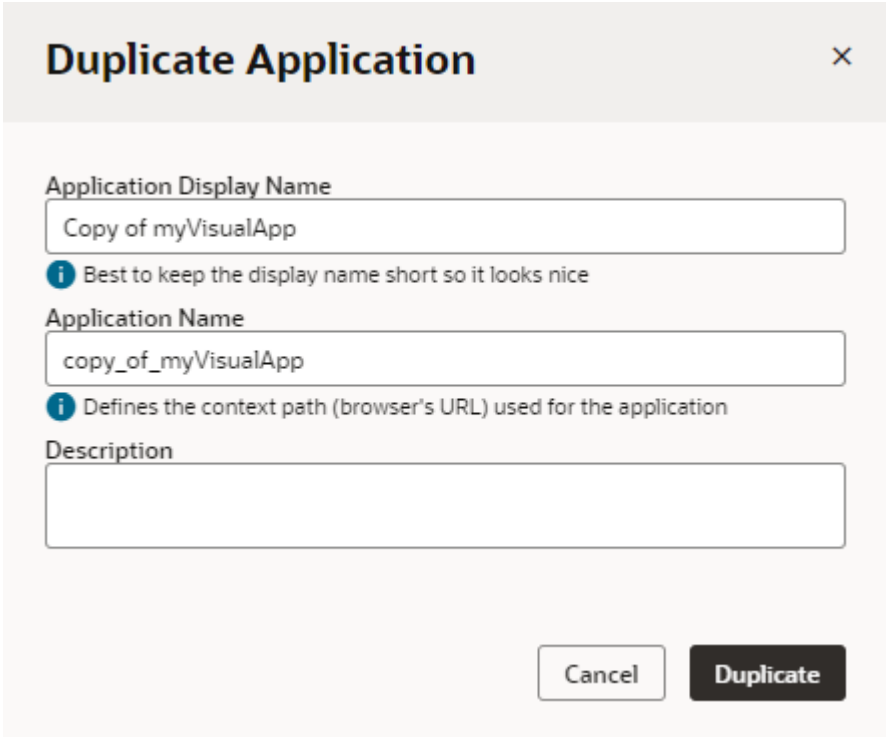
To create a copy of an application:

1. On the Visual Applications Home page, locate the version of the application that you want to copy.

2. Open the Application Options menu, then select **Duplicate**:



3. Enter the Application Display Name and Application Name in the Duplicate Application dialog box.



A starting version is added to the application name behind the scenes to construct the complete application ID, something like `MyNextVisualApp-1.0`, which you can view later on in the Settings editor. The application ID will be exposed as part of the browser URL used to access the application.

4. Click **Duplicate**.

Export and Import Visual Applications

You can export a visual application as an archive to your local system, then import the archive to create a new visual application. Use the import and export mechanism to share source files and to move applications between instances.

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.

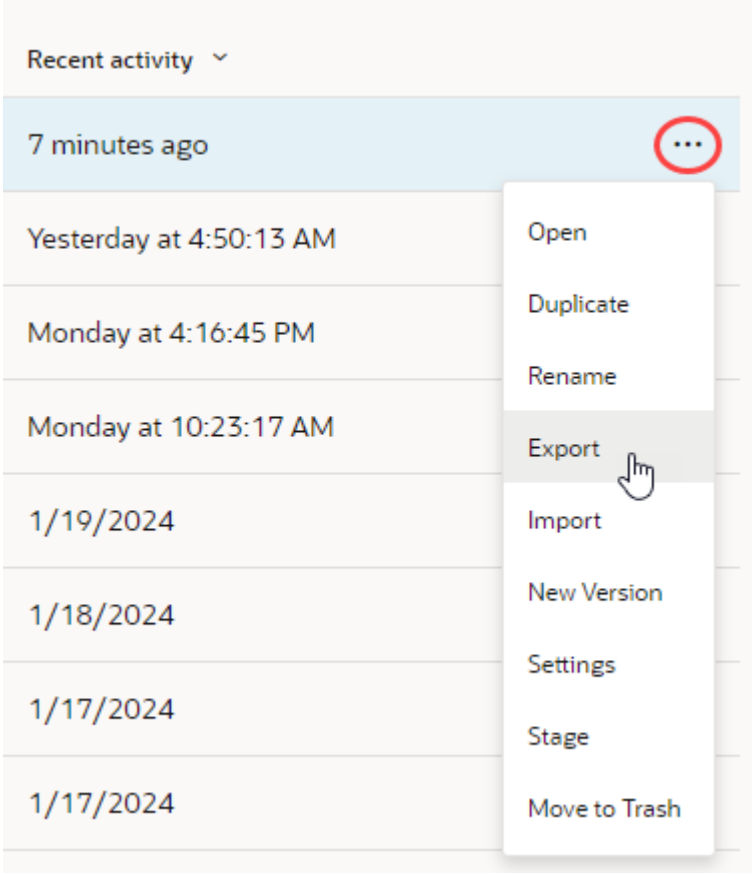
When you export the application, some information, such as credentials for external REST endpoints. This information needs to be provided after the archive is imported.

> **Note:**
>
> The instance from which you export the archive must be on the same or earlier version of Visual Builder as the instance to which you plan to import it. If it isn't, you won't be able to import successfully.

To export a visual application:

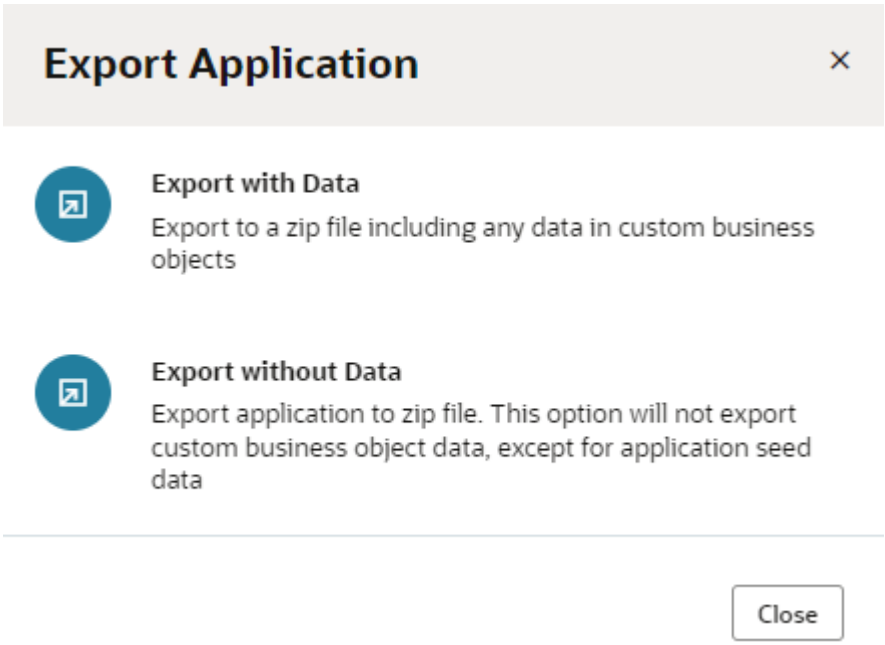
1. On the Visual Applications Home page, open the Application Options menu for the application version you want to export and select **Export**.

   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 in the Designer, you can choose **Export** in the application's Menu in the toolbar.

2. If the application contains business objects, you can choose to include the data stored in the objects when exporting the application. To do this, click **Export with 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.
If you exported the application with data, the archive will include a JSON file (`entity.json`) and a spreadsheet (`entity-data.csv`) for each business object. The JSON file describes the business object and the spreadsheet contains the business object data. If you chose 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.

If user roles are defined for the application, the role-mapping definition (which maps user roles to IDCS groups) will be copied to a JSON file (`role-mapping.json`) and included in the exported application archive.

Import a Visual Application

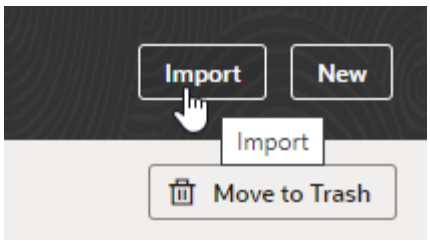
You can use the Import action to import an archive of a visual application, for example, to create a new application from the archive.

To import a visual application, you need an archive: Either a teammate must share the archive with you, or you can export one yourself, as long as you have access to the application.

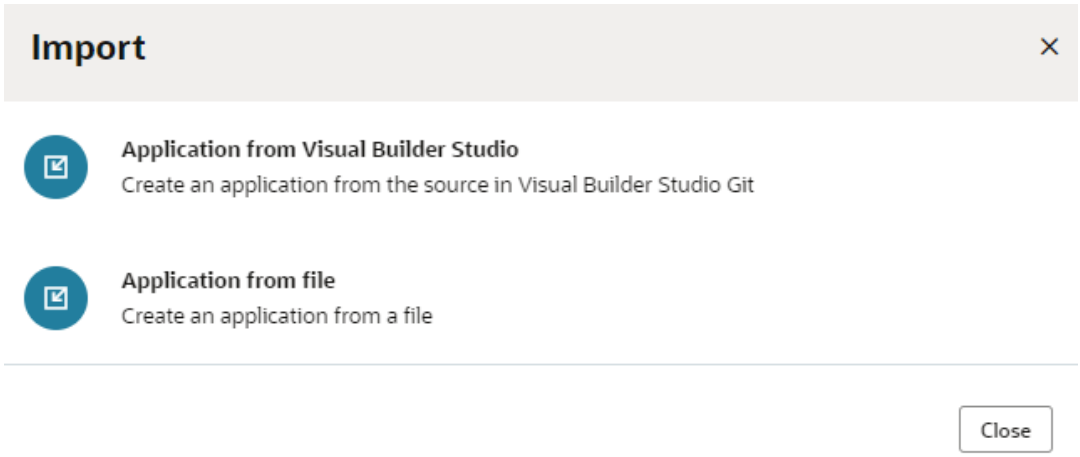
> **Note:**
>
> To successfully import a visual application, the instance from which you export the archive must be on the same or earlier version of Visual Builder than the instance to which you import it. You'll get an error if the source instance is running a later version of Visual Builder than the target instance.

To import a visual application:

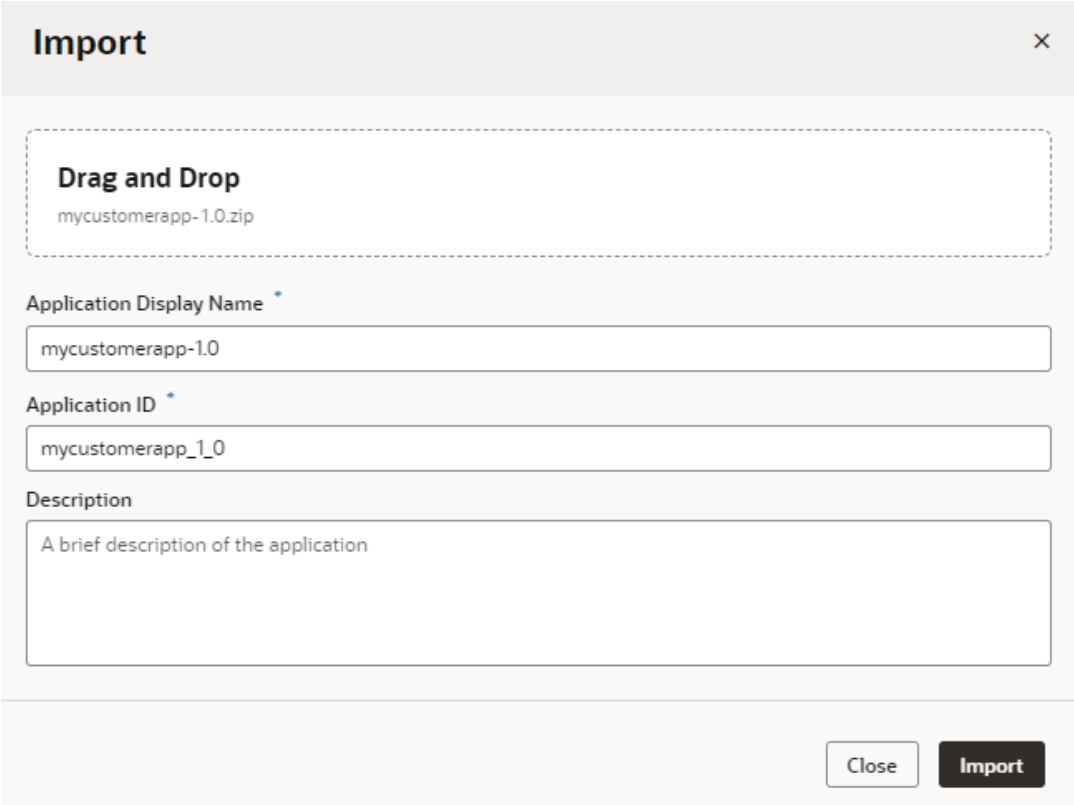
1. Navigate to your Visual Applications Home page and click **Import**.



2. Click **Application from file** in the Import dialog box.



3. Drag the 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 the Application Display Name and Application ID. Both fields are automatically populated based on the archive name, but you may want to modify the name as desired and the ID to be unique in your identity domain.



5. Click **Import**.

After the application is imported, you may need to provide additional details for the newly created app. For example, if the application you're importing contains backend/service connection definitions that use credentials, you will need to specify the credentials in those backends/service connections.

Any user roles that were defined for the exported application are re-created for the new app in the Settings editor on the User Roles tab (based on the `role-mapping.json` file in the exported application archive). If this doesn't happen (say, because you're importing an older app whose users and groups no longer exist in IDCS), you'll need to set up the user roles again in the new application. See Manage User Roles and Access.

About Classic Applications

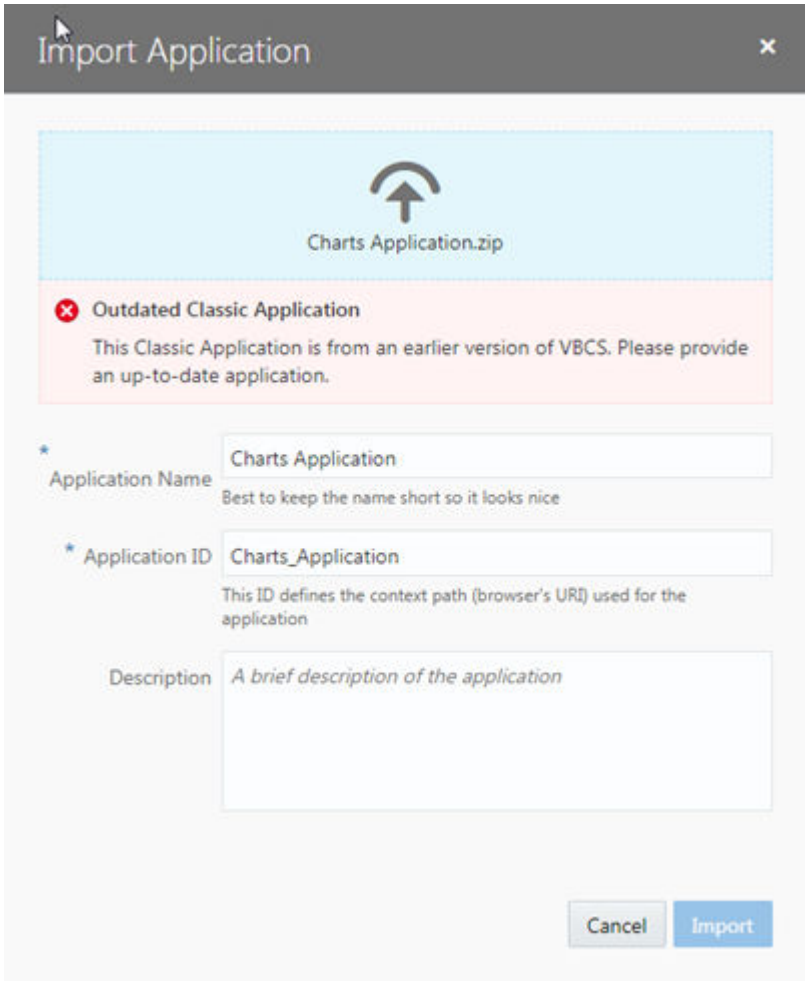
Classic applications were created in earlier versions of Oracle Visual Builder and used a structure that is not compatible with the visual application structure now used in Oracle Visual Builder. There will be no additional feature development work to support classic applications. If you have any classic 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.

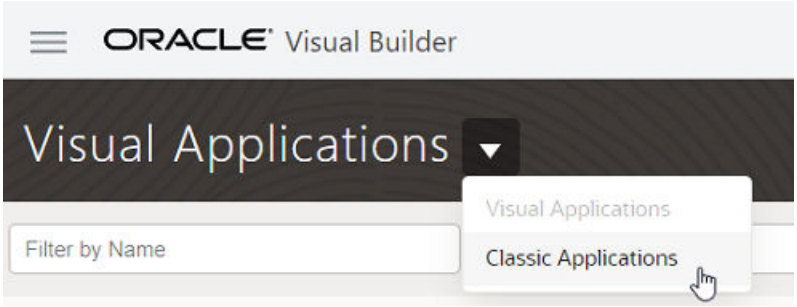


> **Note:**
>
> If you see an error message similar to `Failed to import application myClassicApp. The file is an invalid application export file`, you'll need to change the extension of the file you want to import from `.zip` to `.ovb`.

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**.

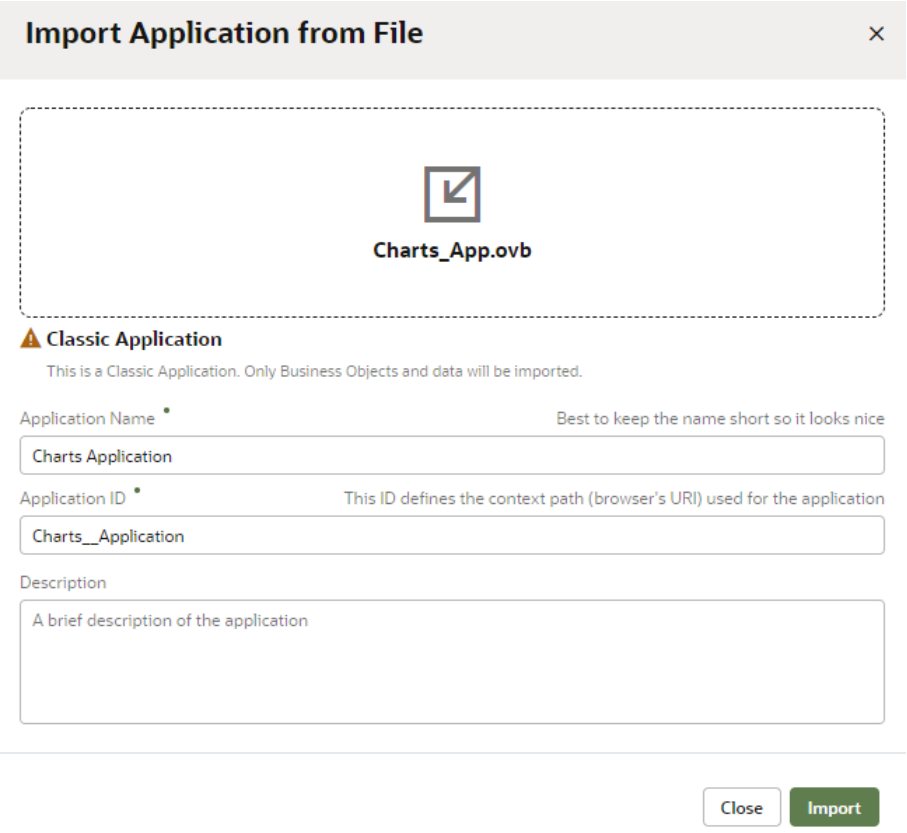
   The application is exported as a `.zip` file to your local system.

7. Select the `.zip` file imported to your local system and change its file extension to `.ovb`.

8. 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 (see Import 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:



Add Web (and Mobile) Apps to Your Visual Application

A visual application is a container for all your web (and mobile) applications. You can have any numbers of apps in your visual application, even both web and mobile apps in the same visual application.

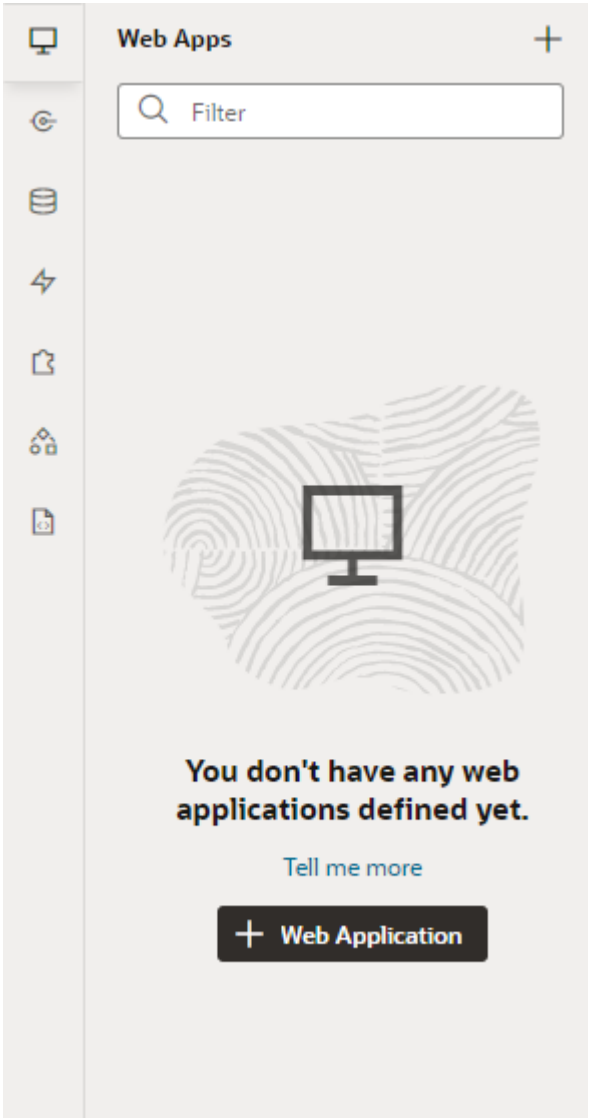
Create a New Web Application

You can create multiple web apps within your visual application. Each web app is independent, but they can all use the data sources defined in the visual application.

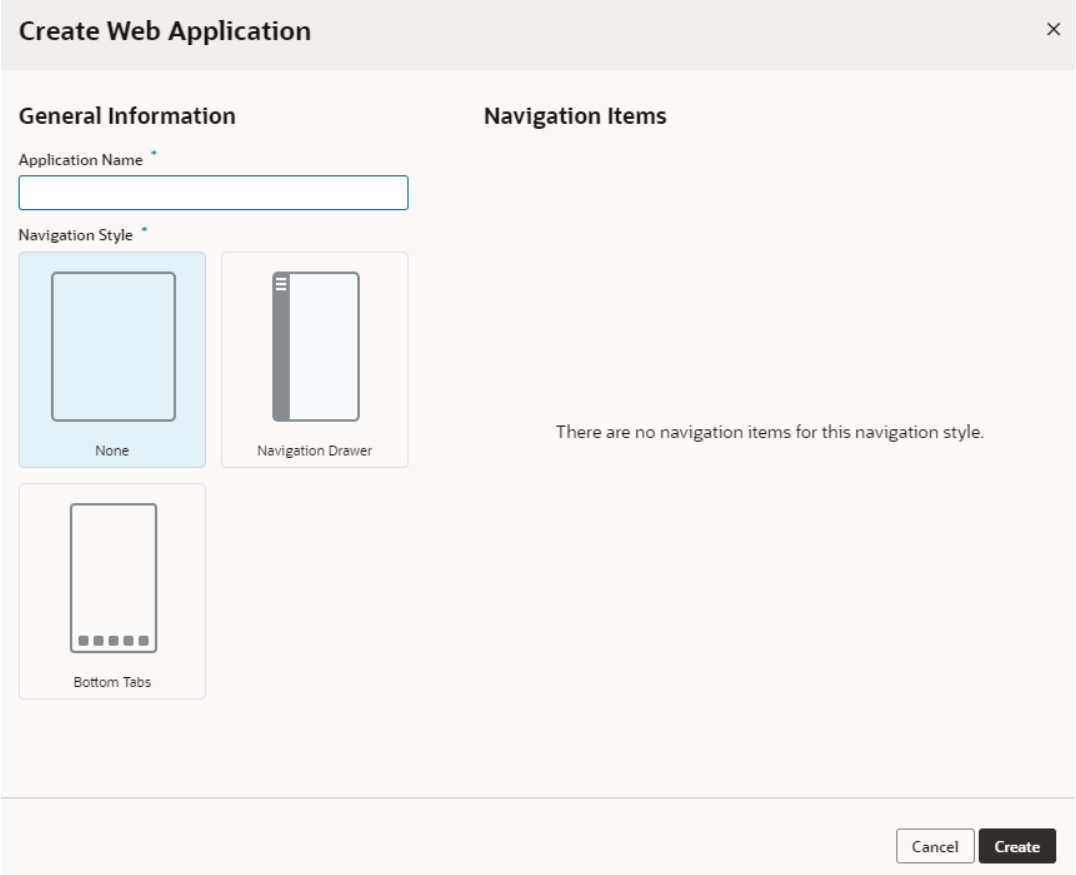
To create a new web application:

1. Click **Web Applications** in the Navigator to open the Web Apps pane.

   Structural representations of each web application in your visual application show in the Web Apps pane. If no web applications have been created, you'll see a message and a + Web Application button.



2. Click **+ Web Application**, or the Create Web Application icon (+), in the Web Apps pane.
3. In the Create Web Application dialog box, enter a name in the **Application Name** field under General Information. The name you enter will be used as the app's display name in the Web Apps pane. You can specify uppercase as well as lowercase characters, but the name will be converted to lowercase.

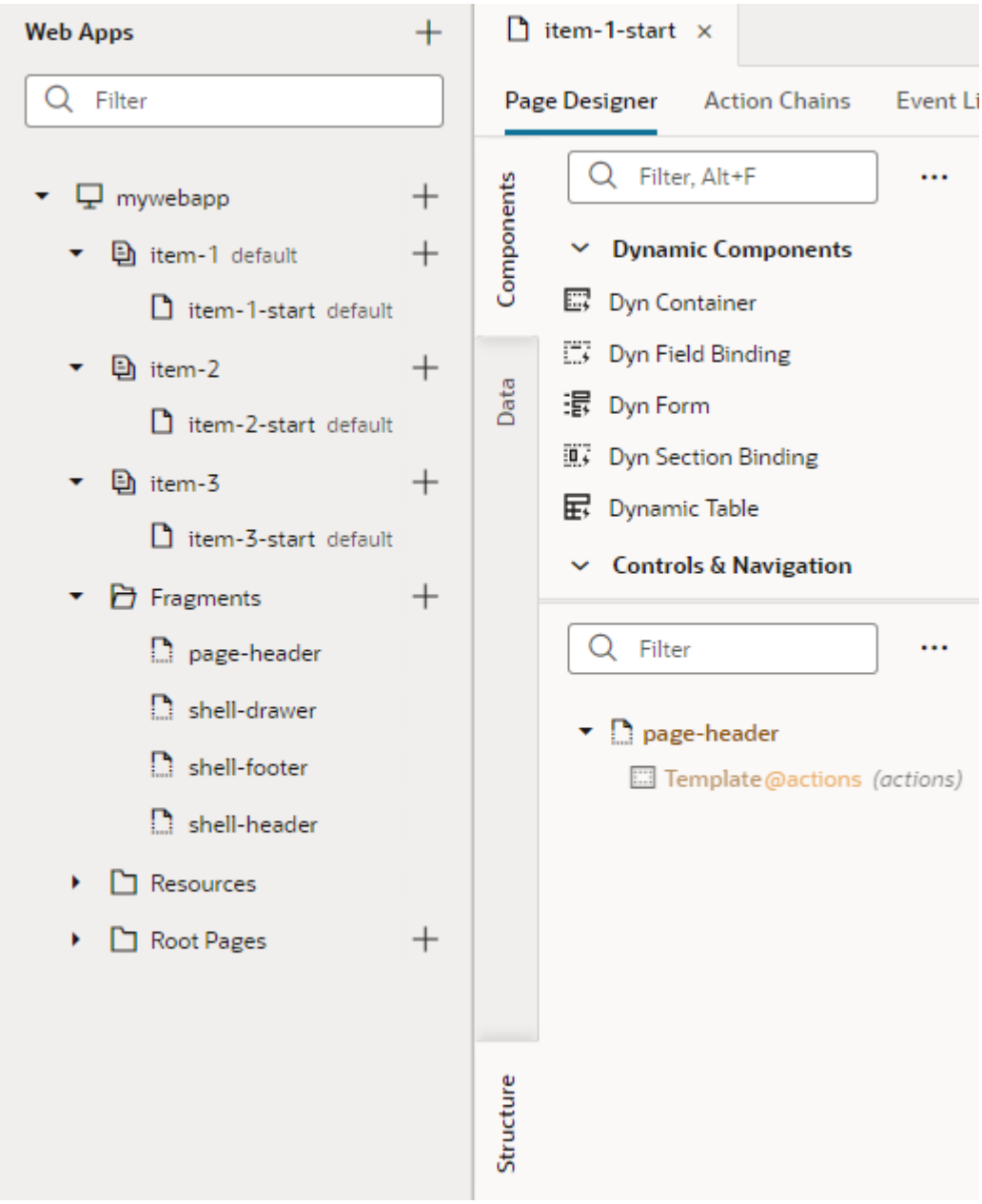


4. Select a navigational style for the app based on the available templates:
   - Select **None** to create a web app without any navigational components, if you want to design the app's navigation on your own later.
   - Select **Navigation Drawer** to create a web app with a Navigation Drawer, which shows the app's main navigation menu in a separate panel (users would get to this navigation menu by clicking ••• in the application header).
   - Select **Bottom Tabs** to create a web app with a tab bar at the bottom that enables navigation between items.
5. If you chose Navigation Drawer or Bottom Tabs, specify a name for each item under Navigation Items. Click **Add Item** to add as many as you need or to remove those you don't need.

   Each navigation item serves to group related pages under a separate *flow*. A flow can have one or more pages and is typically used to group pages by business function. For example, if you were creating an expenses app, you might have two navigation items: one called `My Expense Reports` to group the pages that summarize expenses and let users create and edit expense reports, and another called `Administration` to group the pages where managers approve or reject expense reports.
6. Click **Create**.

Your new web app shows up in the Web Apps pane and opens the *`flowname`*`-start` page, created by default as your application's home page. This page is the default page in the default flow (badged as `default` next to the page *and* flow) and is what users will first see when the app is run.

Expand the application node in the Web Apps pane to see the app's structure, with nodes and subnodes representing its artifacts and files. For example, here's a tree view of a web app that uses the **Navigation Drawer** template (with the default navigation items):



Because a flow is generated for each navigation item, you see multiple flows (`item-1`, `item-2`, and so on), each with its own starter page (`item-1-start`, `item-2-start`, etc). If you chose **None** as your template, you would see a single page flow (`main`).

Irrespective of the template used, all web apps include reusable fragments that define common components that appear throughout your application. Fragments prefixed by `shell-` provide a common set of interactions throughout your application and helps users navigate and interact with your UI:

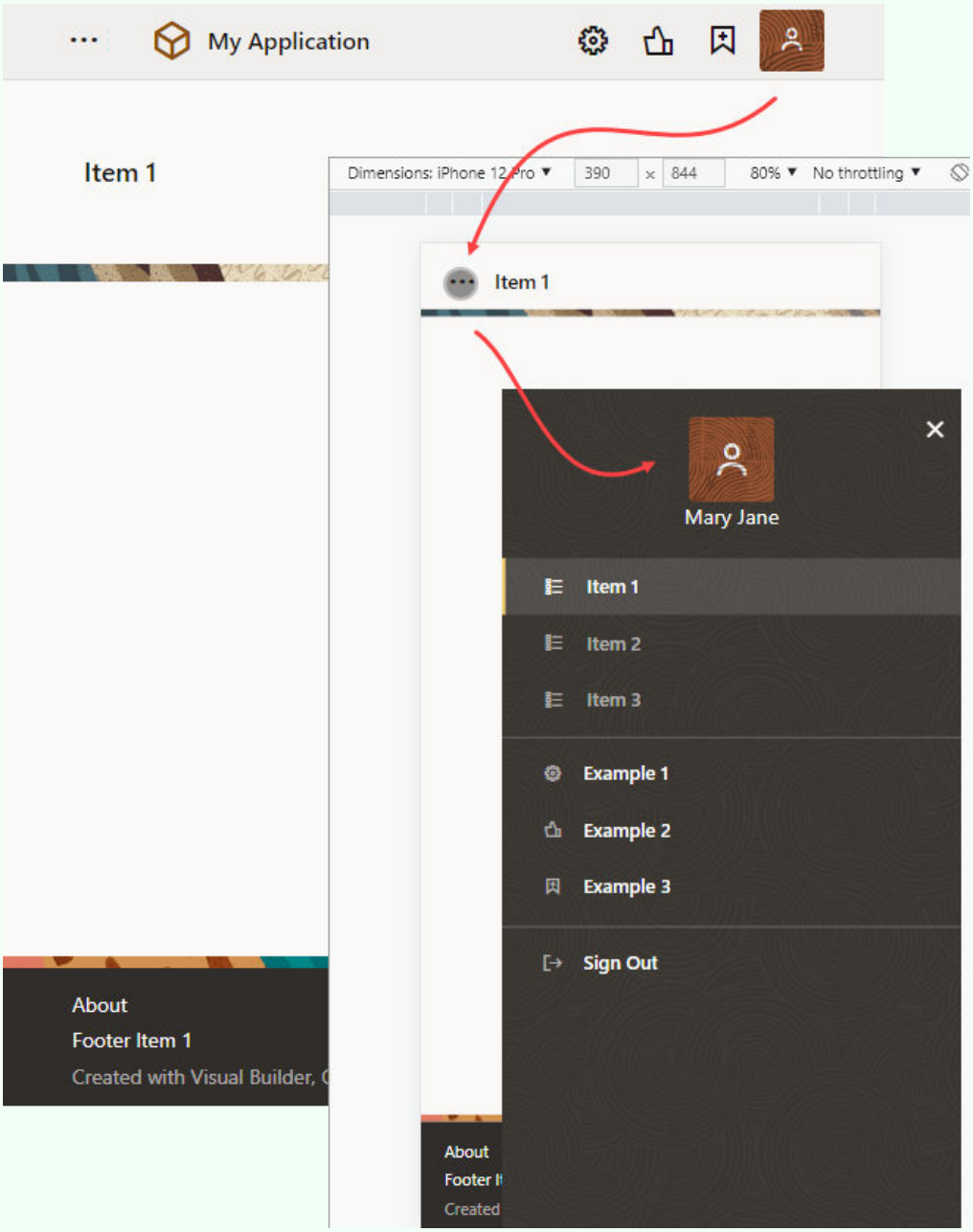
- The `shell-header` defines the global header which appears as the topmost element in the browser. It includes your company logo, application name, and user profile, besides other actions and utilities.
- The `shell-footer` defines informational components, such as contact details and copyright information, and appears at the bottom of the browser.
- The `shell-drawer` is specific to the Navigation Drawer template and defines the items that form the app's navigation menu in a separate panel.

In addition to shell fragments, a `page-header` fragment defines page-specific content that appears above a page's main content. You would not work with the `page-header` fragment directly, but you can customize this fragment on your page to create a custom page-level header.

With your app now created, you are ready to design and develop its pages. You might also want to familiarize yourself with the Designer.

Tip:

Web app templates lend themselves to responsive layouts and can adjust to the size of the user's screen, ranging from small phones to wide-screen desktops. Here's an example of a web app that uses the Navigation Drawer template:



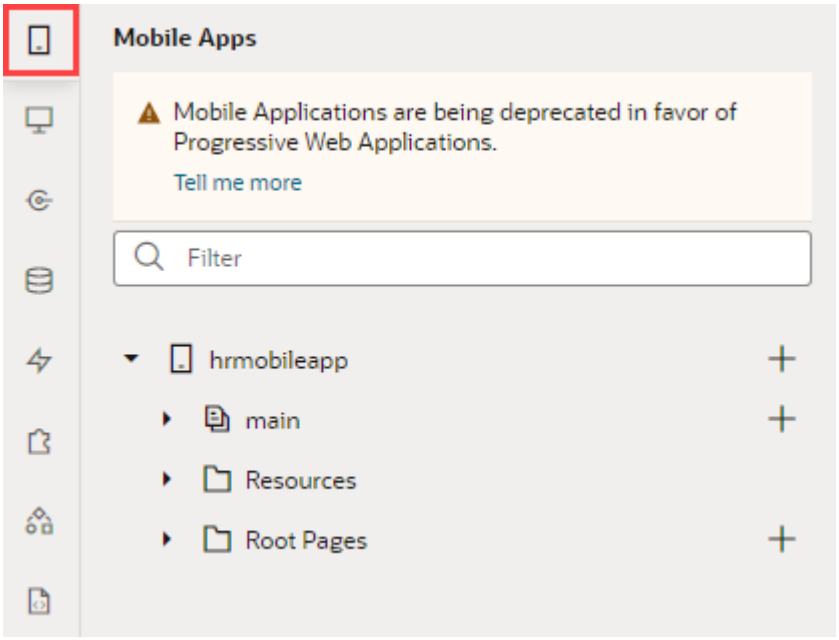
When the form factor is changed to a mobile device, notice how the header items move into the drawer panel when the user clicks `•••`. Additionally, the application title (`My Application` in the example) is replaced by the page title (`Item 1`). The global header (which contains the company logo and application title) is displayed only in desktop mode. To customize your app's navigation items as well as elements in the header and footer, see Edit an App's Header, Footer, and Navigation Items.

Import an Existing Mobile Application

Starting with version 23.10, mobile apps have been replaced by Progressive Web Apps (PWAs). You can no longer create a mobile app, but you can import an existing mobile app and work on it—though you must deploy it as a PWA to be able to use it.

To import an existing mobile app, you must import the visual app containing the mobile app you want to work on. See Import a Visual Application.

When you import mobile app, your mobile app will show up in the Mobile Apps pane (which won't appear otherwise):

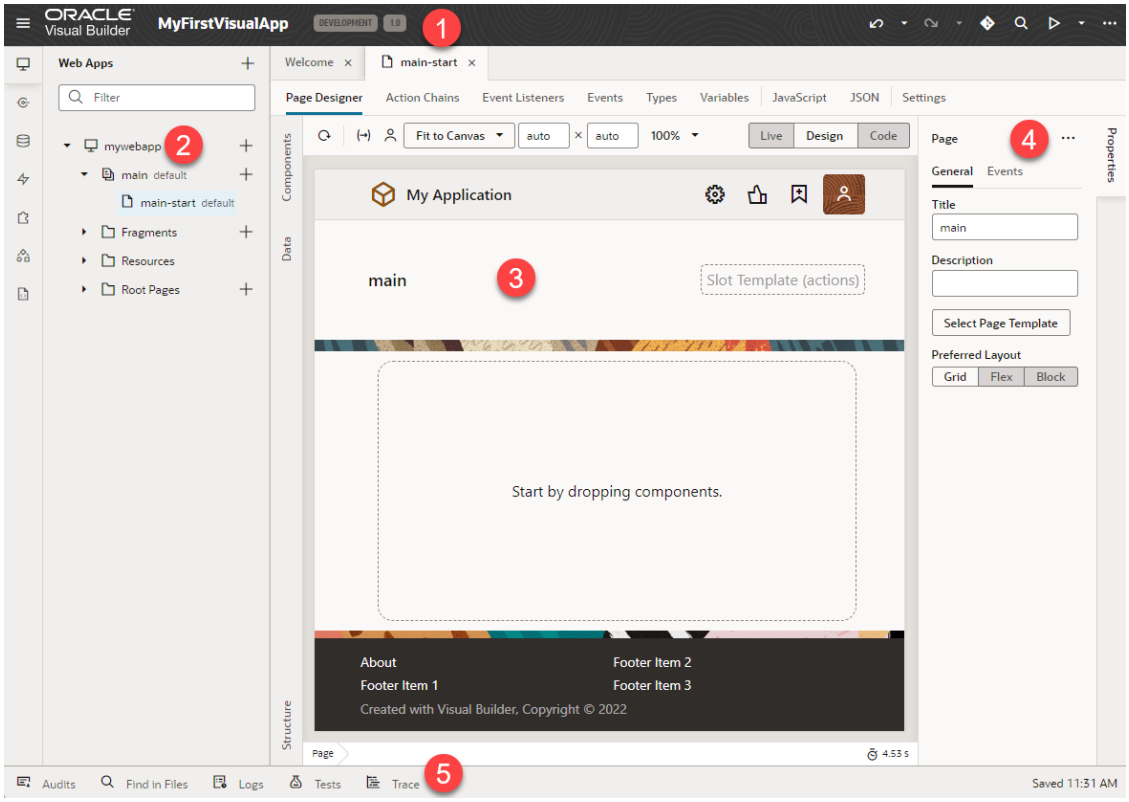


Once you import your mobile app, you can work on it as before, but you must enable PWA support to allow users to install the app on mobile devices. See Run Mobile Applications as PWAs.

Tour the Designer

A visual application is your ticket to the Designer, a declarative environment that you use to design and develop apps within your visual application.

The Designer has five distinct areas (as highlighted in this image):



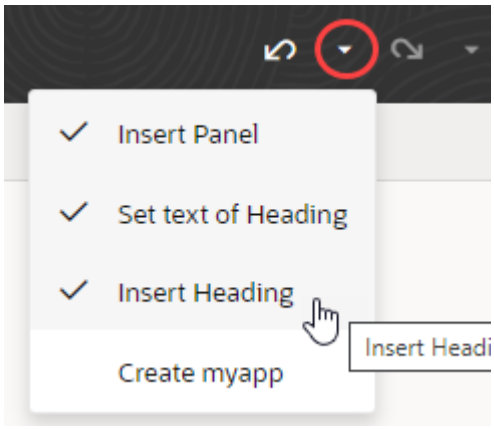
Label	Element
1	Header
2	Navigator
3	Canvas/Editors
4	Properties pane
5	Footer

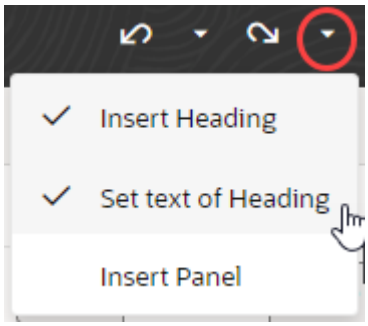
Let's take a look at each of these areas to learn more about what each one does.

The Header

The header contains information about your current Visual Builder session, as well as access to the tools you need to develop your application.

Label/Icon	Element	Description
	Toggle main menu	Open a side panel, where you can select **All Applications** to go to the Home page. If you're an administrator, you'll see additional options to manage **Settings** and **Certificates**. You can also click to navigate to the Oracle Integration Home page.
MyFirstVisualApp	Visual app name	Name of the visual application you're working with.

Label/Icon	Element	Description
Development, **Stage**, or **Live**	Status indicator	Status of the visual application.
1.0	Version	Version of the visual application.
	Undo	Undo one or more of your changes. To undo your most recent change, click **Undo** (hover your cursor over the icon to view the action that will be undone). To undo multiple changes, click the Undo drop-down list and select the actions you want to undo. For example, selecting the Insert Heading action in this image will remove the heading and undo other changes you made after adding the heading:  **Tip:** You can undo up to 10 of your changes at a time (your last 500 actions are stored in the browser and will be lost if you clear the browsing cache). To undo more than 10 actions, simply undo a few items, then open the drop-down list again.

Label/Icon	Element	Description
	Redo	Redo one or more changes after undoing them. To redo your most recent change, click **Redo** (hover your cursor over the icon to view the action that will redone). To redo multiple changes, click the Redo drop-down list and select the actions you want to redo. For example, selecting the Set text of Heading action in this image will revert two of the previously undone actions: 
	Git	Integrate your visual application with a Git repository.
	Go to File	Search your application's sources by file name. When you first click **Go to File**, your most recently used files show. If the file you're looking for isn't listed, start typing the name of the name in the search box, then select the file you want in the suggestions.
	Preview	Use the **Preview** icon to see how your pages look and behave in a browser. Typically, you'd use Preview mode, but you can also use the Debug Preview mode to debug issues with your application.
	Menu	Open a menu containing the Share, Import, and Export actions, as well as options to open the Settings editor. You can also navigate to the Visual Builder Help Center and discussion forum.

The Navigator

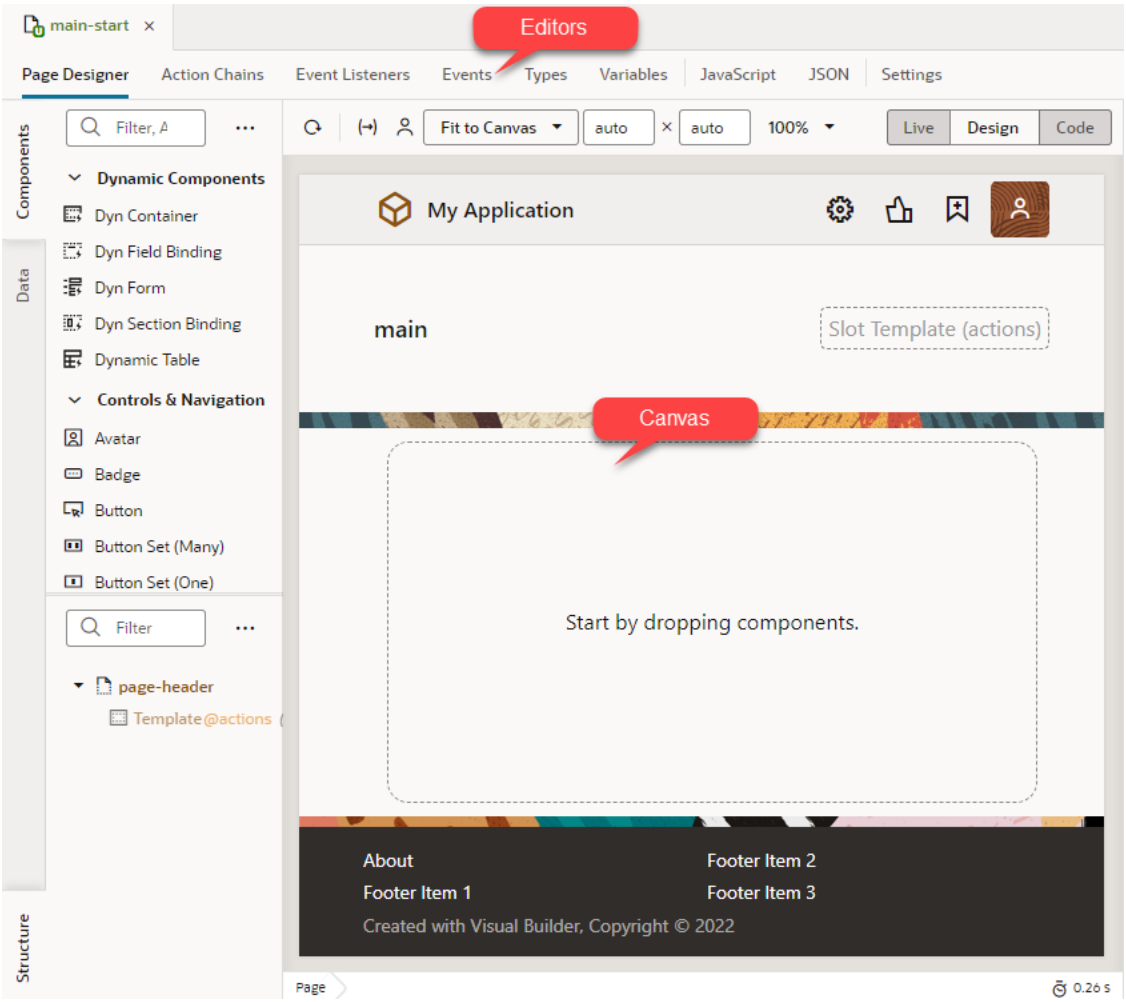
The Navigator helps you move between the artifacts in your extension, and provides access to the Visual Builder editors:

Icon	Element	Description
	Web Apps	This is where you: • Create a web app; • Create a fragment, if you need a reusable piece of UI you can use in your app; • Import resources to use in your app.
	Services	To interact with external REST APIs in your visual app, you can create connections to the services that provide access to these API endpoints. You also have access to a catalog of predefined services through *backends* such as Oracle Cloud Applications and Oracle Integration. The Oracle Cloud Applications backend, for example, exposes REST APIs—from Human Capital Management, Sales, and more—that your visual app can consume right out of the box. You can also connect to services that aren't listed in this catalog. See Manage Service Connections.

Icon	Element	Description
	Business Objects	Business objects are custom data objects that implement your app's business logic. Your visual app can access and interact with the business objects you create through REST endpoints that Visual Builder generates for you. By default, this data is stored in an embedded database associated with your environment's Visual Builder instance. See Work With Business Objects.
	Layouts	A Layout represents a set of data fields that can appear in one or more related dynamic components, like a table or form. Create a new Layout here by choosing a data source, then defining the rule set that governs how that data looks and behaves. See How to Create Layouts With Dynamic Components.
	Components	The Components tab helps you to install and manage custom web components that you download from the Component Exchange, a repository of components that can be installed in your Visual Builder instance. See Work with the Component Exchange.
	Processes	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. After registering your business processes in your visual application, you can add UI components to your pages that start processes and that can be used to complete user tasks. See Work With Business Processes. **Note:** Oracle Process Cloud Service, which is included in the Enterprise edition of Oracle Integration Gen 2, is deprecated. See Process Features for details.
	Source	Although the Designer is primarily a visual editor, you can always work with source code if you prefer. See Work With Code.

The Canvas/Editors

The canvas, which appears to the right of the Navigator when you open a page, is where you do the bulk of your work in Visual Builder. When you open a page, you'll see different editors along the top to help you modify and create artifacts used in the page, like Page Designer, Actions, Event Listeners, and so on:



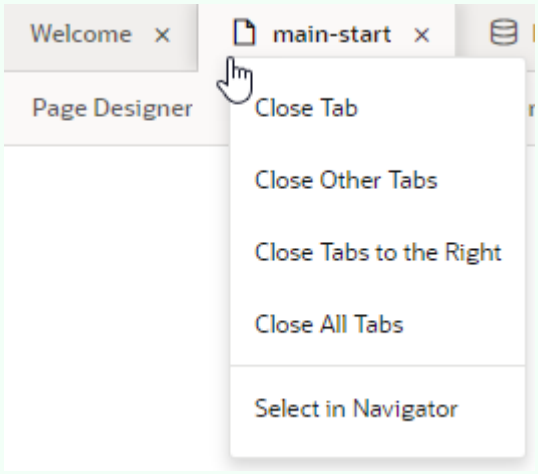
Depending on which aspect of the page you're customizing or building, Visual Builder invokes the proper editor to provide the experience you need, and displays that editor in the canvas. Perhaps the most important editor is the Page Designer, described in detail in Use the Page Designer.

All of the changes you make through the editors—the Page Designer, Actions, Event Listeners, etc.—are saved as JSON, which you can access through the **JSON** pane, next to Settings. In addition, you can use the **JavaScript** pane to enter any custom functions you may need. For details about the different editors, see Which Editor Do I Use?.

Tip:

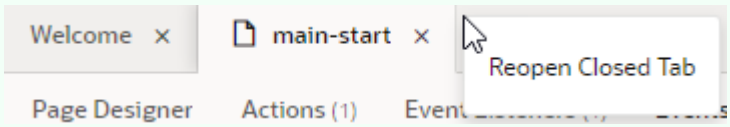
When working with multiple artifacts (pages, flows, business objects, and so on), each artifact opens as a separate tab on the tab bar. Here's how you can better manage these in your work area:

- Right-click a tab to see options to close the particular tab, close other tabs, close tabs to the right, or close all tabs. You can also use this menu to select a particular tab in the Navigator:



  If the tabs overflow available space on the tab bar, click at the edge of the tab bar and select the tab you want to open. Note that the active tab always stays in focus.

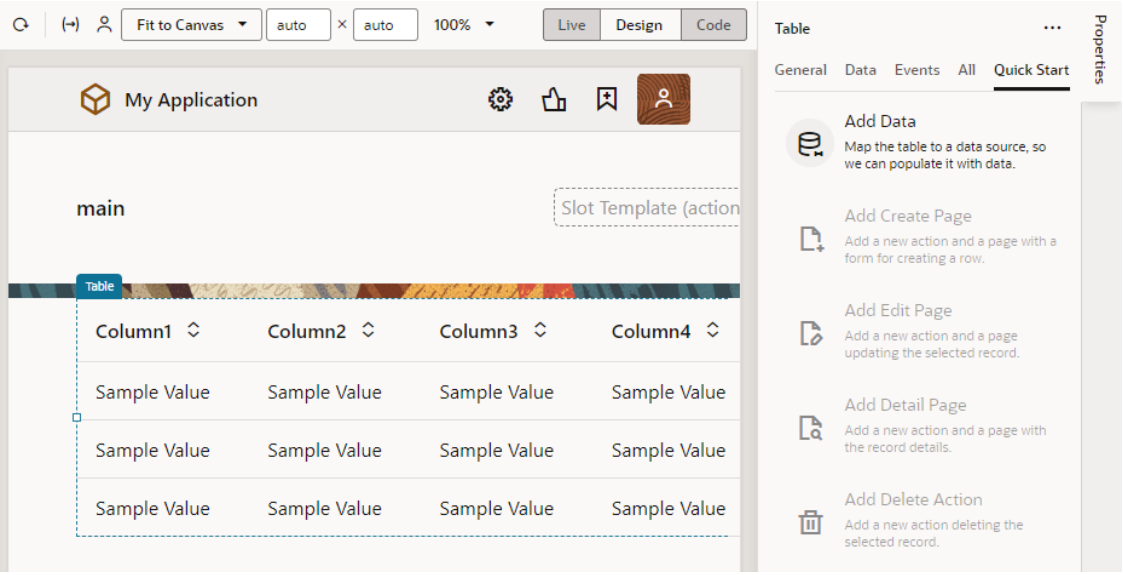
- Right-click the empty space on the tab bar and select the option to reopen closed tabs. Recently closed tabs are saved in session history, so you can keep reopening tabs until you get to the one you want.



The Properties Pane

As the name suggests, the Properties pane lets you specify the properties that control the appearance and behavior of whatever is currently selected in the canvas. While you're in the Variables editor, for example, you use the Properties pane to set the variable's default value, type, and other attributes. When you click a component on a page in the Page Designer, the Properties pane is immediately updated to reflect the component's properties. Depending on the component, the Properties pane might display additional tabs for modifying the component's attributes or its behavior.

When you add a collection component to the canvas, like a table or list, you'll see a Quick Start tab added to the Properties pane:



Quick Start wizards 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.
To hide or show the Properties pane, just click the tab itself.

The Footer

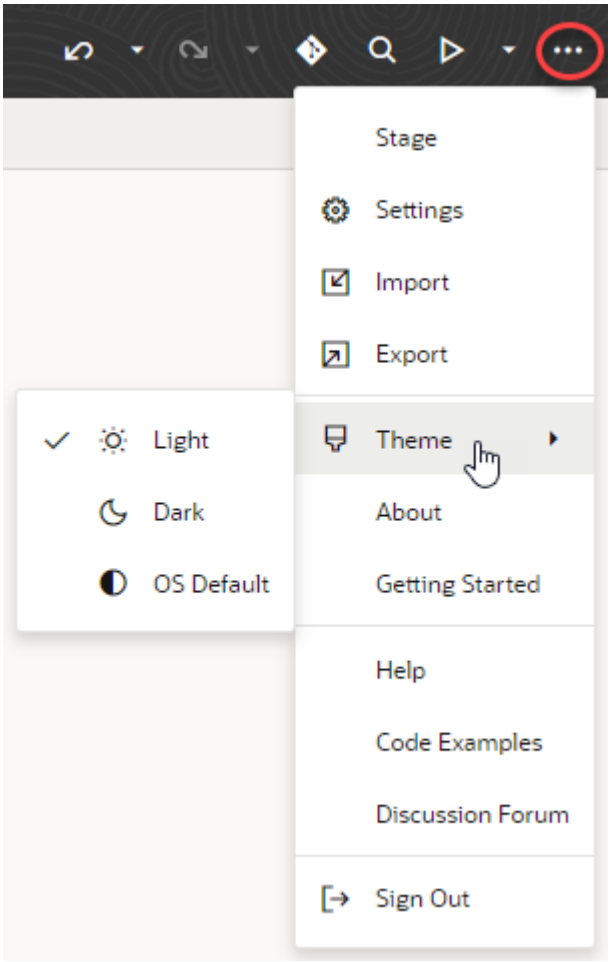
At the bottom of the Designer, you have several tools that can help you debug your visual application:

Element	Description
Audits	Scan your visual application's code for errors and warnings. Your code is scanned when you open the Audits pane. See Audit Application Code.
Find in Files	Search your application's Git repository for a text string.
Logs	View build logs when a visual application is staged or published. See Troubleshoot Build Issues. Also, enable logging to assist with debugging when developing rules for a business object. See Enable Logging for Scripting Events.
Tests	Run all test cases defined for action chains in a visual app and run them to make sure code changes haven't broken any functionality. See Use the Tests Footer in a Visual Application.
Trace	Enable tracing of a business object's endpoint requests to diagnose performance bottlenecks. See Enable Tracing to Monitor Endpoint Calls.

Color Theme

Visual Builder, by default, uses a light theme based on Redwood to set the color palette for your work environment. You can personalize this theme to switch to a dark theme or sync with your OS settings.

1. Click **Menu** in the upper right corner of the Designer.
2. Select **Theme**, then choose an option:



- Select **Dark** to use a dark color display, more suited for low-light conditions. This option switches the background and text used in all the editors, except the Page Designer canvas, where application pages continue to display against a lighter background with dark text.
- Select **OS Default** to inherit the theme used in your operating system's settings. If your system settings are configured to use dark mode, the Designer will also use those settings.
- If you changed the default, select **Light** to switch back to a lighter background with dark text display.

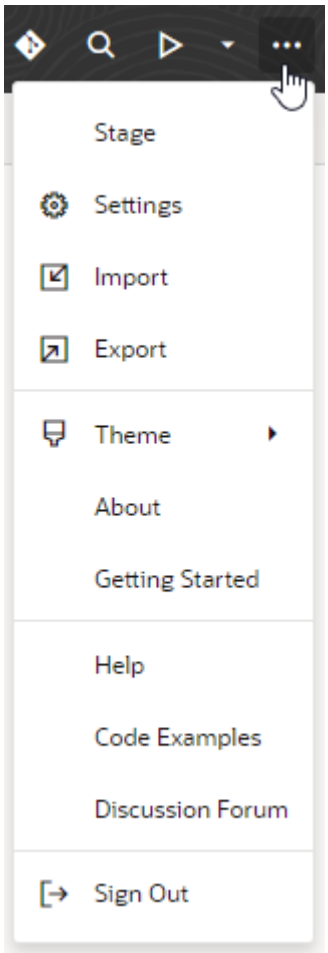
If you want to change the way your app appears to users, you can customize the app's theme to render its components against a dark background, even let your users choose between a Light or Dark theme. See Customize a Web App's Appearance.

Common Tasks for Visual Applications

Because a visual application is a container for your web (and mobile) apps, you can manage things at the *visual application level*, meaning settings at this level will apply to all the web (and mobile) apps within the visual application.

Manage Visual Application Settings

You configure settings for a visual application in the Settings editor. To access the Settings editor, locate the application whose settings you want to change on the Visual Builder Home page. Click **Menu** in the upper-right corner and select **Settings**:



The Settings editor includes several tabs that group related settings. Here's how you use the different settings tabs for a visual application:

Tab	Description
Application	Manage general and runtime dependency settings:
Application Display Name	(Read-only) Application's display name, which you can see in the Designer's header as well as all pages used to manage the application.
Application Name	(Read-only) Application's internal name, which becomes part of the browser URL used to access the application as well as in the Developer's console.
Application ID	(Read-only) Application's unique ID based on the application name and version, both of which become part of the browser URL used to access the application.
Vanity URL	Custom domain that you can use to shield customers from the details of your server's host and domain name. See Specify a Custom App URL.
Description	Optional description of the application.

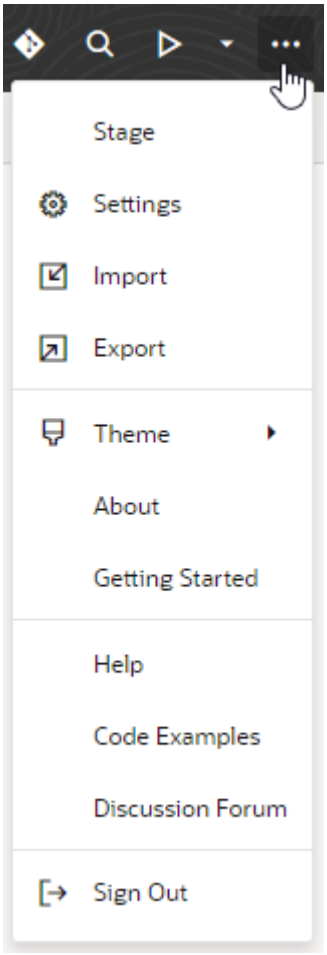
Tab	Description
Runtime Dependency	Client-side libraries that, along with the accompanying version of Oracle JET, determine features and enhancements available to your visual application. See Manage Runtime Dependencies for Visual Applications.
Troubleshooting	Option to clear the resource cache. See How Do I Clear My App's Resource Cache?
Translations	Download the strings that appear in the user interface of your visual application's web (and mobile) apps to import into a third-party translation tool for translation. You then upload the translated strings from the translation tool to use for those apps that support different languages. See Work With Translations.
Application Profiles	Deploy your app with different settings depending on the environment. For example, you won't want to use a production REST service with access to live customer data when developing an app. Instead, you'll use a development or test instance of the service. Once you complete development and your app is deployed to production, you'll want it to connect to the production REST service. Application profiles help manage the switch between the different instances of the REST service. Application profiles can be associated with your application's backends and service connections, as well as user roles. They can also be mapped to environment-specific schema when you bring your own database schema for business objects. **Note:** Application profiles belonging to visual applications created on Visual Builder Studio and deployed to a Visual Builder instance may at times be marked as disabled on your instance. While you won't be able to readily remove disabled profiles, you can't duplicate, rename, or change their configuration either.
Team	Collaborate with others on the visual application. Only users who have been added to the app as team members can edit an app or perform lifecycle operations. See Add Team Members.
User Roles	Control access to business objects and data in your apps based on a person's user role. See Authentication Roles Versus User Roles.
Business Objects	Retrieve the API for the catalog of endpoints exposed by business objects in your visual application. Other settings in this tab configure client's access to this API. You can configure anonymous access, basic authentication, or get an access token that a client can use. See Allow External Access to Your Business Objects and Get an Access Token for Authentication.

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. As an admin, you can see all applications, but you cannot act on the app unless you are a team member. Your admin access, however, will let you add yourself to the app when you want to edit it or perform other lifecycle operations.
To add a team member to an application:

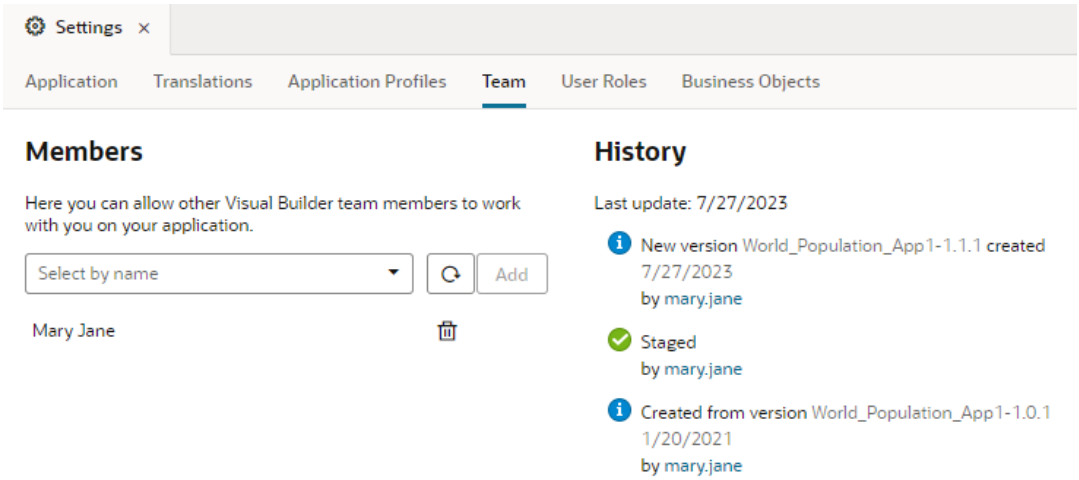
1. Open your web (or mobile) and choose **Settings** in the application's Menu in the toolbar.



Alternatively, on the Oracle Visual Builder Home page, locate the application where you want to change the settings and choose Settings in the Application 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.



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

   The drop-down list displays the email addresses of all the members in your identity domain who can be added to the application as developers.

Export and Import Application Resources

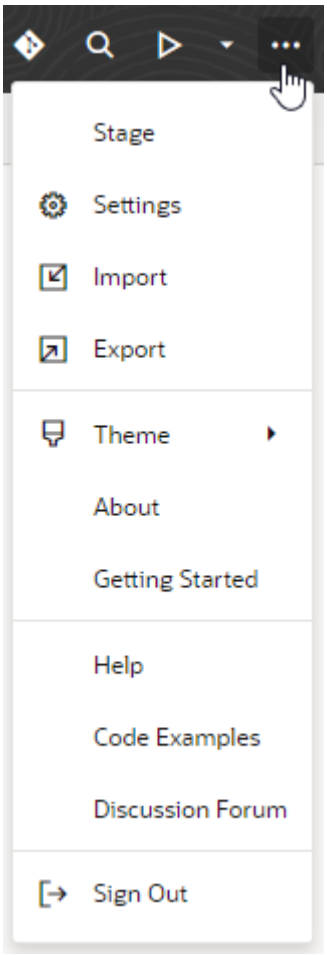
You can import and export a visual application's resources to share source files and to move applications between instances.

Export Application 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. Exporting an application downloads its resources as a ZIP archive to your local file system.

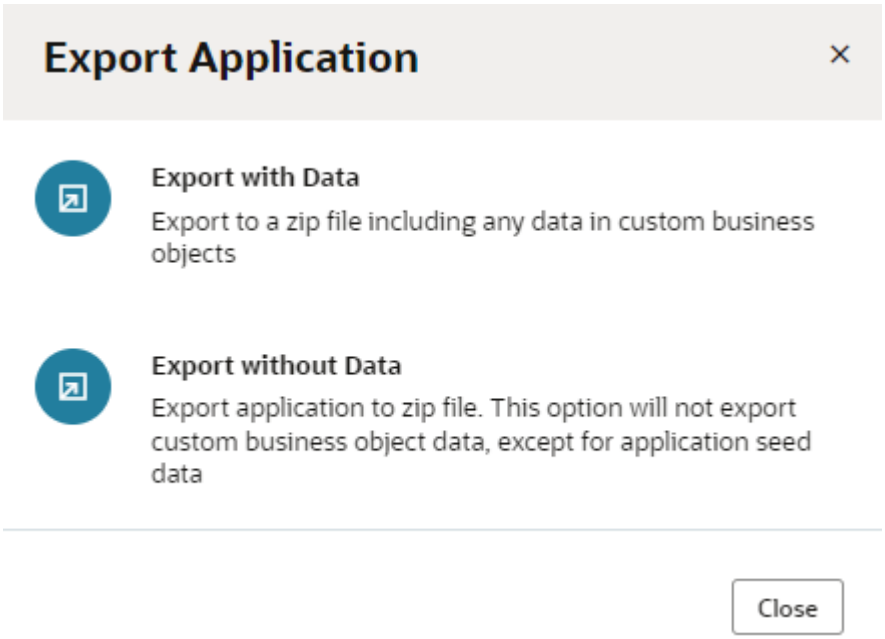
To export an application's resources:

1. Click **Menu** in the upper right corner and select **Export**.



Alternatively, on the Visual Builder Home page, locate the application and select **Export** in the Application Options menu.

2. If your application contains business objects, you can choose to include the data stored in the objects when exporting the application:



An archive that includes the application's resources is downloaded to your local file system, in the location specified for your browser's downloads.
If you exported the application with data, the archive will include a JSON file (`entity.json`) and a spreadsheet (`entity-data.csv`) for each business object. The JSON file describes the business object and the spreadsheet contains the business object data. If you chose 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.

If user roles are defined for the application, the role-mapping definition (which maps user roles to IDCS groups) will be copied to a JSON file (`role-mapping.json`) and included in the exported application archive.

Import Application Resources

You can import resources to replace your visual application's source files with those from another visual application.

To import resources from one application to another:

1. Click **Menu** in the upper right corner and select **Import**.
2. In the Import Resources dialog box, drag and drop the ZIP archive of a visual application that you've previously exported, or click the drop area to locate the archive on your local system.
3. To replace all existing files (and prevent duplication), select **Delete existing files and resources**.
4. Click **Import**.

The resources are imported to your visual application's root directory.

Upgrade Imported Resources

Sometimes when resources are imported, you might see a message that the app's resources are using an older set of runtime dependencies and need to be migrated.

To migrate these resources:

1. Click **Upgrade** in the banner.
2. In the Upgrade Imported Files dialog, click **Upgrade** to make your app's resources compatible with its current runtime version.

3

Anatomy of Visual Applications

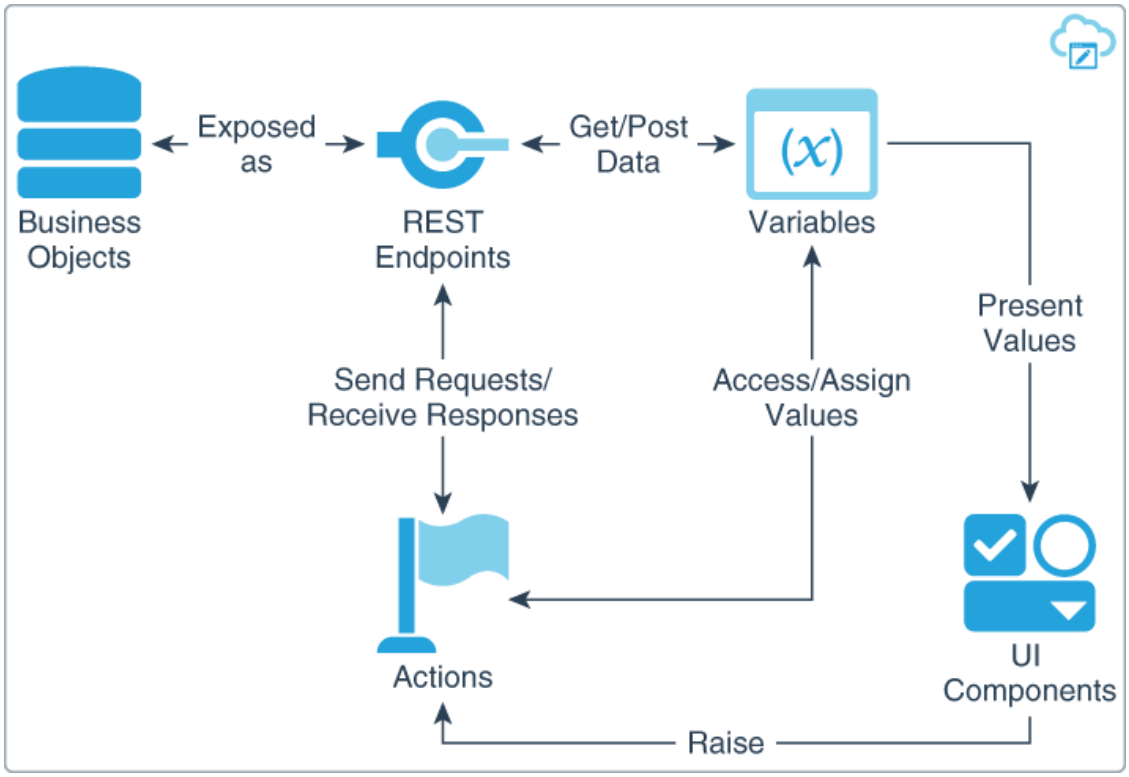
To develop applications with Oracle Visual Builder, you need to understand a few basic concepts.

The basic components of a visual application are web apps, services, business objects, and processes. The basic building blocks of an app 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) component contract. Component attributes are bound to variables, and component events and variable changes trigger action chains.
- All data access is based on REST. This data can come from business objects and service connections. Actions and variables control how data is sent to and from a REST endpoint in an app. 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.

Variables store intermediate state on the client between the 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 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.

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

Constants are a type of variable of the `Constants` namespace and are used to store values that do not need to change over time such as company name or measurement conversion values. Constants are typed like other variables and are supported in the same scopes. But

they differ from other variables in that you can't change their values after they are initialized. For more information, see Constants in the *Oracle Visual Builder Page Model Reference*.

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).

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, see Variables in the *Oracle Visual Builder Page Model Reference*.

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.

Property	Required?	Description
`type`	Yes	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 Extended Types).
`input`	No; applicable only if the property is within the page scope	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).
`required`	No	Whether or not the variable must be set
`defaultValue`	No	The default value for the variable to be initialized. If no default value is provided, the value is "not set" or undefined. The `defaultValue` can be bound to an expression, or it can be a structure that uses expressions that reference other variables.

Property	Required?	Description
`persisted`	No	Use persisted variables to extend a variable's lifespan beyond its defined scope. For example, when you want to keep an authorization token for the duration of a session, set this property to make sure the token is available throughout the session, even if the page is refreshed. Can be set to: • `device`: Stores the variable in the browser's local storage and persists it on the device where the application is running, even if the browser is closed. If you want to store a variable across sessions, use this setting. • `session`: Stores the variable but only during the current browser session. • `history`: Stores the variable 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. • `none`: Variable is not stored (default).

Expressions

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

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

The 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 or constant.

You can use the following implicit objects in expressions (all are prefixed by a dollar sign (`$`):

Name	Where Available	Description
`$application`	Everywhere	Application object
`$page`	In the current page	Current page instance
`$flow`	In the current flow	Current flow instance
`$variables`	Every scope that has a `variables` property	A shortcut for `$`*`most_specific_scope`*`.variables` in the current scope. In a page, `$variables` is a shortcut for `$page.variables`.
`$chains`	Every scope that has a `chains` property	A shortcut for `$`*`most_specific_scope`*`.chains`
`$chain`	Actions executing in an action chain	Chain in which the action is executing
`$parameters`	In the `beforeEnter` event	Input parameters for the page. This object is needed because page variables do not exist yet in the `vbBeforeEnter` event.

Name	Where Available	Description
`$listeners`	In a flow or page	Event listeners of a flow or page
`$event`	Event listeners and variable `onValueChange` listeners	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 `onValueChange` listener on a variable,`$event` is a structure with the properties `name`, `oldValue`, `value`, and `diff` (itself a structure).

`$initParams`	Everywhere	Declarative initialization parameters which can be used in expressions. Initialization parameters—or `initParams`—are evaluated early and can be used in expressions that are evaluated before variables exist (for example, in service declarations or translation bundle paths). The `initParams` are defined within a `configuration` block in `app-flow.json`, for example:
		
		`"configuration": {`
		`  "initParams": {`
		`  "myServicePath": "some/path/",`
		`  "anotherPath": "http://somehost/foo"`
		`  }`
		`},`
		`"services": {`
		`  "myservice":`
		`"{{ $application.initParams.myServicePath +`
		`'myservice.json' }}"`
		`},`
		`"requirejs": {`
		`  "paths": {`
		`  "myPrefix":`
		`"{{ $initParams.anotherPath }}"`
		`  }`
		`}`

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.

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.

Action Chain Part	Description
ID	String identifier for the action chain
Input parameters	Zero or more variables that can be passed into the action chain and added to the action chain context
Variables	Zero or more variables that are internal to the action chain and usable internally by actions

For more information, see Action Chains in the *Oracle Visual Builder Page Model Reference*.

Built-in Actions

Visual Builder comes with a set of built-in (or predefined) actions for an action chain, used for example navigation or assigning variable values. An action has the following parts that the developer can define:

Action Part	Description
ID	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.
Configuration	Any properties of the action that the user can configure. For example, for the Navigate action, the page to navigate to and any parameters required for that navigation.
Outcomes and Results	An action may have multiple potential outcomes (such as success or failure, or a branch). It can also return results.
Exported State	An action may export state that is available to future actions within the context of the same action chain.

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 JavaScript Actionsin the *Oracle Visual Builder Page Model Reference*.

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 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.

Event	Applies To	Can Cancel Navigation	Returns	Description
`vbBeforeEnter`	Page	Yes	None or `{ cancelled : true }` to cancel navigation	Dispatched to a page before navigating to it. 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). After this state is exited, the previous page's state can be torn down. Page state is not available. The following variable scopes are available: • `$application`: All application variables • `$flow`: All parent flow variables • `$parameters`: All page input parameters from the URL
`vbEnter`	Page or flow	No	None	Dispatched after all container-scoped variables have been added and initialized to their default values, values from URL parameters, or persisted values. The following variable scopes are available: • `$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.
`vbBeforeExit`	Page	Yes	None or `{ cancelled : true }` to cancel navigation	Dispatched to a page before exiting it. A developer can cancel navigation for this event by returning to the listener chain an object with the property `cancelled` set to `true`. This can be useful if the page has to be saved before the user exits it.
`vbExit`	Page or flow	No	None	Dispatched when exiting the page. This event can be used to clean up resources before leaving the page.

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 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 input parameters that you use. (You can only set the value of page input parameters.)

There are two possible results:

- Navigation was successful
- Navigation was canceled 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. 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 Visual Builder, provided by a partner, or created by you, must follow the same component contract. This contract allows Visual Builder to expose the functionality of a component declaratively through the 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 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.

Property Type	Description	Can Be Bound To
One-way (read-only)	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.	Expressions (which may contain variables)
Two-way (read/write)	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.	Variables

Events

A component can fire zero or more events (for example, an onClick event for a button). Each event has a payload. The 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 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 Visual Builder user interface provides access to two basic kinds of data:

- Business objects in Visual Builder, which you can create yourself and use in applications
- Service connections, which can be Oracle services or external REST services

When you create a 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 in OpenAPI/Swagger 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 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 Visual Builder, see Introduction to Accessing Business Objects in *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 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.

Part II
Connect Applications to Data

All business applications require some type of data, and when working with applications in Oracle Visual Builder, you can connect to multiple sources of data—as long as you're able to access them through REST.

Visual Builder provides easy access to two basic data sources:

- *Business objects* which are data objects you create to implement your app's business logic. A business object is a complex data type that groups together related data. For example, if you're creating an application that requires information about an employee, you might create a business object that stores the name, address, salary, and other information about the employee.

  When you create a business object, a REST API is automatically created for you (with GET, POST, PATCH, and DELETE endpoints). You interact with the data in your business object through these REST endpoints. Your data is by default stored in an embedded database associated with your Visual Builder instance, but you can use your own Oracle database if an administrator has configured one for you.
- *Service connections* which are data objects that you connect to. They can be endpoints exposed by Oracle services, like Oracle Cloud Applications or Oracle Integration, as well as external REST services. For example, your employee application can use a service connection to access data exposed by HCM endpoints in the built-in service catalog, even access an external REST service to get information about the country that an employee works from.

No matter what form your data takes (business objects or service connections), the basic principles of creating an application are the same. The key difference between the two data sources: business objects store data as part of the app itself, service connections receive data from REST APIs.

If you know the structure of your objects or the data sources you want to use in your application, you can start defining your application's business objects, service connections, or both.

Topics:

- Work With Business Objects
- Manage Service Connections

4

Work With Business Objects

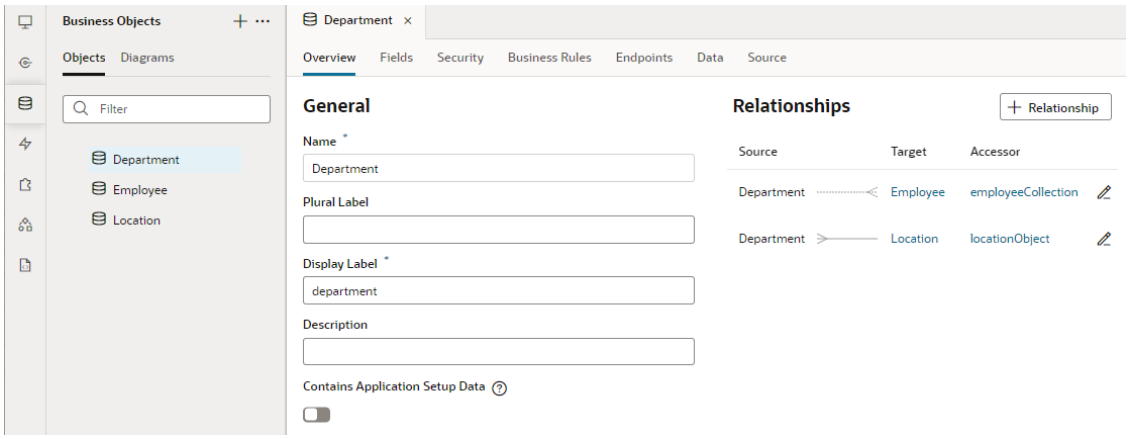
A business object is a resource, such as an invoice or purchase order, similar to a database table; it has fields that hold the data for your application. Like a database table, a business object provides the structure for data. Business objects are stored in a database. The apps in your visual application and other clients access the business objects via their REST endpoints.

About the Business Objects Pane

The Business Objects pane in the Navigator lists all the business objects that are available for use in your application.

You can view your business objects, create new ones, and open pages where you can edit object details. You can also create diagrams to visually represent business objects and their relationships (see Work with Business Object Diagrams).

Use the + button on the Business Objects pane to create business objects and business object diagrams. Use the Menu option to open the Data Manager, where you can work with business object data (see Work with the Data Manager).



After a business object is created, you can select it in the Business Objects pane to view and edit details such as fields and data, related objects, and security settings. The following table describes the tabs in a business object's editor:

Tab	Description
Overview	Displays the business object Id and contains fields for specifying the singular and plural forms for the label used to identify the business object. It also includes the following options: • **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 and Stage and Publish Visual Applications.
Fields	Contains a table displaying the fields defined for the business object. The tab contains a **+ Field** button for defining new fields. You can select a field in the table to edit its properties in the editor. See Edit Business Object Fields. Every business object you create includes these default fields: an `id` plus audit fields that provide information on who created and updated the object and when. It also includes a version number used in the generation of an entity tag (ETag) which protects against users overwriting changes.
Security	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	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	Displays a list of endpoints 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. See Work With Query Parameters. To expose only those endpoints that your application requires, see Add or Remove Exposed Endpoints. To define a caching strategy that safely stores the business object's data, see Control Data Caching for Business Objects. To allow other clients and applications access to the APIs using basic authentication, see Manage User Roles and Access and Allow External Access to Your Business Objects.
Data	Displays data stored in the business object's fields. The tab contains tools for adding and editing the data. See View and Edit Data in Business Objects.
Source	Displays JSON metadata that describes the business object. This source view is typically read-only and you won't make changes here, except to fix issues such as merge conflicts.

Create and Edit Business Objects

You use business objects to store data that is not provided by a service connection. As you develop your applications, you can create business objects and edit the business object fields to meet your needs.

> **Tip:**
>
> The Business Objects pane in the Navigator is where you'll create and manage your business objects. While the **Objects** tab provides a standard view of business objects, you can use the **Diagram** tab to view a visual representation of business objects and their relationships. You can also create and manage business objects using the Diagrammer. See Create Business Objects with the Diagrammer.

Create a Business Object

You can create business objects to store data in the database that was provisioned for your service instance.

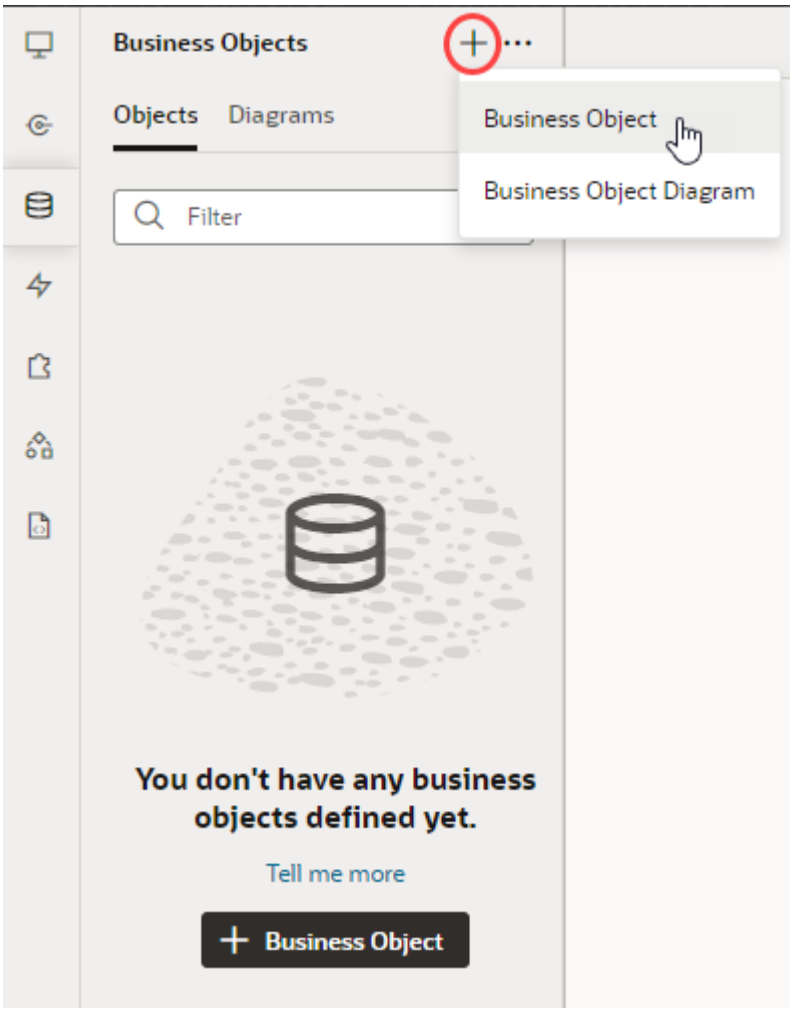
When you create business objects, you specify the fields that your application needs. As you develop your application, you can modify your business objects to add and modify fields as needed. Your business object will be exposed as a set of endpoints that provide REST APIs for operations that you can 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 and select **Business Object**.



3. In the New Business Object dialog box, enter a **Name** for the business object (for example, `MyObject`). The **Display Label** is automatically populated based on the name you enter (for example, `My Object`); update it as needed. Click **Create**.

The newly created 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. The Endpoints tab shows the endpoints that are created by default for the business object.

To delete a business object, right-click the business object and select **Delete**.

Add Fields to Business Objects

You can use the Fields tab of the Business Object editor 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 **+ Field** button, then select **Field**.
3. In the **Label** field, enter the name of the field (which is set as the display label).

   The **Field Name**, by default, is filled in based on the field label, but can be changed if you want.

   > **Note:**
   >
   > Do not use "`items`" as a field name; it's a reserved keyword used for the array that contains the fields in the business object's endpoint description and should not be used as a field name.

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.

   For a string type field, make sure your data doesn't exceed 4000 bytes.

5. Click **Create**.

Edit Business Object Fields

You can use the Fields tab of the Business Object editor to modify your business object's fields. You can do this to add functionality to a field, or even to fix errors relating to invalid property values. Fields that contain errors will be underlined in red on the Fields tab.

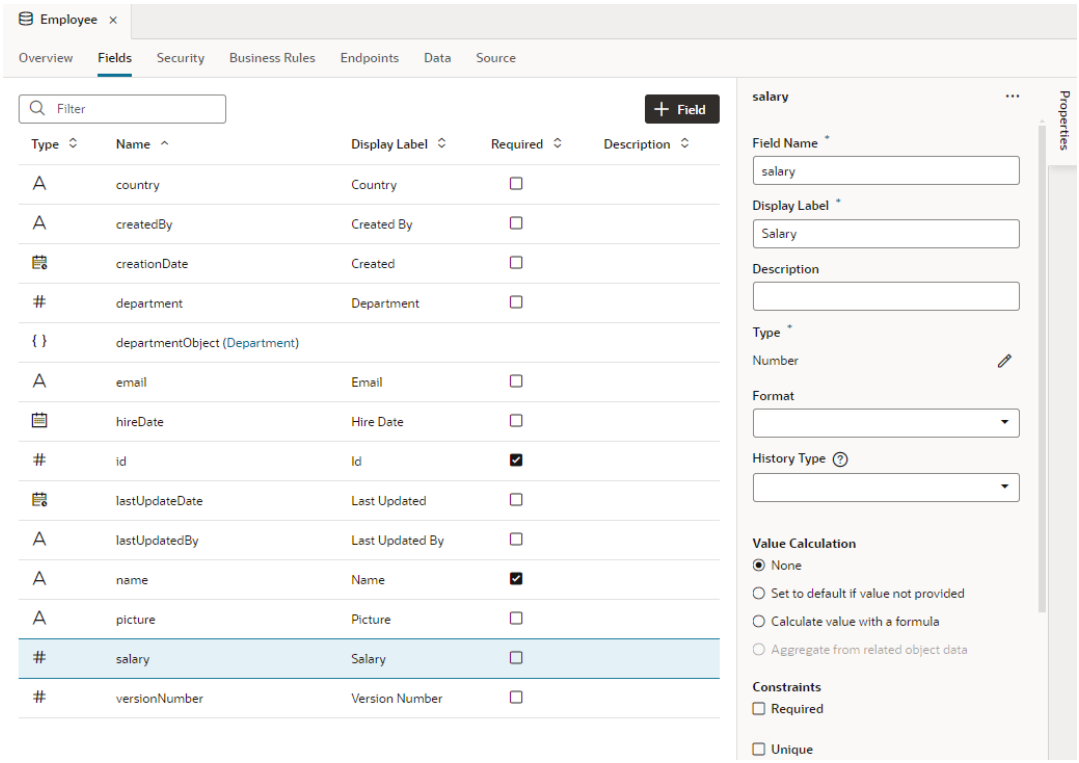
You can edit the fields of business objects when your application is in Development status. To edit the properties of a field:

1. Open the **Fields** tab of the business object that you want to edit.

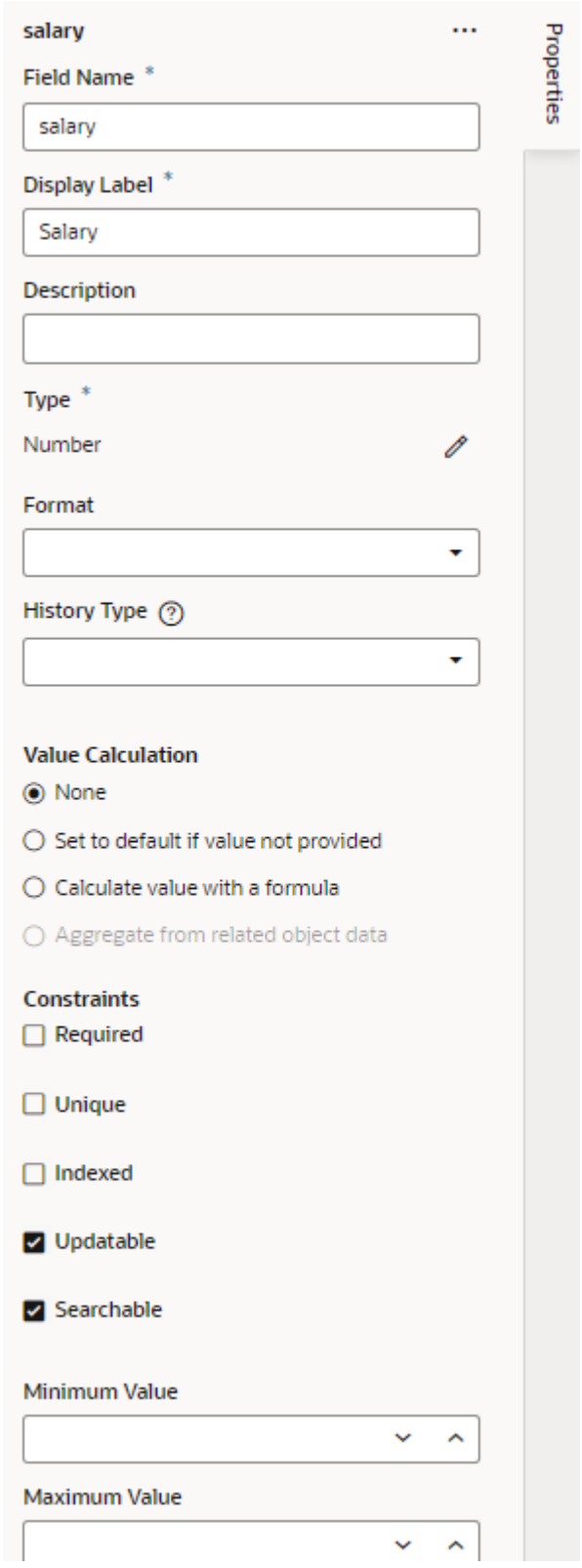
   The Fields tab displays a table that lists all the fields that are defined for the business object.

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:



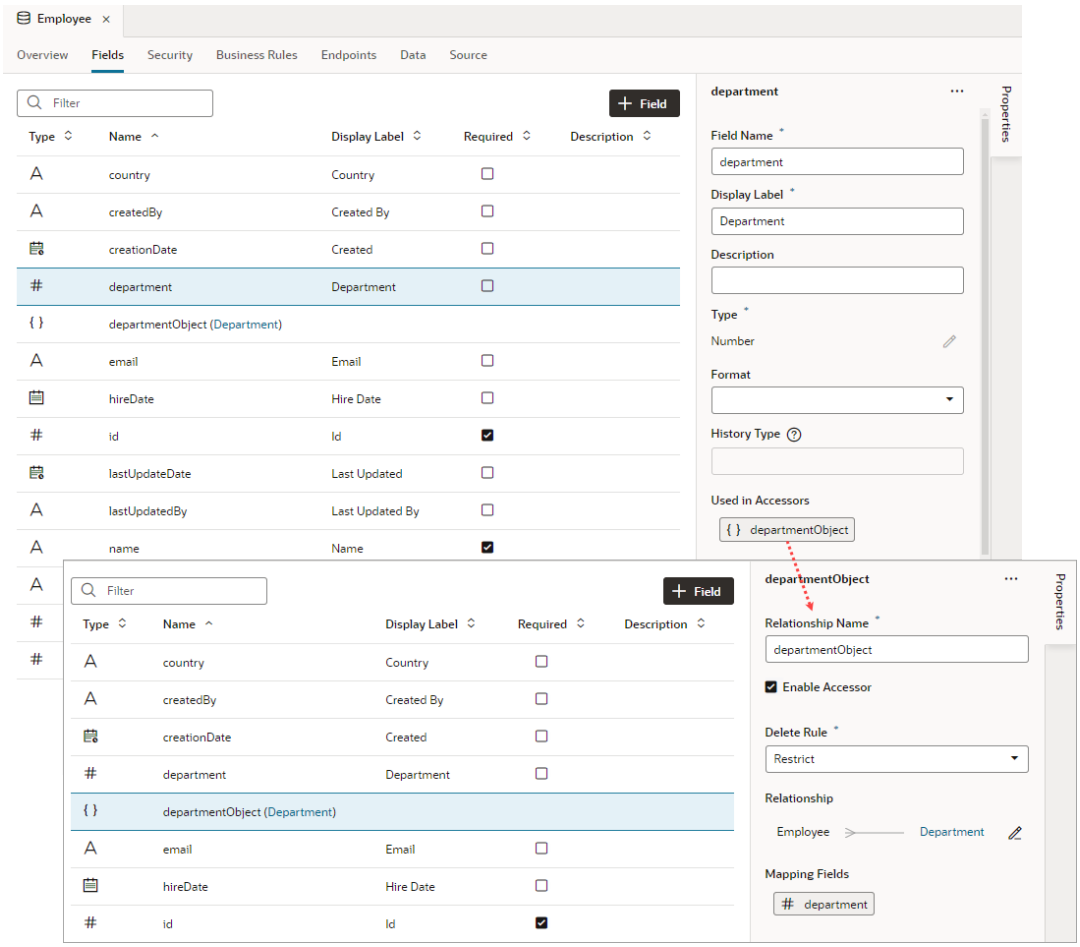
3. Edit the field's properties.



The properties that are available will depend upon the data type of the field. You might be able to specify a default value for the field using a static value, an expression, or a formula. You can also specify one or more of the following field constraints: Required, Unique, Indexed, Updatable, and Searchable.

Depending on the field's data type, you can set values for the Minimum Length, Maximum Length, and so on. If invalid values are used, the field's name will be underlined in red on the Fields tab.

When your business object references other objects, you can view and edit (if necessary) the properties of the accessor that lets you access the related business object. (You can click the link in the accessor field to directly access the related object.)



> **Note:**
>
> Depending on the field's data type, you might also see the History Type property, used to audit a business object's history. While the option does exist, it isn't particularly relevant for objects you create because audit fields are automatically added for every business object you create. The property is more useful when you want to set your own audit fields for business objects created from a file or those that use your own database schema.

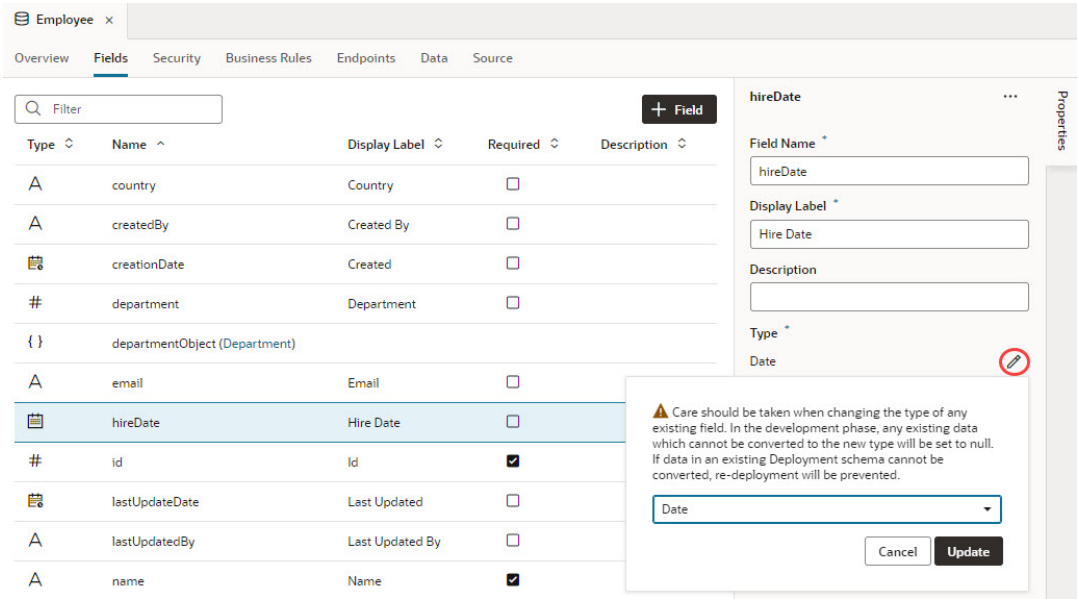
Change a Field's Data Type

You can change a field's data type even after it's been created. This option is useful when you want to create a business object that uses a non-numeric key.

> **Caution:**
>
> Consider the impact of changing an existing field's data type, especially that of a key field. When a field's data type is changed, Visual Builder tries to convert the existing data into the new type, for example, converting a string `"10.5"` to its numeric equivalent `10.5`. In the development phase, on a record-by-record basis, if the data cannot be converted to the new type, it will be set to null—other than for a key field. If the key field's data cannot be converted, the entire record will be deleted. If data in an existing deployment schema cannot be converted, you won't be able to stage or publish your app unless you undo the type change, fix the data in the deployment schema to remove any incompatible data, or redeploy choosing the option to create a new schema.

To change the data type of a field:

1. Open your business object's **Fields** editor and select the field whose type you want to change.
2. Under **Type** in the Properties pane, click next to the current data type.



Type changes for a business object's key field are not allowed if the object has incoming relationships. For example, when a `department` field in the Employee business object references the Department object (via its `Id` field), Department is said to have an incoming relationship with Employee. In this case, you won't be able to change the data type of the Employee object's `department` field as well as the Department object's `Id` field.

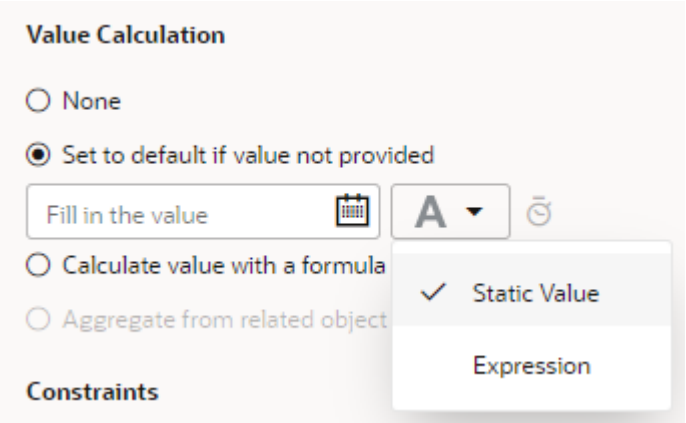
3. Select the type you want to change to and click **Update**.

Set a Default Value for a Field

You can set a default value for a field if the user does not provide a value for it. Based on the field's data type, you might be able to set a static value or an expression as the default.

To set a default value for a field:

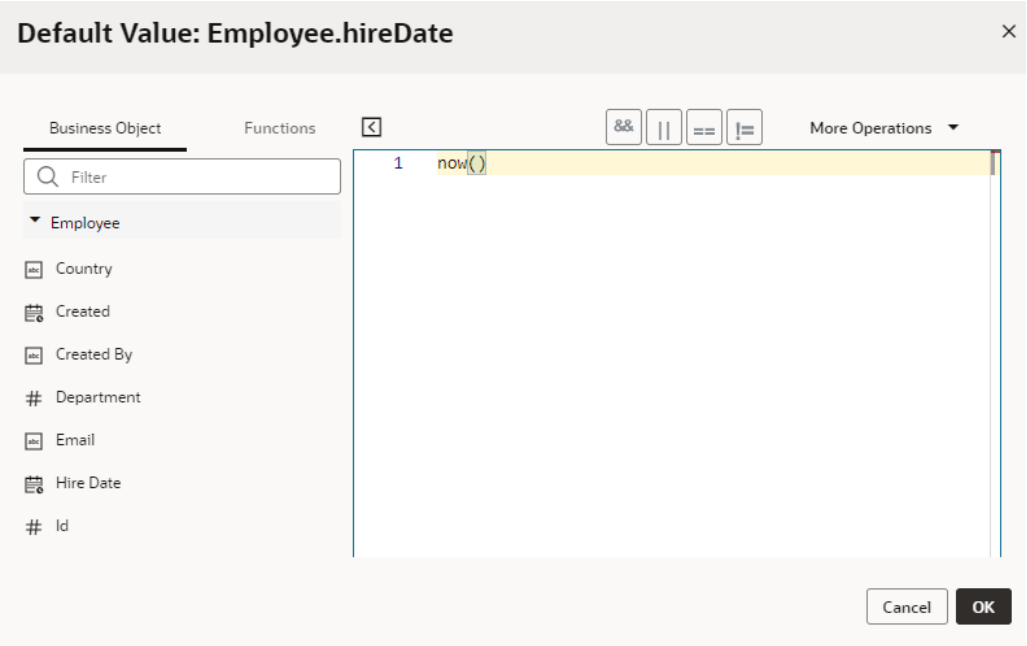
1. Open your business object's **Fields** editor and select the field you want to modify.
2. In the field's Properties pane, select **Set to default if value not provided**.
3. Choose to define a static value or construct an expression:
   - To set a static value, select **Static Value** (selected by default) and enter a value:



   - To set an expression, select **Expression** from the drop-down list, then construct an expression in the Expression Builder.

     Specify operands by typing in the text area or click an operator in the toolbar to add it to your expression. You can also select the Insert arrow for a field in the Business Objects tab or for a function in the Functions tab.

     For example, to set the current date as the default hire date for an employee, you might enter `now()` as the expression for the Hire Date field:



Click **OK**.

A field's default value won't be visible to users when they access the field. That's because the value is added just before the record is stored in the business object's database.
If you defined a Groovy expression for a field's default value, you can optionally override the default Groovy timeout by clicking the icon in the Properties pane.

Add a Formula to a Field

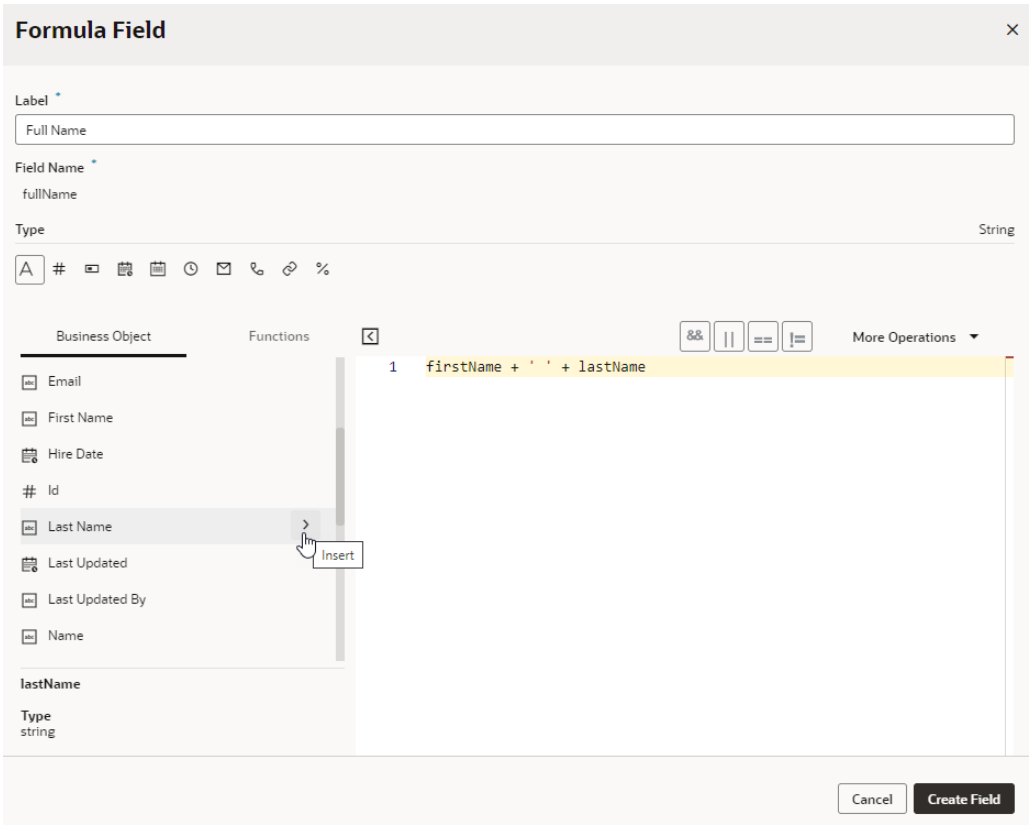
You can add a formula to calculate the value of a business object's field. You can create a formula to calculate a numerical value, such as a percentage, or you can create a Groovy expression that uses available fields to generate a value. For example, you can concatenate strings stored in local fields (`firstName + ' ' + lastName`) or determine a value based on a comparison or logical expression (`qualityLevel != 5`).

You calculate a field's value with a formula in one of two ways: by creating a new formula field or by editing an existing field and adding a formula to calculate its value.

- To create a new formula field:
  1. Open your business object's **Fields** editor.
  2. Click the **+ Field** button, then select **Formula Field**.
  3. In the Create Formula Field dialog box, enter a label for the formula field you want to create, then select the field's Type.

     (The Field Name is automatically filled in based on the field label.)
  4. Enter a valid expression for the formula in the text area.

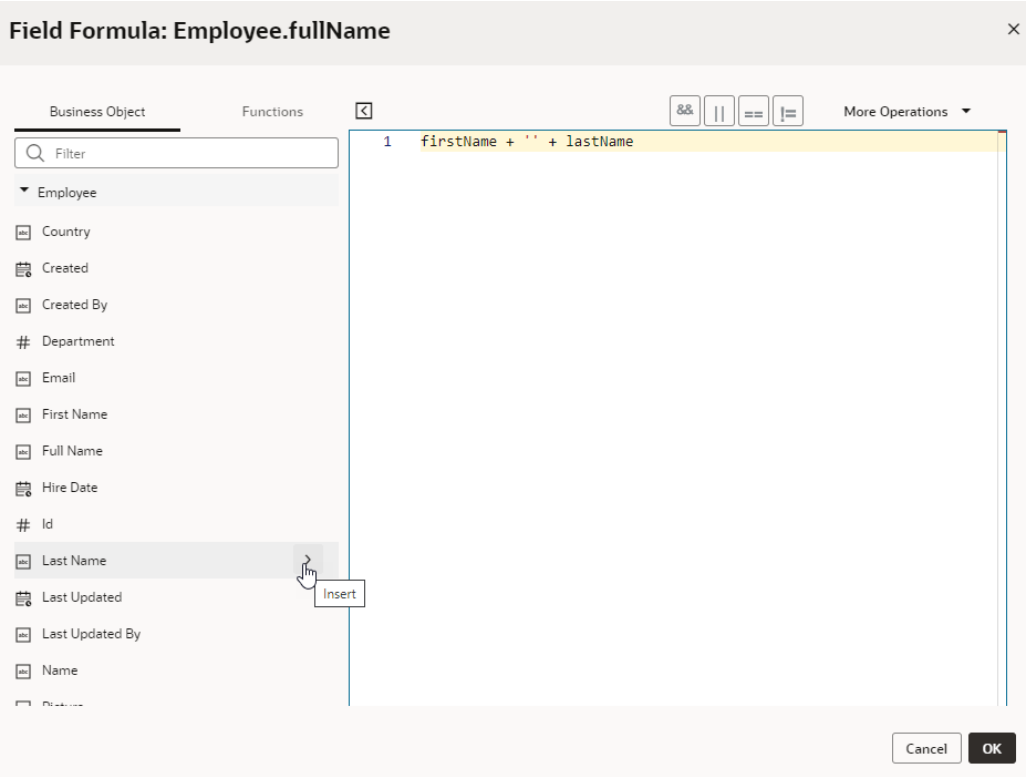
     The formula you enter must be a valid expression. You can specify operands by typing in the text area or by selecting the Insert arrow for a field in the list of Available Fields. You can also click an operator in the toolbar to add it to the formula.



Click **Create Field**.

The new field is created and its formula displayed in the Properties pane.

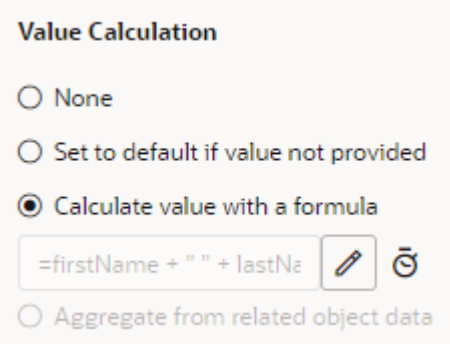
- To add a formula to an existing field:
  1. Open your business object's **Fields** editor and select the field you want to modify.
  2. In the field's Properties pane, select **Calculate value with a formula**.
  3. In the Field Formula expression builder, enter a valid expression for the formula in the text area. 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 also click an operator in the toolbar to add it to the formula.



Click **OK**.

The formula is saved and displayed in the Properties pane.

Once your formula is created, it will be displayed in the Properties pane:



Click next to the formula to edit it in the Expression Builder. The expression you supply is evaluated at runtime to return the field's value each time it is accessed. And when the field is populated, you'll see the value on the **Data** tab. A field whose value is calculated using a formula is read-only.

When you define a Groovy expression for formula fields, you can optionally override the default Groovy timeout by clicking the icon in the Properties pane.

Add a Field for Aggregating Data

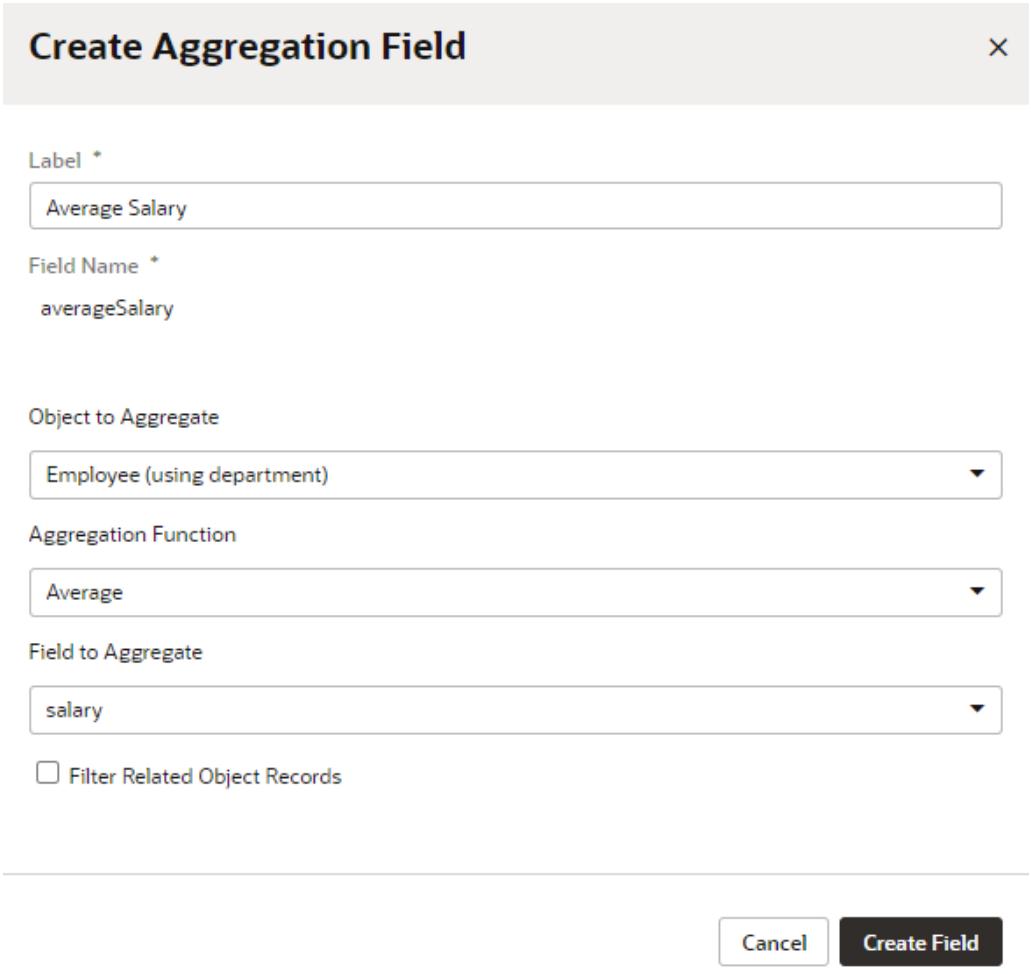
Use aggregation fields in your business objects to aggregate the data of related business objects, for example, to calculate and store the total number of items in an order. You can

calculate and store the values of simple operations such as calculating an average or sum, or counting the number of related business objects.

You aggregate a field's data in one of two ways: by creating a new aggregation field or by editing an existing field and adding an aggregation function. For both options, your business object must have a related object with fields that can be aggregated; in other words, a business object with incoming relationships. Let's say you create a field in the Employee business object that refers to the Department object; Employee is the source object and Department is the target object. Department, in this case, is said to have an incoming relationship with Employee.

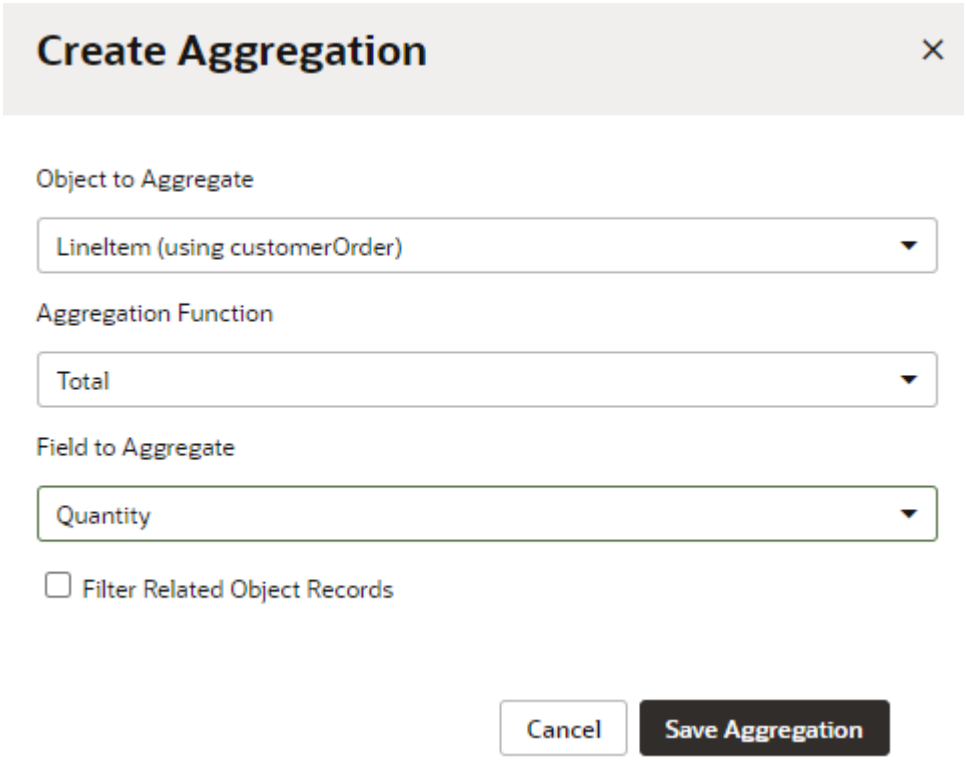
- To create a new aggregation field:
  1. Open your business object's **Fields** editor.
  2. Click the **+ Field** button, then select **Aggregation Field**.
  3. In the Create Aggregation Field dialog box, enter a label for the aggregation field you want to create.

     (The Field Name is automatically filled in based on the field label.)
  4. Select the object to aggregate (for example, the Employee business object which uses Department as a target business object), the aggregation function (which can be Average, Count, Maximum, Minimum, or Total), and the field to aggregate (for example, Salary):



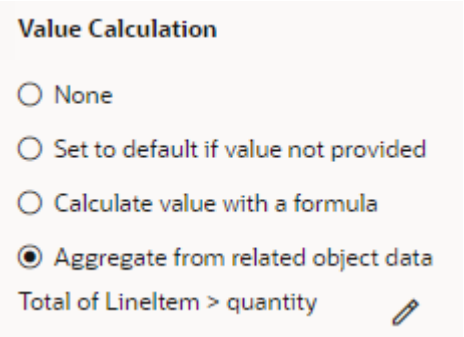
You can also select **Filter Related Object Records** to create a filter that limits the fields that are aggregated.
Click **Create Field**.

- To create an aggregation for an existing field:
  1. Open your business object's **Fields** editor and select a field that can be aggregated.
  2. In the field's Properties pane, select **Aggregate from related object data**.
  3. In the Create Aggregation dialog box, select the object to aggregate (for example, the LineItem business object which is referenced by the customerOrder object), the aggregation function (which can be Average, Count, Maximum, Minimum, or Total), and the field to aggregate (for example, Quantity):



You can also select **Filter Related Object Records** to create a filter that limits the fields that are aggregated.
Click **Save Aggregation**.

Once your aggregation formula is created, it will be displayed in the Properties pane:



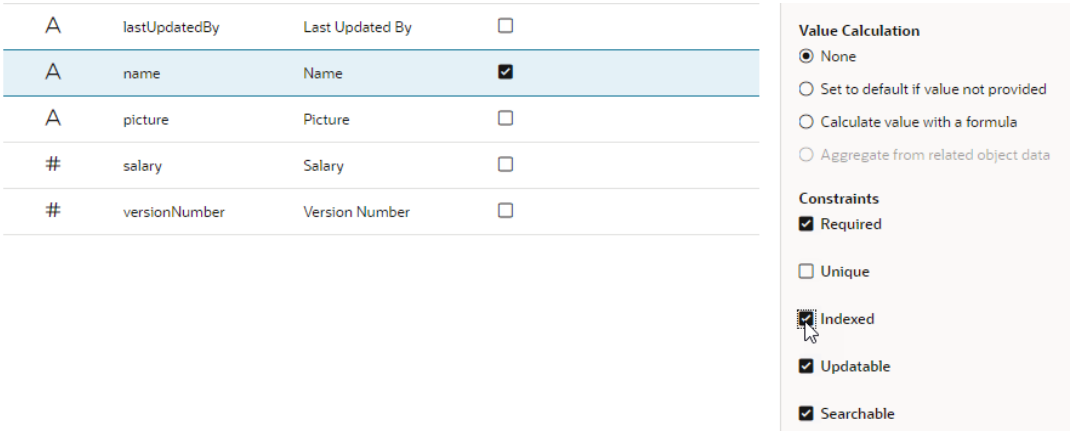
Click next to the aggregation formula if you want to edit it. When the field is populated, you'll see the aggregated value in the Data tab.

Index a Field

You can index a business object's fields to speed up searches, especially if the field contains thousands of records that are frequently accessed. Indexing a field adds a non-unique index to the field's database column and improves performance when you search for the field's value via REST requests or Groovy code.

To index a business object's field:

1. Open your business object's **Fields** editor and select the field you want to index.
2. Under Constraints in the Properties pane, select **Indexed**.



This property does not show for the Id field, a Reference type field, or one that's marked unique because these fields are implicitly indexed.

View, Create, and Edit Business Object Relationships

Define business object relationships to more easily create pages with parent-child relationships or drop-down lists that reference another business object's values. These relationships can be one-to-one, many-to-one (and one-to-many), and many-to-many.

It's possible to create, edit, and delete relationships from the business object's Overview page as well as in the Diagrammer, both of which show *all* relationships of the current object. When a business object field references another object, you can also edit the *specific* relationship from the referenced field's Properties pane.

The Diagrammer helps you visualize business object relationships. To define a relationship when working with a business object diagram, see Create Relationships with the Diagrammer.

Create a Many-To-One or One-To-One Relationship

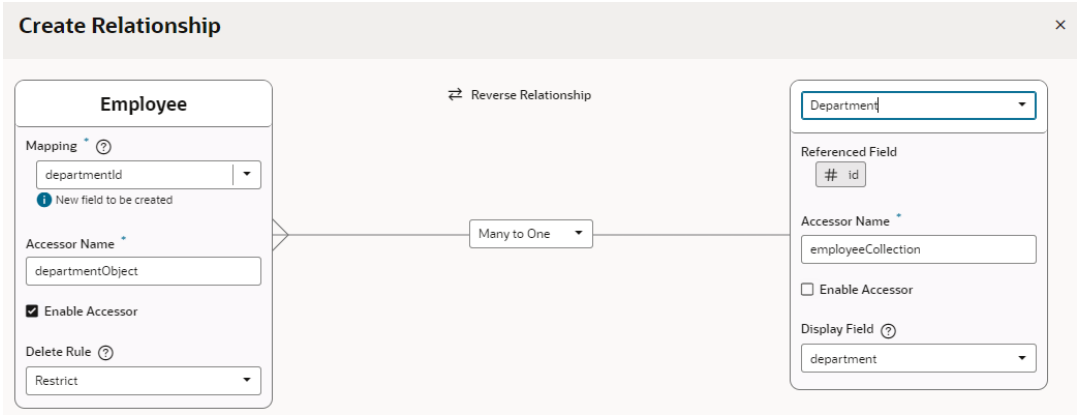
You can create a many-to-one or one-to-one relationship from a business object's Overview tab. Here are examples of a many-to-one (or one-to-many) and a one-to-one relationship:

- A many-to-one relationship between the Employee and Department business objects means a department can have many employees but each employee can only work for one department.

- A one-to-one relationship between the Employee and EmployeePII objects means that each employee has one set of sensitive personally identifying information and that information belongs to only one employee.

To create a many-to-one or a one-to-one relationship:

1. Go to your source business object's **Overview**.
2. Click **+ Relationship**, then select **Many to One / One to One**.
3. In the Create Relationship dialog, select the target business object.

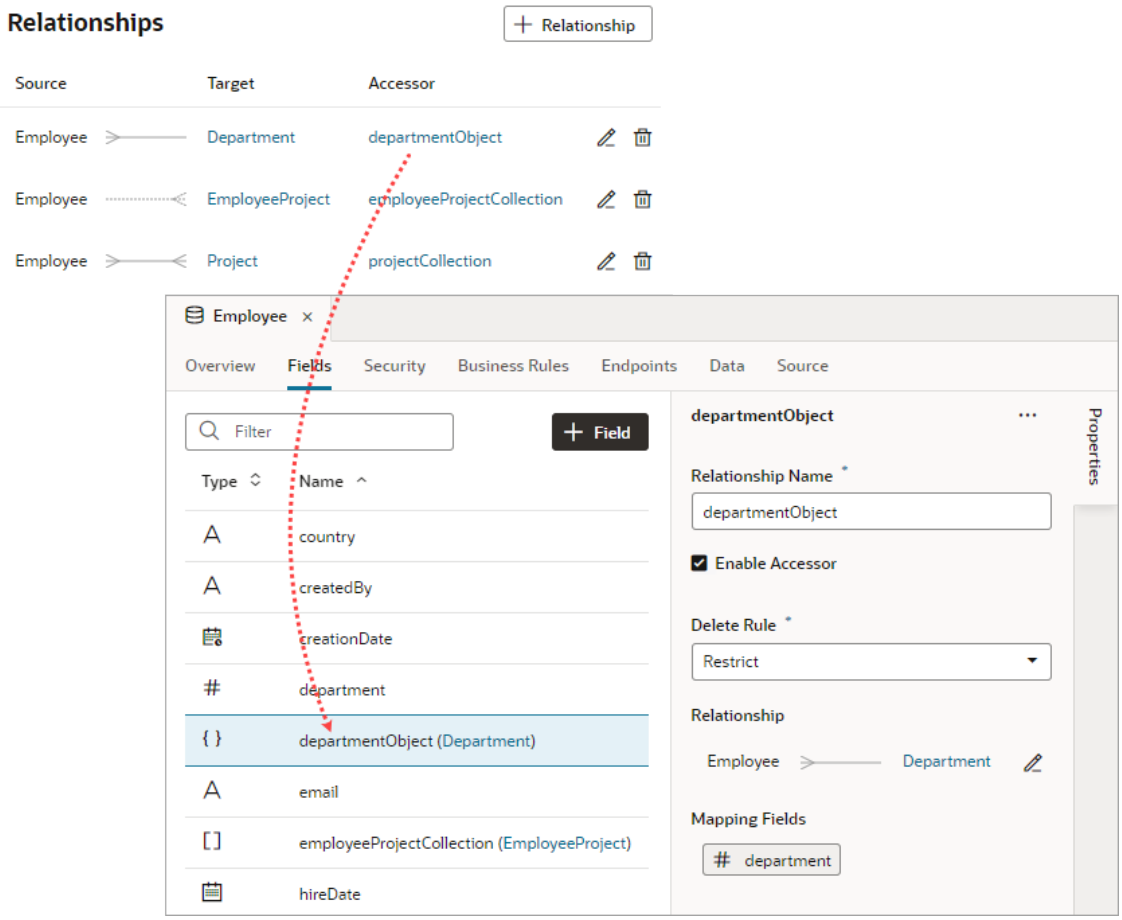


4. Check the cardinality between the objects. The default cardinality between objects is many-to-one.
   - To create a one-to-many relationship between A and B, you need to create a many-to-one relationship between B and A. Click **Reverse Relationship** to switch the position of objects in the relationship.
   - To create a one-to-one relationship, change the cardinality from **Many to One** to **One to One**.
5. If needed, change the target object's **Accessor Name**, which you'll need to issue REST calls.
6. If you want to access the source object from the target object, select **Enable Accessor** for the target object.

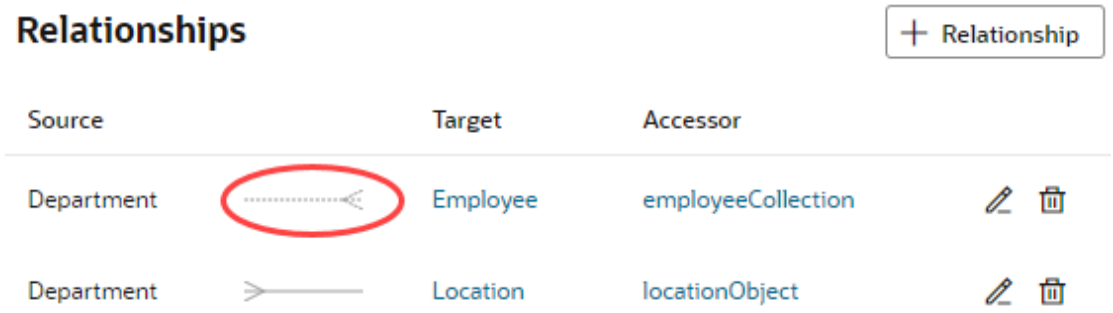
   By default, the accessor from the source object to the target object is enabled (`departmentObject` for Employee, in our example). This means that if you were to drag the Employee object onto a page to create a table, you'll be able to fetch the employee's details, including the department they work in. But if you want to access Department from Employee (say, to fetch employees in each department), you must enable the accessor for the target object (`employeeCollection` for Department, in our example).
7. Select a **Display Field** for the target object. The best choice for this option is a string field that's both unique and required. Look for such fields under **Unique Required String Fields** in the drop-down. If none exist, you might need to select a suitable field; by default, the first field listed under **Other Fields** is selected.
8. Now in the source object's **Mapping** field, specify the new field that will be created as the referenced key (default), or select an existing field. A referenced key is like a foreign key in a database table. It's a field that refers to the key (say, the Id field) of another business object to link the two business objects together.
9. If needed, change the automatically generated **Accessor Name**.

10. Keep **Enable Accessor** selected for the source object to retain access to the other business object.
11. Select a **Delete Rule** to determine what happens when a record that has a relationship to another record is deleted. Your choices are:
    - **Restrict** (default): You aren't allowed to delete a parent that has children (the default rule for a many-to-one relationship)
    - **Cascade**: When you delete a parent, the children of that parent are automatically deleted
    - **Set To Null**: When you delete a parent, the mapping field in the children is set to null.
12. Click **Create**.

The new relationship is displayed in the Relationships list on the Overview page, along with the accessor that lets you access the business object at the other end of the relationship. Clicking the accessor name will take you to the Fields tab, where you'll see the accessor listed along with other fields.



For referenced business objects, the relationship between the current business object and the one being referenced is, by default, many-to-one. When you create a reference from, say, an Employee business object to a Department business object, an Employee can belong to only one Department, but a Department normally has many employees. Such a relationship 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:



The referenced business object, Department, also appears in the Endpoints tab for the Employee business object.

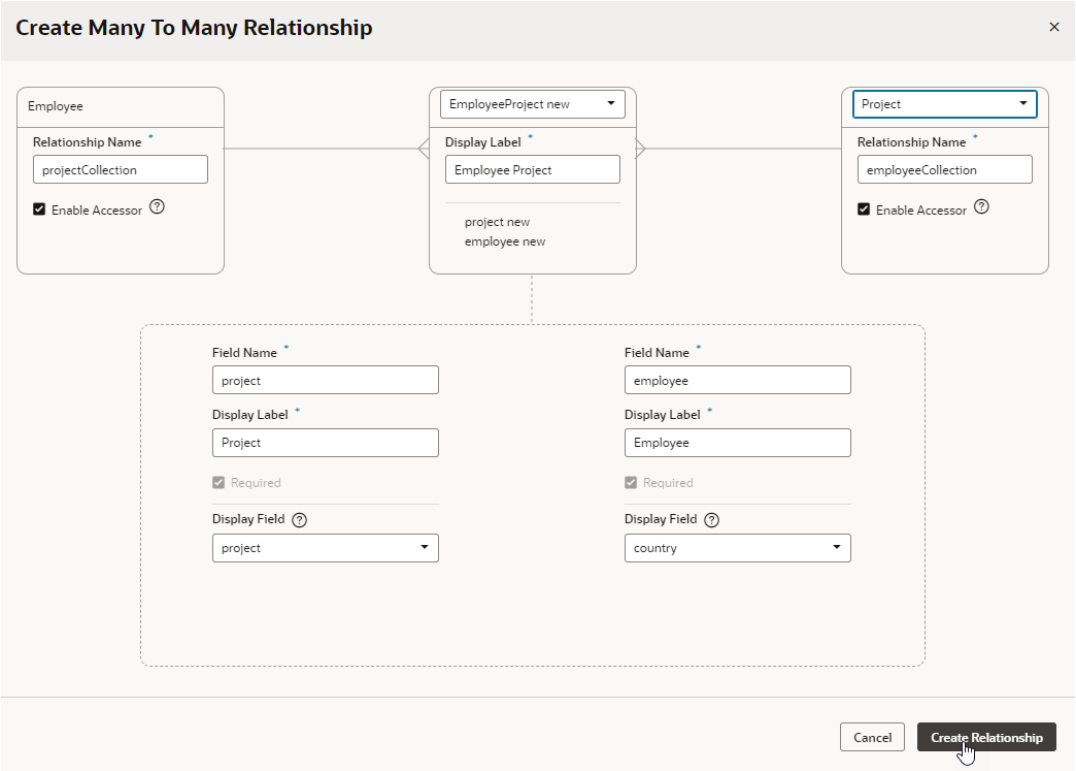
Create a Many-to-Many Relationship

Create a many-to-many relationship from a business object's Overview tab. When you create a many-to-many relationship between the Employee and Project business objects, an employee can work on multiple projects, and each project can have many employees.

To create a many-to-many relationship:

1. Go to your business object's **Overview**.
2. Click **+ Relationship** next to Relationships, then select **Many to Many**.
3. In the Create Many to Many Relationship dialog, select the business object you want to create a relationship with.

   Because you chose a many-to-many relationship, an intersection business object with two reference fields (in our example, `EmployeeProject` with the `project` and `employee` fields) is automatically created. Its default name is a concatenation of the two business objects (which can be changed if you want). For an intersection business object, you can't deselect the Required checkbox in either of the reference fields.



> **Tip:**
>
> If an existing business object has many-to-one relationships with both source and target of the many-to-many relationship, the business object will appear in the **Select a business object** drop-down list. You can use this object instead of creating a new intersection business object.

The **Enable Accessor** checkbox is selected by default. Keep it selected unless you want to disable access to the other business object.

4. Edit the properties of the relationship field (Field Name, Display Label, and Display Field). Whether you can edit a property and what values are available depends on the nature of the relationship and the objects.
5. Click **Create Relationship**.

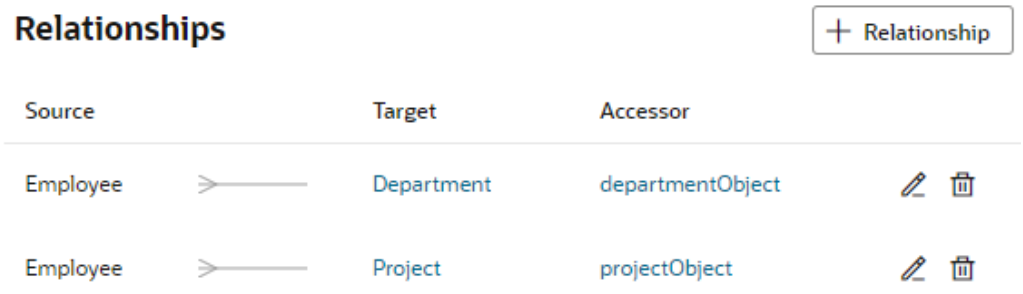
The new relationship is displayed in the Relationships list on the Overview page, along with the accessor that lets you access the business object at the other end of the relationship. Clicking the accessor name will take you to the Fields tab, where you'll see the accessor listed along with other fields.

Edit a Business Object Relationship

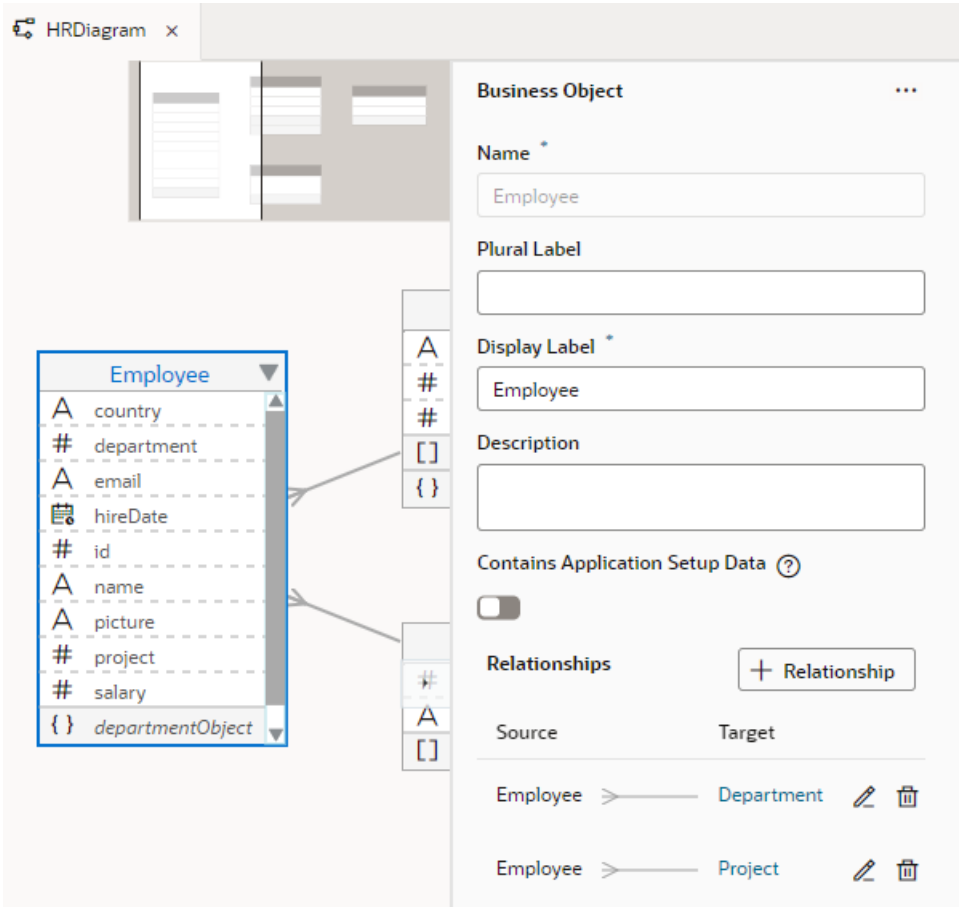
You can edit the details of a business object's relationship to make changes as required. Whether you can edit a property and what values are available depend on the nature of the relationship and the objects.

To edit a relationship:

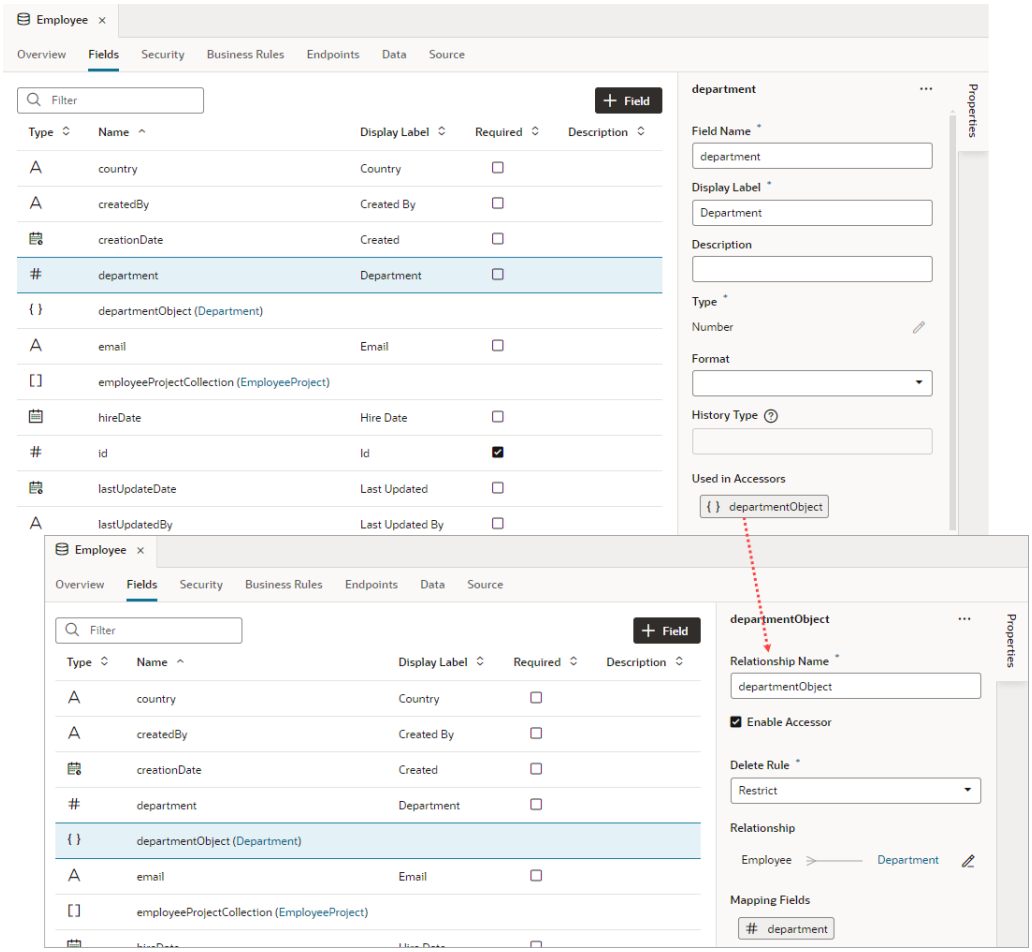
1. Locate the business object relationship to edit. You can edit a relationship from many contexts:
   - From a business object's Overview tab:



   - From a business object's properties in the Diagrammer (see Work with Business Object Diagrams):



   - From a referenced field's Properties pane:



2. Click the **Edit** icon next to the relationship name and make the desired changes.

   Some fields cannot be edited and are disabled. Enable or disable the **Enable Accessor** checkbox if needed. Note that disabling the accessor means you won't be able to access the related business object's information. Additionally, you can specify the Display Label, select the Display Field, and change the selection that makes the field required. You can also specify a Delete Rule.

3. Click **Done**.

Secure Business Objects

Configure a business object's security settings to control the user roles that can access the endpoints and the types of operations they can perform. See Access and Secure Business Objects.

Create Rules for Business Objects

Most applications require rules for business objects to execute business logic that deals with the data. For each business object in your application, you can create business rules that validate objects and fields and that trigger actions based on events or field changes.

To define business rules for each business object, you can create:

- **Object** and **Field triggers** that let you react to data events (for example, when a record is inserted, updated, or deleted). You can use the trigger designer to visually define the conditions and actions that will be executed in those events, or write custom Groovy scripts that define more complex logic.
- **Object** and **field validators** that make sure data at the field or record level is correct.
- **Object functions** that encapsulate logic relating to a business object.

Business rules always run on the server and work the same way no matter how a business object is updated (whether through REST API calls or Groovy scripts).

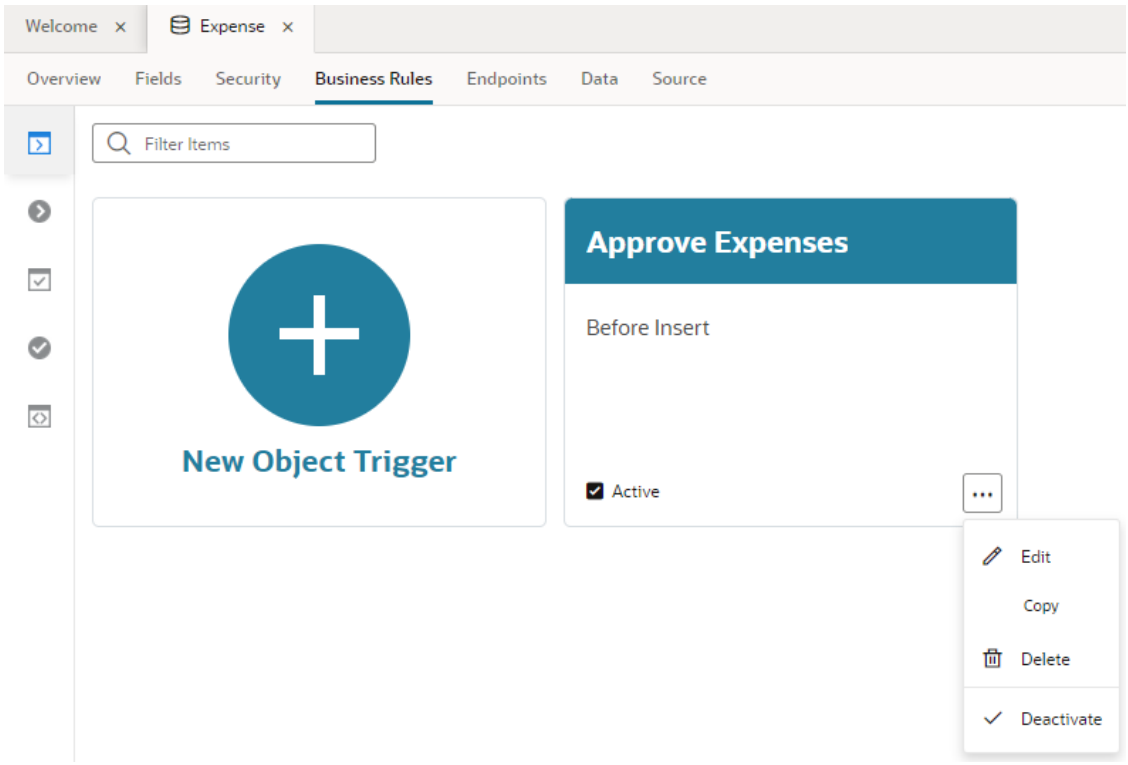
About Adding Business Rules

You can use the trigger designer and code editors in the Business Rules tab to create business rules for your business objects.

On the Business Objects page, you use the Business Rules tab to create and edit business rules for your 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 Rule 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.

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*.

Access the Current User's Details in Your Groovy Script

The recommended way to access the current username in your Groovy script is `adf.context.getSecurityContext().getUserName()`. The only method in the `UserProfile` object that returns a non-null result is `getUserName()`, which is the same value as what is returned from the API above. There is no reason to use `UserProfile` when using Groovy in Visual Builder. For more details, see Referencing Information About the Current User in the *Groovy Scripting Reference*.

Triggers for Business Objects

Triggers are scripts that you can write to complement the default processing logic for 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.

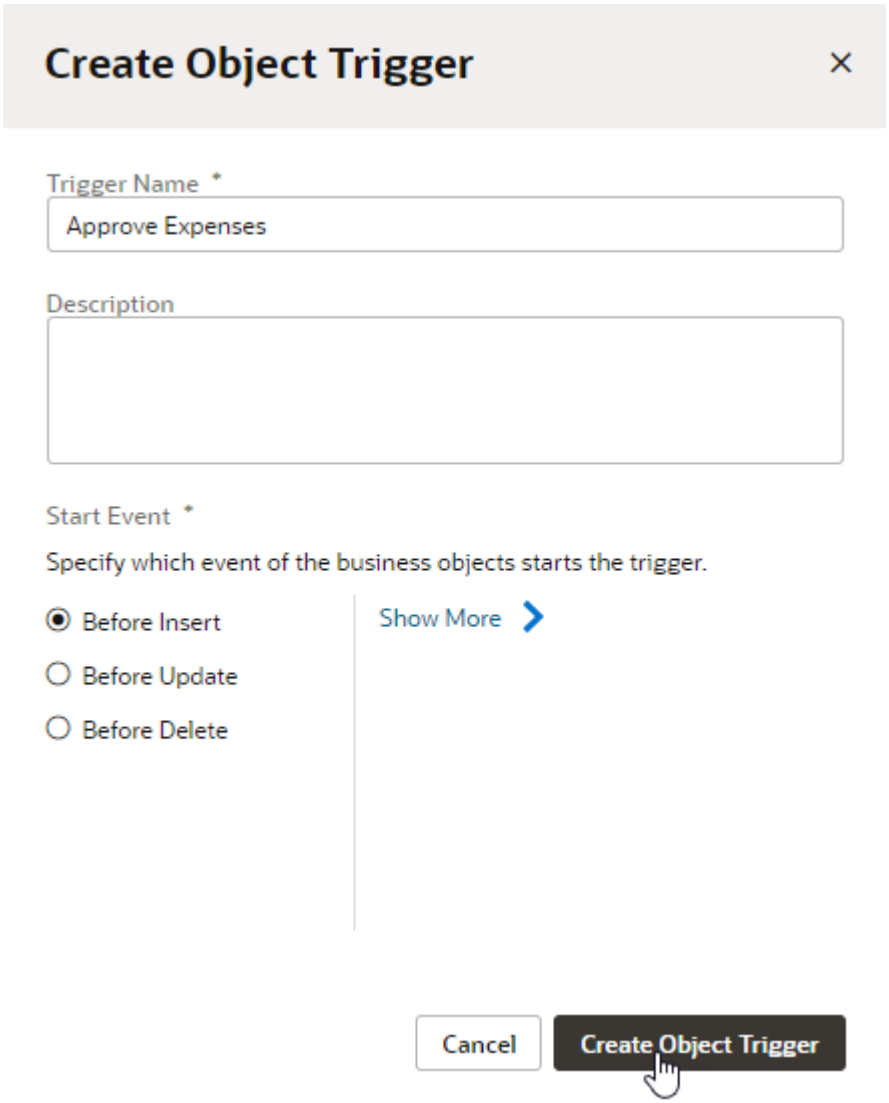
Object Triggers

You can create an object trigger to specify a sequence of actions that starts when a specific event occurs.

A typical event triggering a sequence is adding or updating a record in the business object. When you create a trigger, you specify the trigger event and the actions that are then executed. You can refine the sequence so that actions are executed only when some criteria is met. A sequence can define multiple criteria and multiple actions.

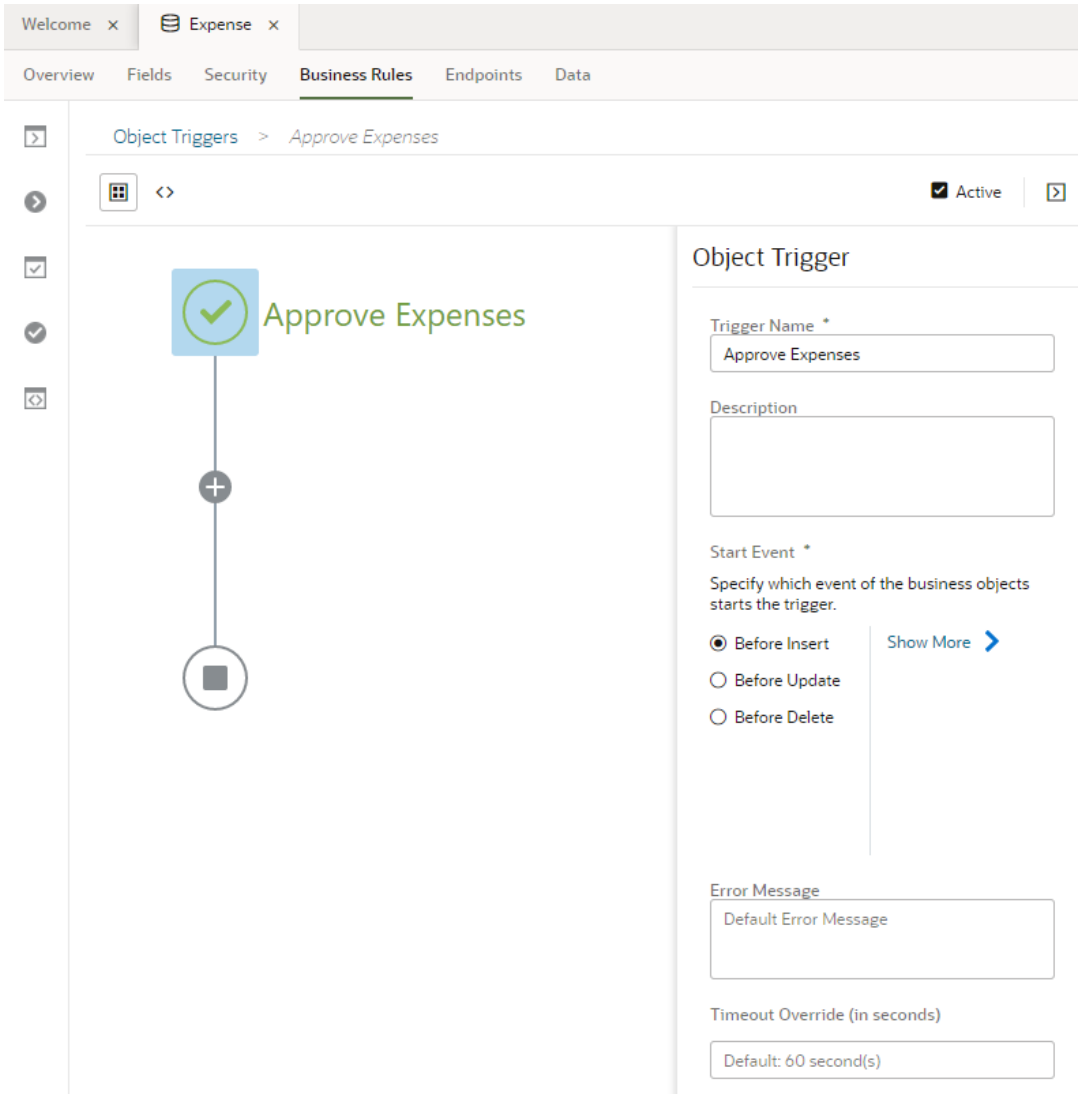
To create an object trigger for a business object:

1. Select the business object to which you want to add an object trigger.
2. Select the object's **Business Rules** tab.
3. Click the **Object Triggers** tab to see a list of all object triggers that are defined for the business object.
4. Click **+ New Object Trigger**, and in the Create Object Trigger dialog box, enter a name to identify the trigger, and select a Start Event for the sequence. You can modify these later if you want.



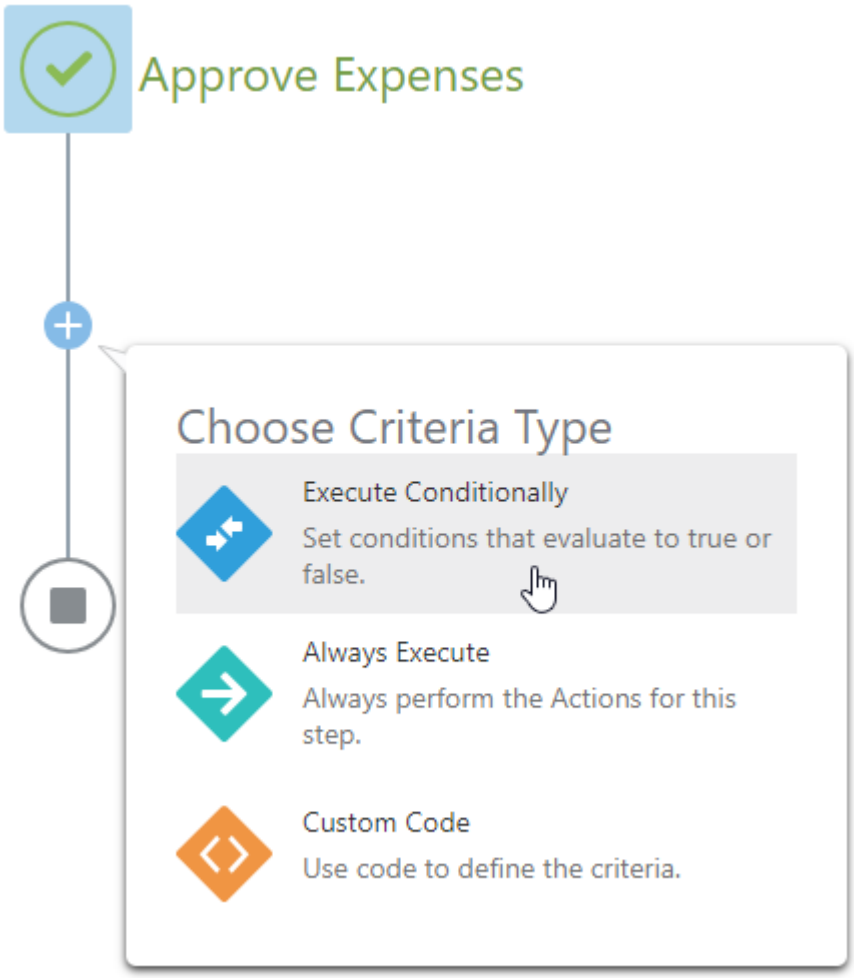
5. Click **Create Object Trigger** to open the trigger designer.

   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.

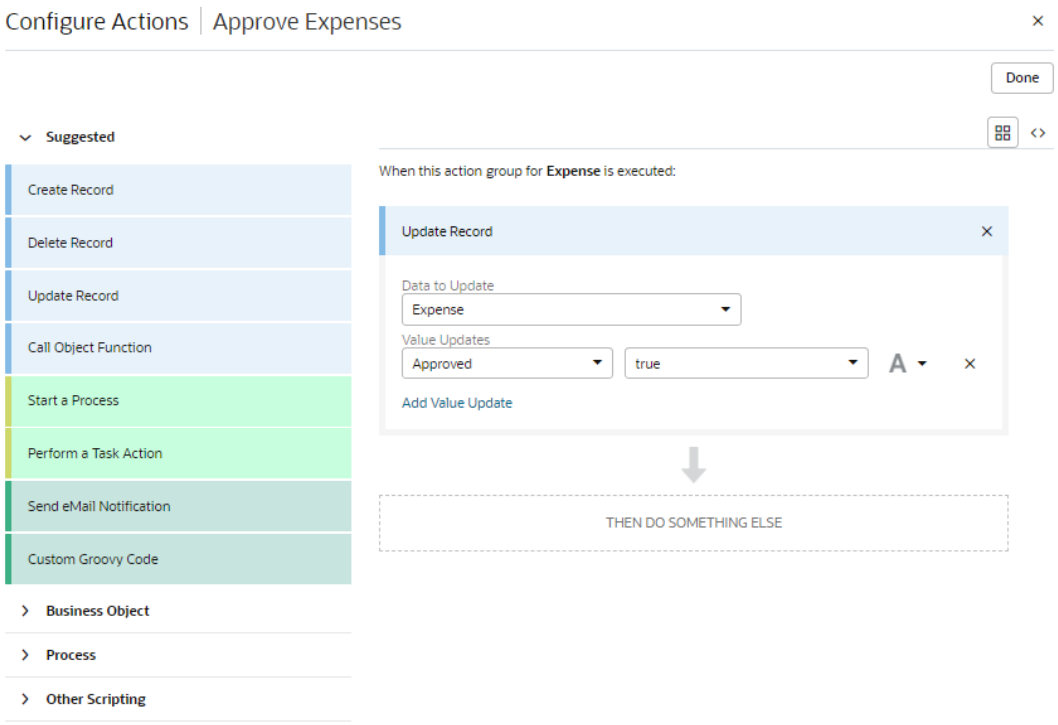


6. Click **Create New Criteria** (the + sign) and choose a criteria type.

   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.



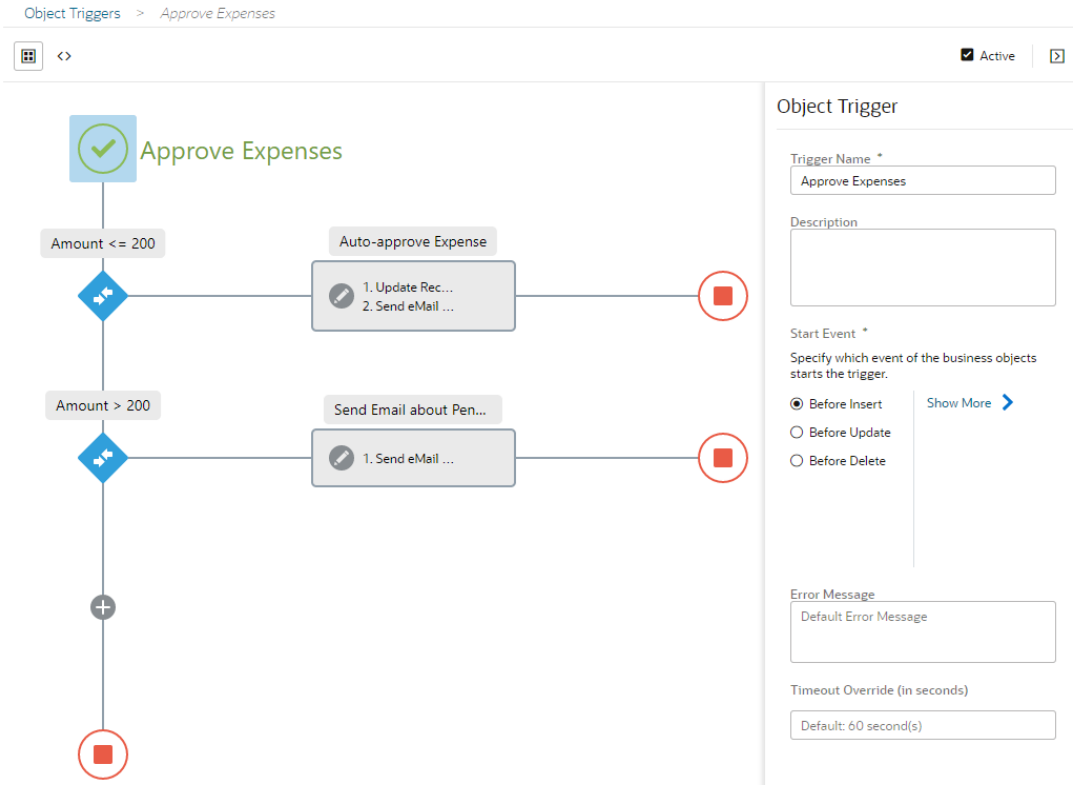
7. If you specify Execute Conditionally, follow the instructions to build conditions for the trigger.
8. Click the **Add Actions** box, then specify a **Name** for the Action Group and click **Add Actions**.
   The Configure Actions page opens when you click Add Actions. You can add one or more actions to the list by dragging predefined functions (such as create a record or perform a process task action) 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 Properties pane 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.



Click **Code Editor** to view the read-only code that is generated by the designer.

Code is read only. If you want to edit the code manually, you have to convert to Custom Code Trigger

```
// Trigger Beginning
println 'Trigger started: Approve Expenses, business object: Expense, event: BeforeInsert';

try {

  // Execute Conditionally Criterion (Amount <= 200)
  def codeCriterion_1 = (amount <= 200);
  println "Trigger condition (amount <= 200) is " + codeCriterion_1;
  if (codeCriterion_1) {
    // Action Group (Auto-approve Expense)
    approved = true;

    eMailNotification_1: {
      def variables = [:];
      variables["amount"] = String.valueOf({ amount }());
      Email.send('myname@oracle.com', "nobody@oracle.com", "Your expense of [[amount]] is a
    };
    return;
  }
  // Execute Conditionally Criterion (Amount > 200)
  def codeCriterion_2 = (amount > 200);
  println "Trigger condition (amount > 200) is " + codeCriterion_2;
  if (codeCriterion_2) {
    // Action Group (Send Email about Pending Approval)
    eMailNotification_2: {
      def variables = [:];
      variables["amount"] = String.valueOf({ amount }());
      Email.send('myname@oracle.com', "nobody@oracle.com", "Your expense of [[amount]] is w
    };
    return;
  }

} finally {

  // Trigger End
  println 'Trigger ended: Approve Expenses';

}
```

When you define Groovy code for object triggers, you have the option of overriding the default Groovy timeout specified in the **Timeout Override** field.

10. When you are finished designing the trigger, click the **Object Triggers** tab to return to the Object Triggers page.

Field Triggers

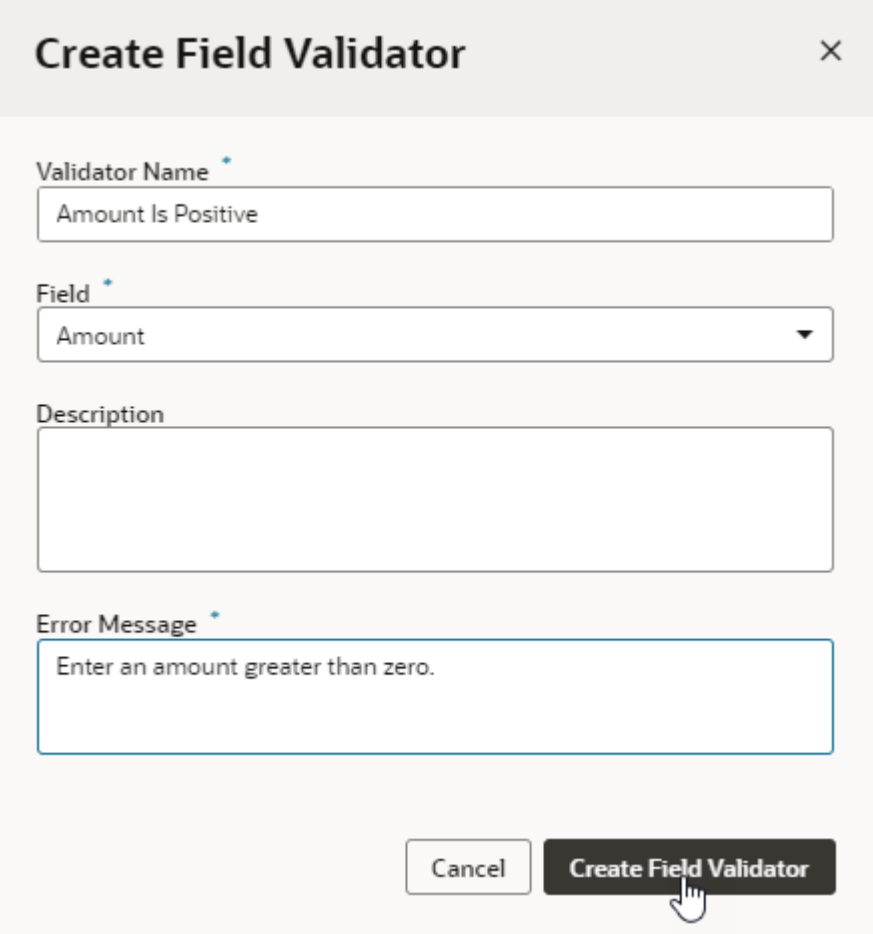
You can create a field-level trigger to define conditions that apply whenever a specific business object field changes in value.

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.
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.



5. Click **Create Field Trigger** 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 + 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 Group 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 Properties pane 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.

Click **Code Editor** to view the read-only code that is generated by the designer.

When you define Groovy code for field triggers, you have the option of overriding the default Groovy timeout specified in the **Timeout Override** field.

10. When you are finished designing the trigger, click the **Field Triggers** tab to return to the Field Triggers page.

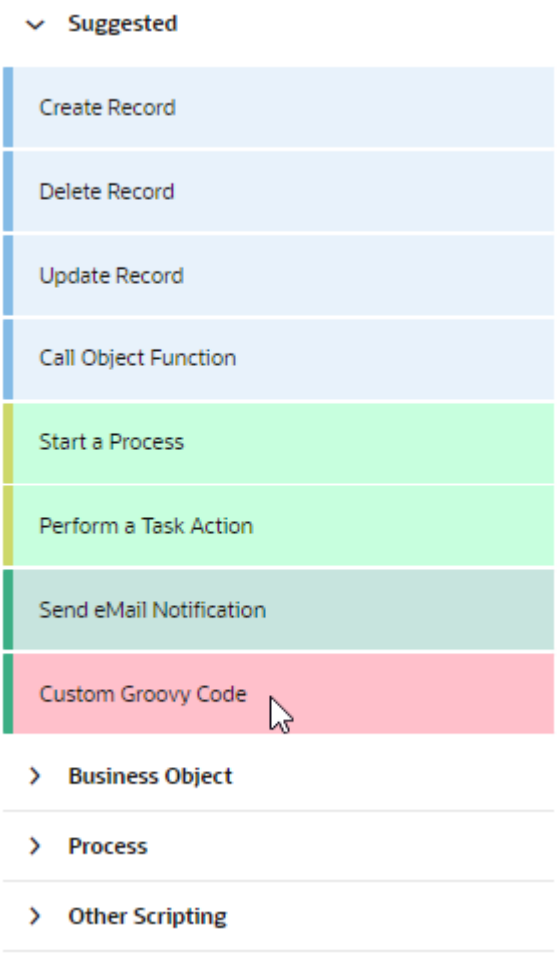
Add an Action to Send Email Notifications

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.

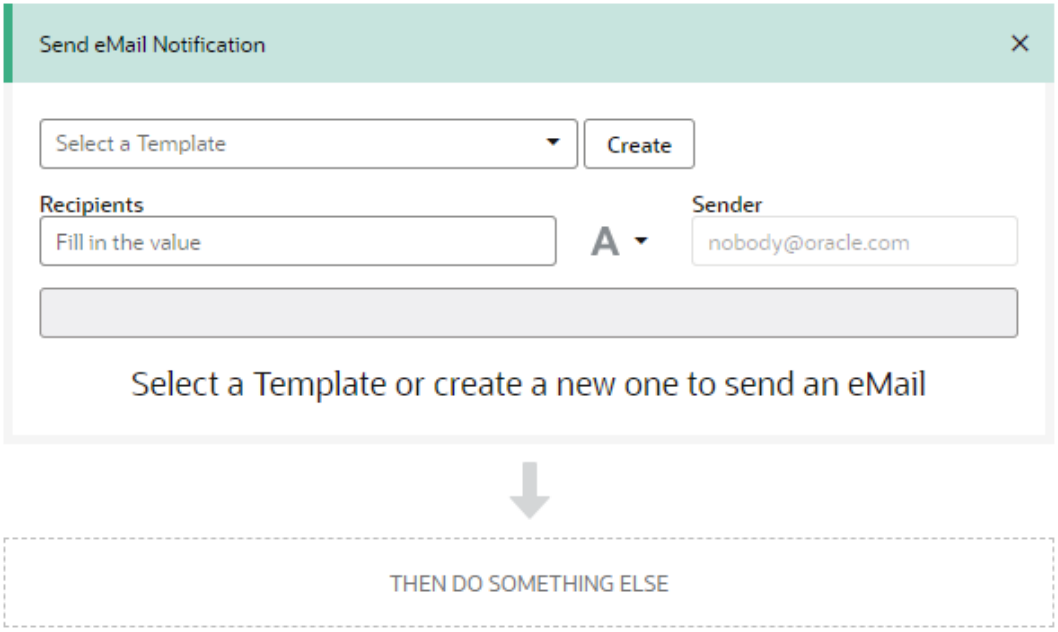
If you add Send eMail Notification to your list of actions, you'll need to specify the email template for the message and the message recipients in the Configure Actions dialog box. You can create your own email template or use an existing one.

To add a Send eMail Notification action to your list of actions:

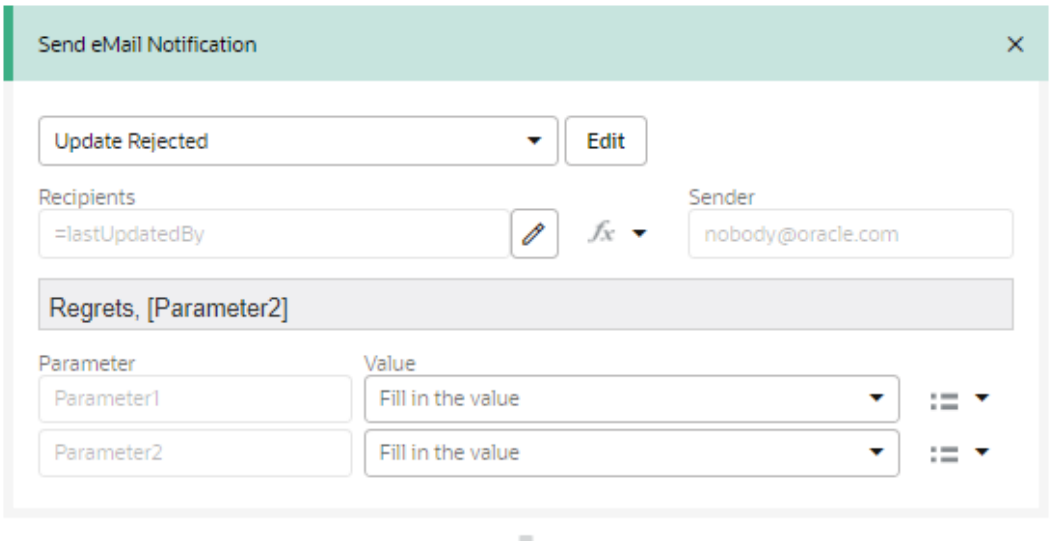
1. For new and existing triggers, click **Add Actions** in the trigger designer to open the Configure Actions dialog box.
   The Send eMail Notification action is available in the list of Suggested actions and under Other Scripting actions.



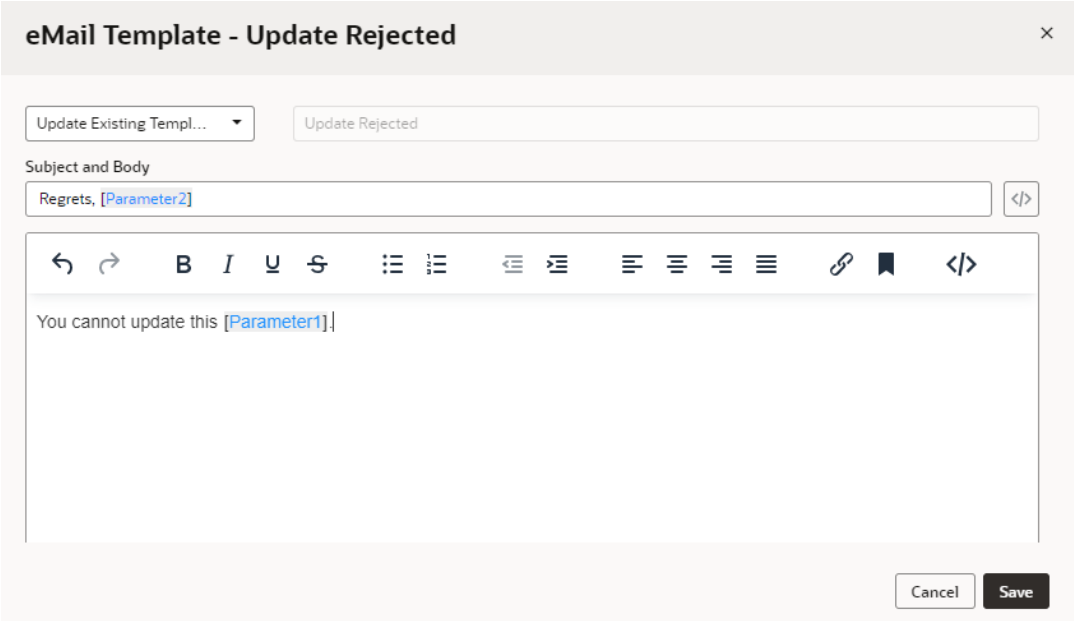
2. Drag the **Send eMail Notification** action into the list.
   After adding the action to the list, select the email template that you want to use, and then specify the email Recipients. Note that the sender's email address is always `nobody@oracle.com` and cannot be changed.



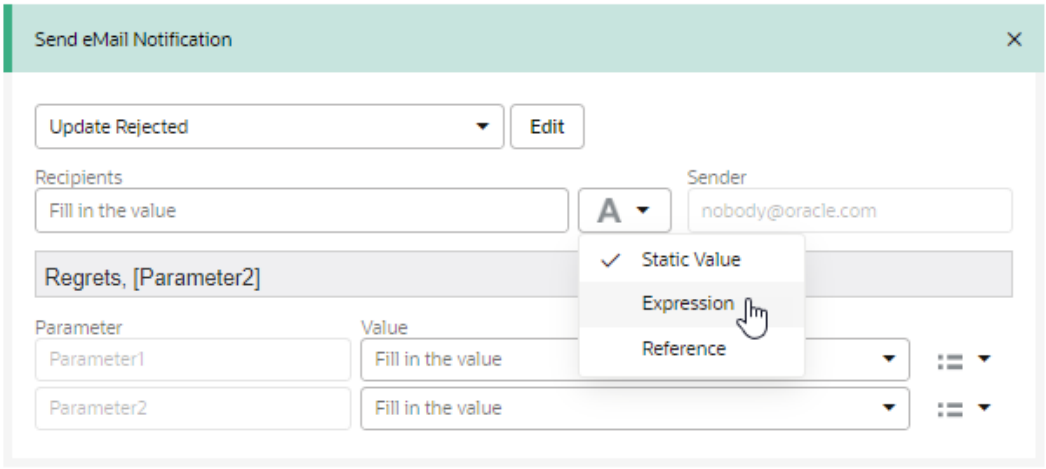
3. Select the email template.
You can select an existing email template from the drop-down list or create a new email template. If you select an existing template, you can click **Edit** to modify the template.



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.



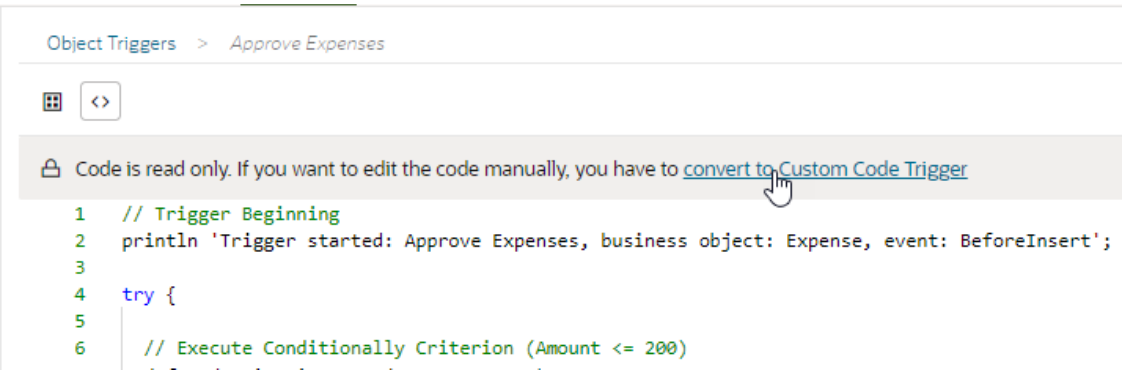
Convert a Trigger to Editable Code

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.

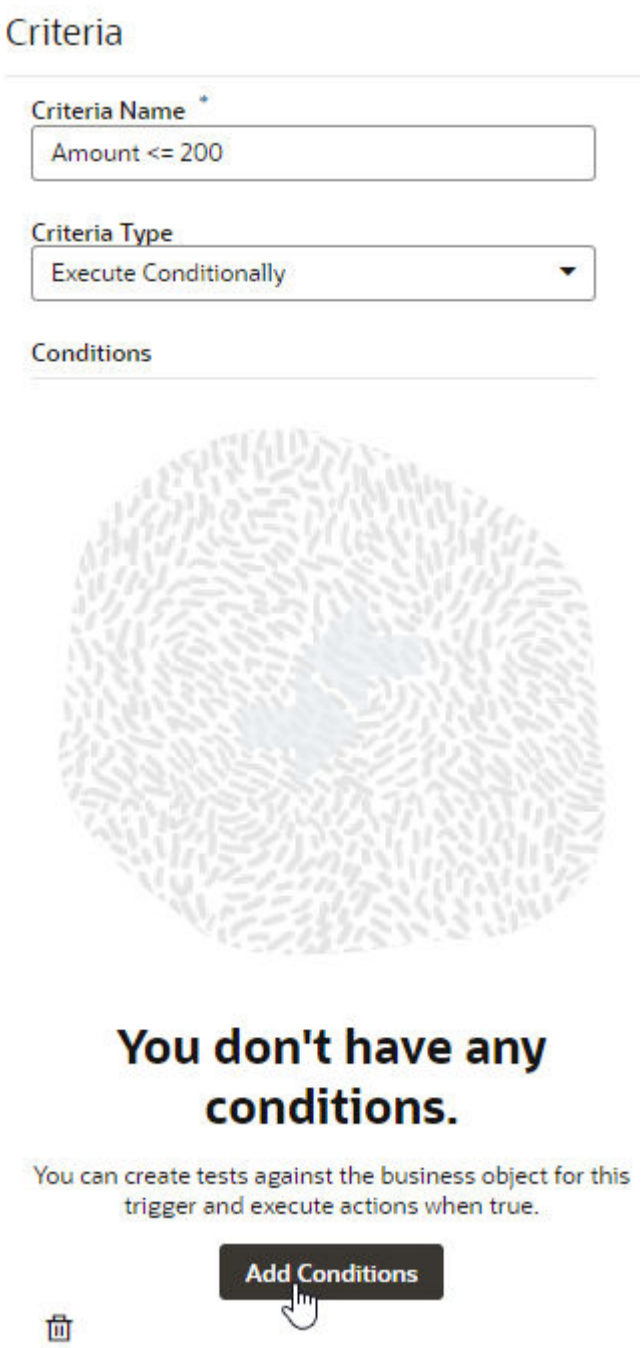
You can view the entire script generated by the trigger designer by clicking <> in the designer. The script displayed in the code editor is read-only. To edit the script in a code editor, click Convert to Custom Code Trigger.



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 Properties pane.



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.

Object Validators for Business Objects

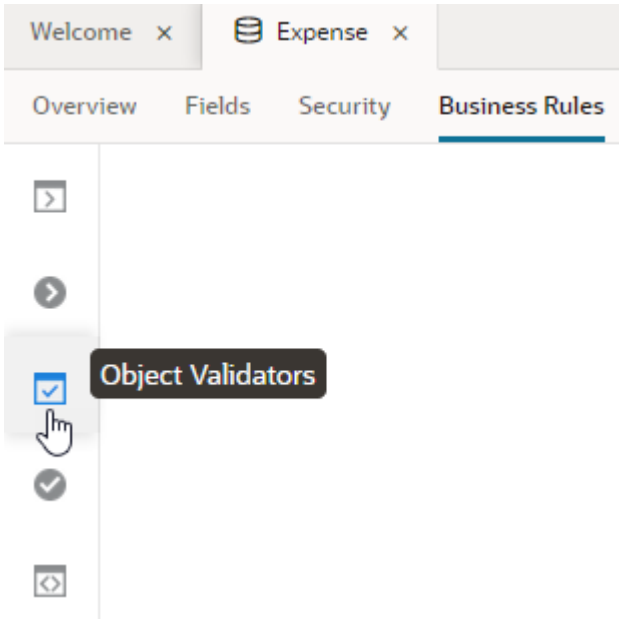
An object-level validation rule is a constraint you can define on any custom object. The rule is used to evaluate the object when attempting to submit an object.

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 optionally override the default Groovy timeout specified in the **Timeout Override** field.

   It's possible to modify all of 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.

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.

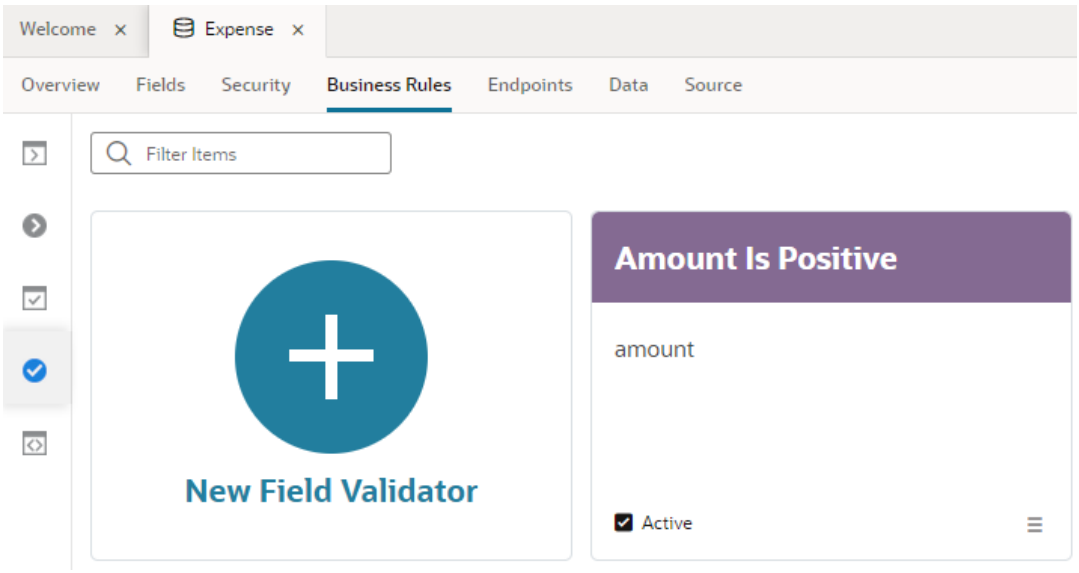
A field-level rule is appropriate when the rule that is to be enforced depends only on the new value being set. At runtime your field validation rule is executed before the field's value is saved.

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:

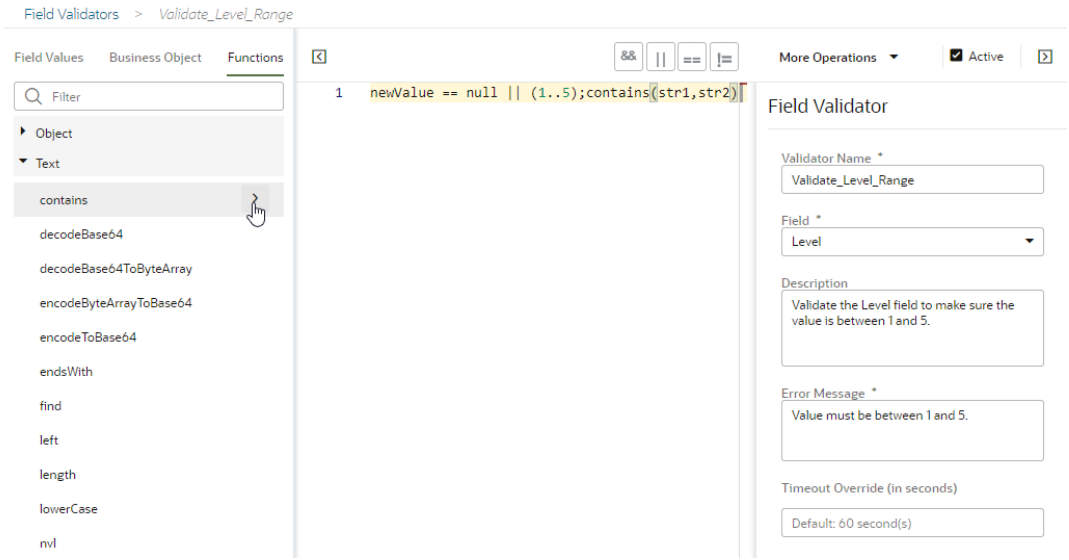
1. Select the business object for which you want to add the new rule.
2. Select the **Business Rules** tab of the business object.
3. Click the **Field Validators** tab.

   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`.

You can optionally override the default Groovy timeout specified in the **Timeout Override** field.

7. Click the **Field Validators** tab again to apply your rule to the field and exit the editor.

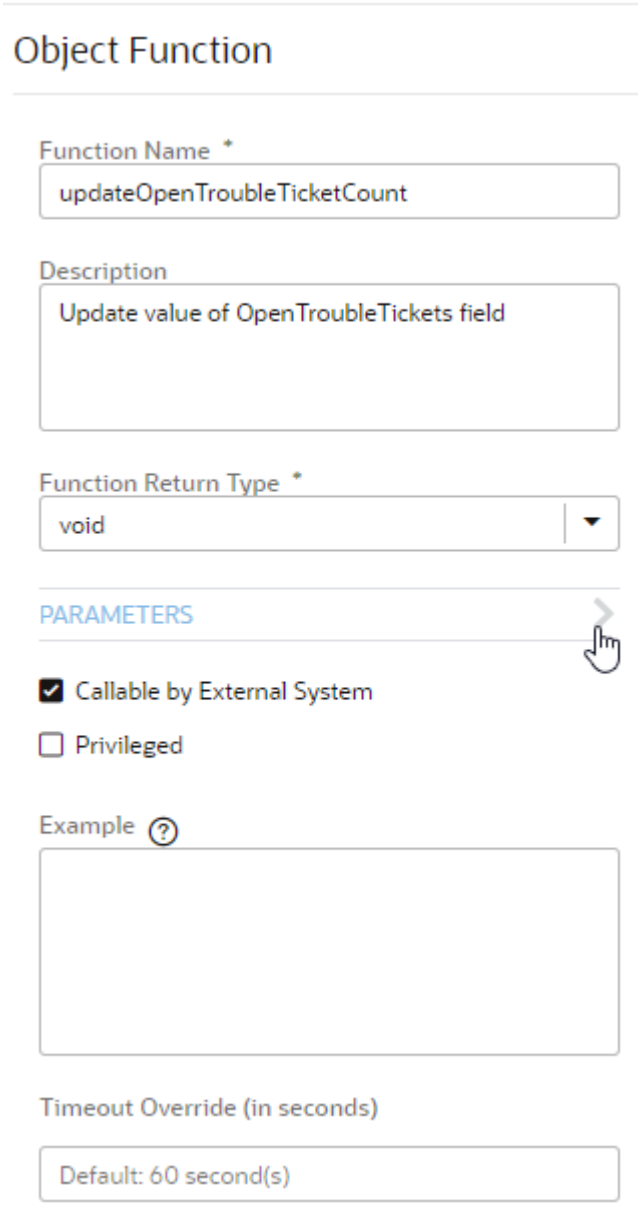
Object Functions for Business Objects

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.

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 object's **Business Rules** tab.
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 enter 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. In the Properties pane, click **Parameters** and add parameters for your function.



When you are done, click **All Properties**.

7. 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.

   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 this default behavior by changing some of the properties in the Properties pane.

- 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 you do not enable this property, then the object function can only be called by some other script on the Department object.

  If you call an object function via REST API from an external system, you'll need to set the value of the Content-Type header to `application/vnd.oracle.adf.action+json`. Note that this Content-type value is not the same as the value used for actions on a business object, for example, a POST action (`application/vnd.oracle.adf.resourceitem+json`).

  When the **Callable by External Systems** property is enabled, you can optionally override the default Groovy timeout specified in the **Timeout Override** field.

- 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.

8. Click the **Object Functions** tab again to add your function and exit the editor.

Log Diagnostic Messages From Your Scripts

When you're developing rules for a business object, you can add a `print` or `println` function to custom Groovy code when you want messages logged by your own script to be written to the diagnostic log.

For example, you might have a Before Update trigger script that calls several object functions. You can add the `print` or `println` function to your object functions to print a message to the log each time the function is called.

- If you're using the code editor, you can add a `print` or `println` statement. The former writes its value without any newline character, while the latter writes its value along with a newline. For example:

  ```
  // Write a diagnostic message to the log
  println("[In: BeforeUpdate] Status = ${Status}")
  ```

- 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.

You can now enable logging to see the functions that were triggered based on the messages, as well as any runtime exceptions. Note that you don't need to do anything special to log trigger starts and trigger ends; these actions are always recorded in the logs.

Work with Endpoints to Access Business Objects

Apps in your visual application and other clients access your business objects through REST endpoints, which are available to you through a business object's Endpoints tab.

To view your business object's endpoints:

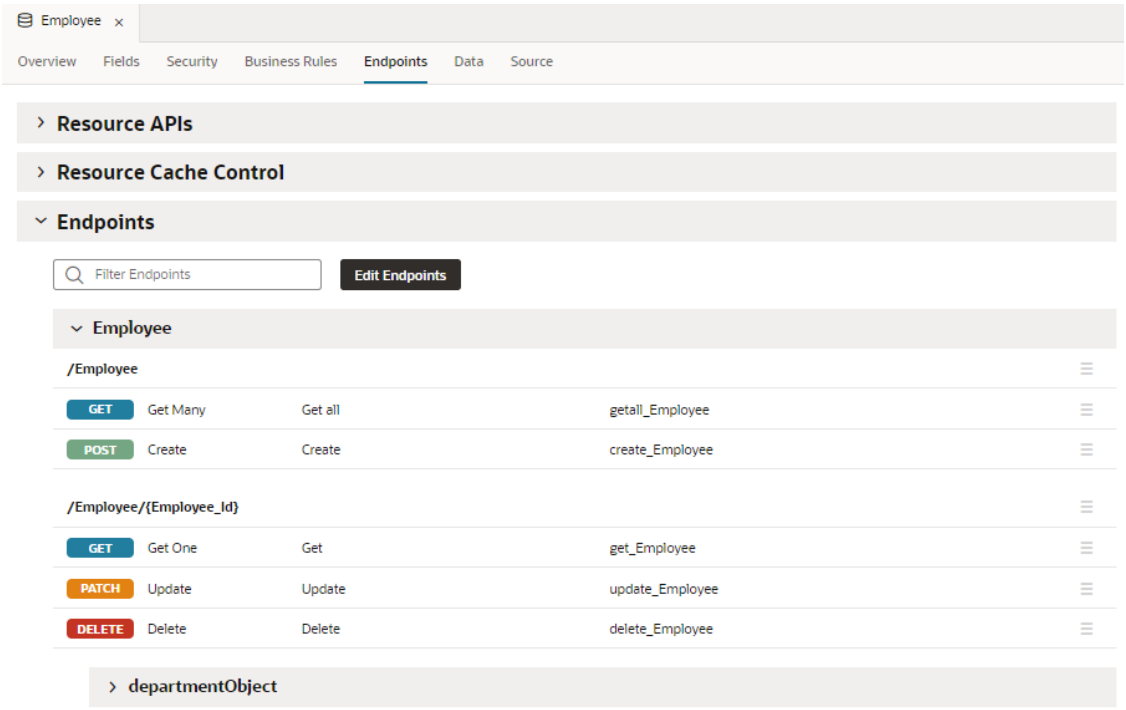
1. Select the business object in the Business Objects tab in the Navigator.
2. Open the **Endpoints** tab for the business object.

If you can't view endpoints, the business object is likely missing a resource file that defines endpoints, or its source file is corrupt (say, because of invalid syntax or merge conflicts in a Git repository).

- If you see a message that there's no resource for a business object, click **Create Resource** to create a *`business-object-name`*`.json` file that defines the default endpoints.
- If you see a message that endpoints cannot be displayed because of errors in the source file, click **Open Source Editor** and resolve the issue.

Access a Business Object's Resource APIs and Endpoints

For each business object, the Endpoints tab displays information about the endpoints you can call in code.



Resource APIs

The **Resource APIs** node displays the URLs you can use to retrieve business object metadata and data in the Development, Staging, and Live databases.

- Use the **Metadata** URLs to retrieve metadata about the business object using its `/describe` endpoint.
- Use the **Data** URLs to modify business object data through CRUD operations.

It's important to use the correct URLs for each phase of your application: **Development**, **Staging**, or **Live**. Development URLs are meant to be used only during your app's development phase and typically take this format:

```
https://{host-name}/ic/builder/design/{app-name}/{app-version}/
```

This URL is a pointer to your app's design-time version (as indicated by `/ic/builder/**design**/` in the path) and is simply a preview of your app. It is *not* meant to allow access to your app from other apps or tools. For example, you might copy the development metadata URL and use it with REST API tools similar to Postman to test the metadata URL during development.

When your app is staged, the URL changes to `https://{host-name}/ic/builder/rt/{app-name}/{app-version}/`, where `/ic/builder/**rt**/` indicates the runtime version of your app. This URL points to your staged application and can be used from other apps or tools.

When your app is published, Visual Builder generates a permanent URL for your application, something like `https://{host-name}/ic/builder/rt/{app-name}/live/`, where **`live`** replaces `{app-version}` to indicate that the app is now live.

Resource Cache Control

The **Resource Cache Control** node displays the caching strategy used to store the business object's data. See Control Data Caching for Business Objects.

Endpoints

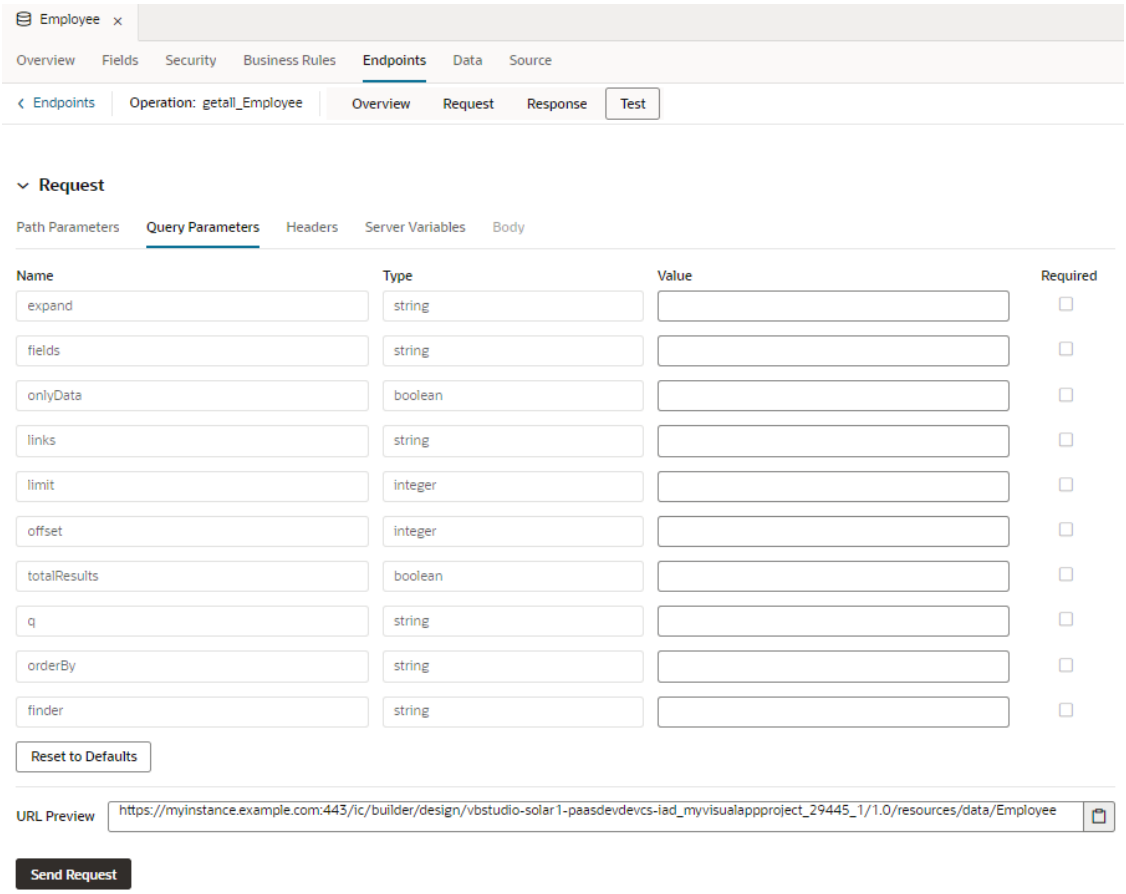
The **Endpoints** node displays the business object's endpoints in tabular format. The display includes the HTTP method, the endpoint URI, an endpoint name that you can use in code, and a description of the endpoint. A filter field at the top of the page allows you to view a subset of the endpoints.

For each 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 that enable you to retrieve, create, delete, and update those business objects are also provided. You can control the endpoints exposed for each business object by adding or removing them in a resource editor (see Add or Remove Exposed Endpoints).

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.



Work With Query Parameters

REST endpoints support static and dynamic query parameters that you can add to the endpoint URL to control what data is returned in endpoint responses. For example, you can use the `q` parameter to build a filter expression that restricts results based on specified criteria, or use `limit` to restrict the number of records returned.

Here's a list of common query parameters that you can use with a GET method:

Query Parameter	Type	Description
`expand`	string	Returns additional information from nested objects in expanded form, for example, `?expand=Employee.JobHistory`.
`fields`	string	Filters data returned based on specified attributes, for example, `?fields=Name,Location;Employee:FirstName,LastName`.
`onlyData`	boolean	Filters data to only return field values and exclude links when `onlyData=true`.
`links`	string	Returns links for a specified object, for example, `?links=linkType1,linkType2`.
`limit`	integer	Restricts the number of records returned, for example, `?limit=200`. To prevent the response from becoming too large, make sure you set the number of records to be fetched appropriately. If the number is too high, you're likely to run into timeout as well as internal server errors caused by a combination of factors.

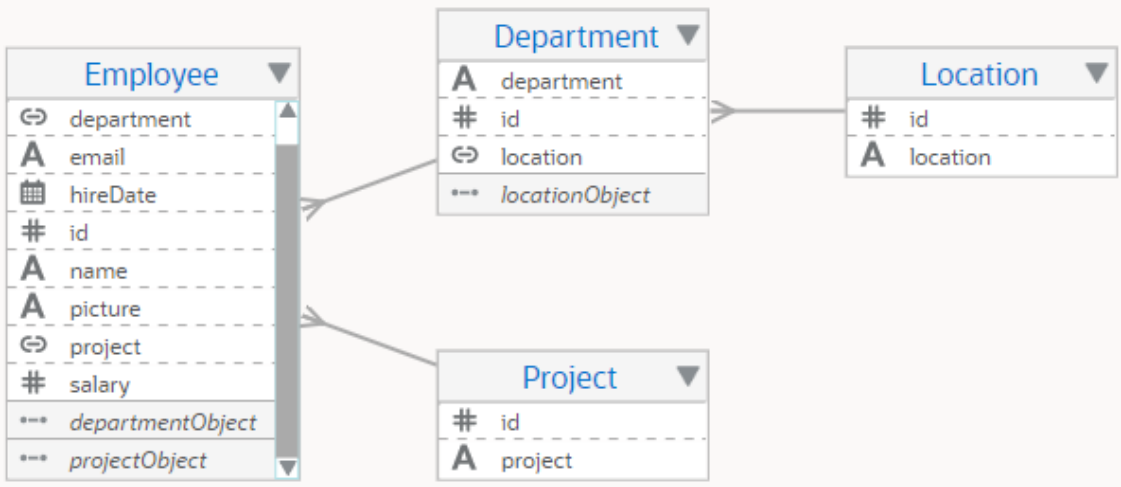
Query Parameter	Type	Description
`offset`	integer	Allows you to page through results by specifying the starting position from which records must be retrieved, for example, `?offset=4`.
`totalResults`	boolean	Returns the total number of records available when `totalResults=true`.
`q`	string	Specifies a filter expression that restricts the items returned based on criteria such as numeric comparisons or string matching with the values of the items' properties, for example, `?q=(Deptno>=10 and <= 30) and (Loc!=NY)`.
`orderBy`	string	Sorts a field's data in either ascending or descending order, for example, `orderBy=salary` or `?orderBy=salary:asc,DepartmentName:desc`.

For detailed examples of query parameters and use cases, see Retrieving Business Objects in *Accessing Business Objects Using REST APIs*. See also the Endpoints reference.

Add or Remove Exposed Endpoints

When your application has many inter-related business objects, you can add or remove endpoints that these business objects expose. Tailoring the endpoints to expose only those that you require controls the size of the metadata file that describes your endpoints and maximizes application performance, both at design time and runtime.

Endpoints for a business object are generated by traversing accessors in a tree structure. By default, only the first level of accessors (those owned by the business object) are added to the object's resource definition. You can change this default definition to add deeper endpoints when related objects are nested several levels down or to remove unwanted endpoints. Let's say your application uses the Employee, Department, Location, and Project business objects, which refer to one other as follows:

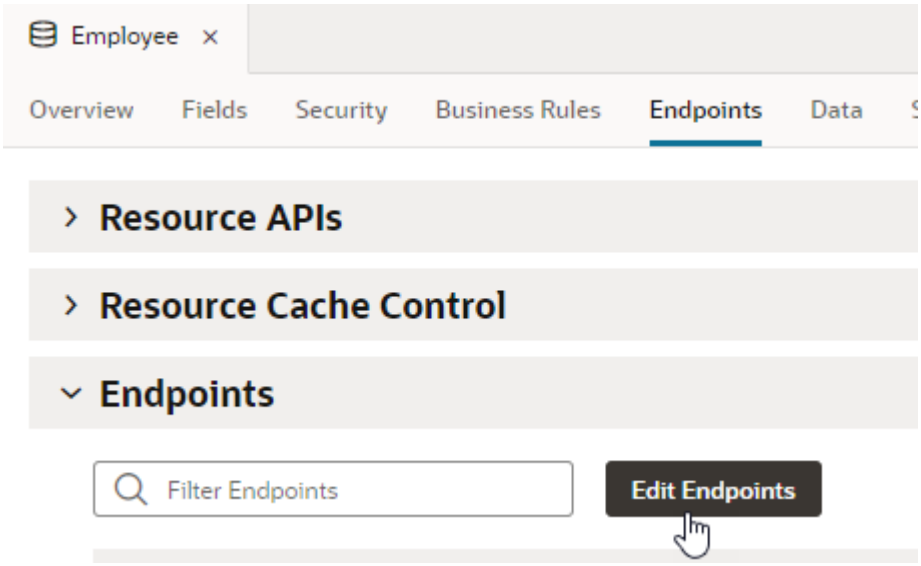


With this relationship, the first level of accessor endpoints for the main Employee object (/departmentObject and /projectObject) are included by default in the Employee object's resource definition. Other deeper endpoints are not included. Now, if you want to show the department and location names in a Table of Employees, you need to add endpoints that are not included by default, but which are required to expose this data. For example, you must add the nested /departmentObject/locationObject endpoint (not selected by default) to show an employee's location.

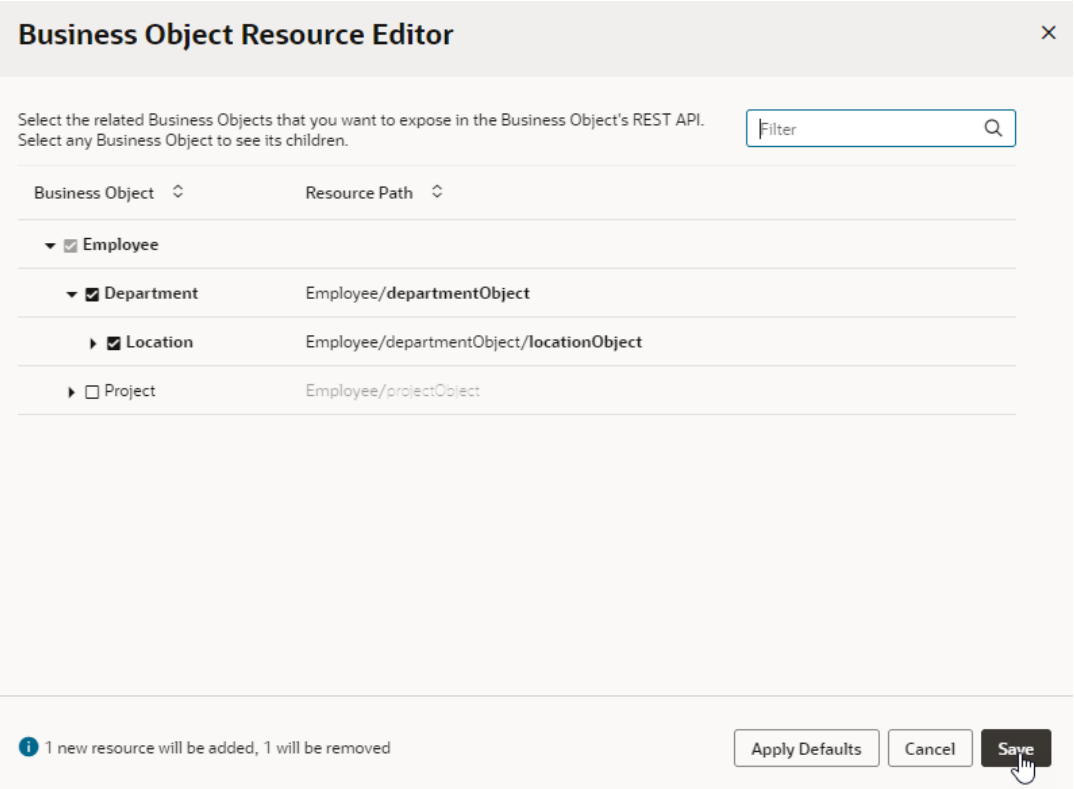
Optionally, if you want to restrict access to employee project information, you can remove the /projectObject endpoint (selected by default).

To select the endpoints you want to expose for a business object:

1. Open the business object's Endpoints tab.
2. In the Endpoints node, click **Edit Endpoints**.



3. In the Business Object Resource Editor, drill down to each related business object to view available endpoints. The endpoints are uniquely identified by their accessors, enabled when one business object references another (see View, Create, and Edit Business Object Relationships).
4. For each related business object, select or deselect endpoints to add or remove them.



By default, only endpoints that are one level down from the parent node are selected. This example adds the deeper Employee/departmentObject/locationObject endpoint and removes the Employee/projectObject endpoint.

5. Click **Save**. To take the default setting, click **Apply Defaults**.

View and Edit Data in Business Objects

You can use the Data tab to view the data associated with your business objects in any of your databases (Development, Staging, and Live).

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 uses columns to display the data stored in the business object's fields. The table displays the data for editable and read-only fields.

2. To sort data in a column, click the column's header arrows, or expand the **Query** node and enter search criteria to display only the rows that meet your criteria.

Edit the Data in Business Objects

You can directly edit business object data stored in your databases, by using a business object's Data tab to modify, export, and import the data.

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:

- For business objects marked as containing application setup data, reload setup data from application sources

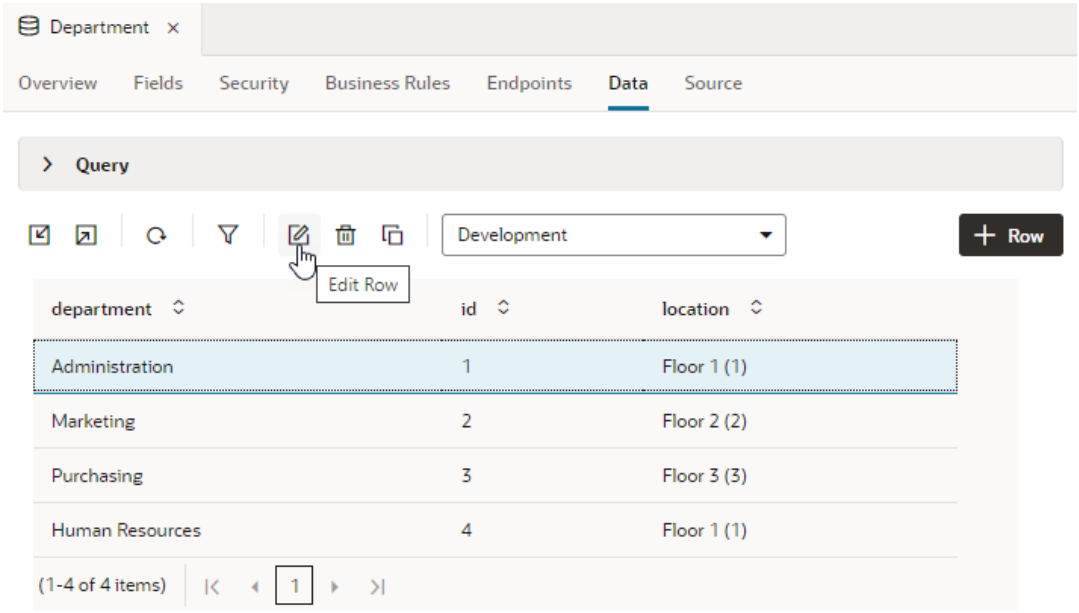
- Import a CSV file or Excel spreadsheet (`.xls` or `.xlsx`) to replace or append data
- Export the entire table as a CSV file
- Refresh the data display
- Select the fields to be displayed (by default, only the **Id** field and user-created fields are displayed)
- Edit the data stored in the editable fields of an object by editing individual rows
- Delete a row
- Duplicate a row

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 business object's fields.

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.

Import Data to a Business Object

You can edit the data stored in a business object by importing a CSV file or an Excel spreadsheet (XLS or XLSX). When you import the file, you can choose to replace all the current data or add the data in the file as new rows to your existing data.

Use this import option to update the data in a particular business object. To simultaneously update the data for one or more business objects, you can import data from the Data Manager (see Work with the Data Manager).

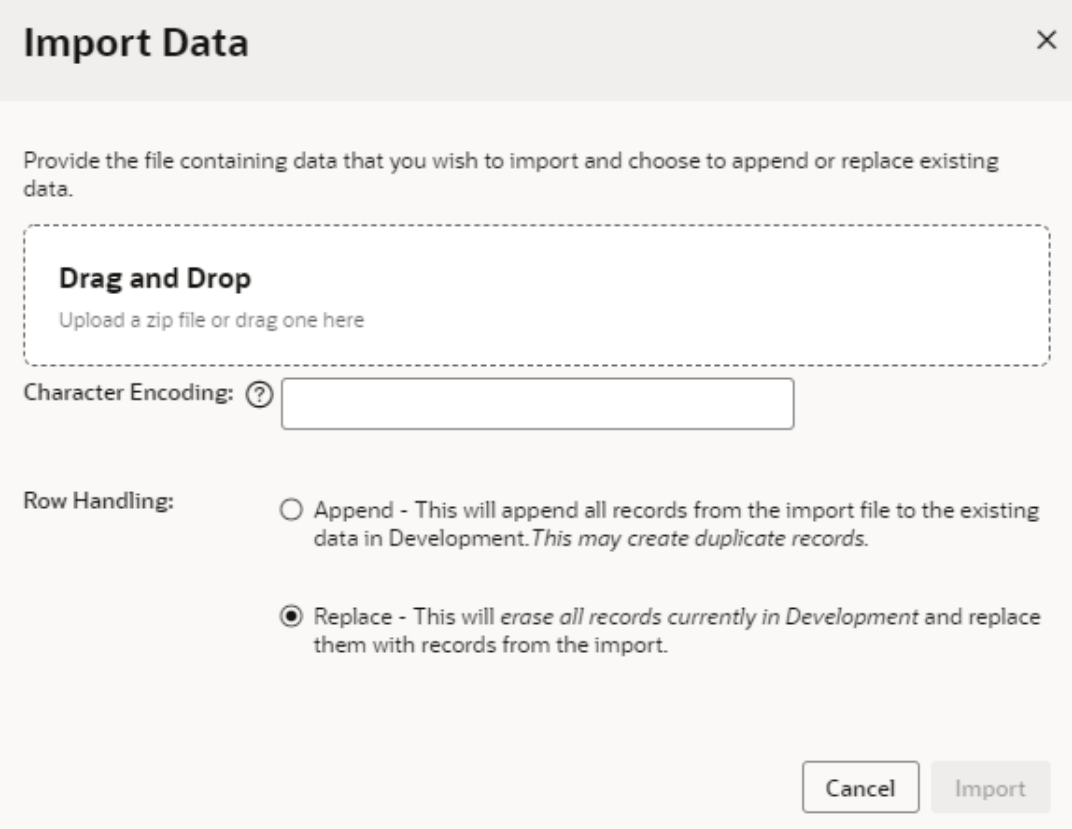
To import data to a business object:

1. Open the **Data** tab of the 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**.

Reload Data from Application Sources

For business objects marked as application setup data, if your development database schema does not show the most up-to-date data (say, because the object's data was updated as part of a Git merge), you'll need to reload data from the application's sources.

When a business object is identified as containing application setup data, its data is considered application metadata and is stored in the `entity-data.csv` file as part of the application's sources. Any time you update the object's data, either by editing it directly in the Data tab or by importing files through the Data Manager, your updates take effect in the `entity-data.csv` file and reflect automatically in your development database schema. Sometimes though, if this data is updated as part of a Git merge, your database might not show those updates. This can

happen when two users update data on their own Git branches and then merge those updates to the default branch (`main`, for example), or even when you switch branches in your Git repository. You'll then need to get these updates by reloading the `entity-data.csv` file from the application's sources to your database schema.
To reload data from the application's sources:

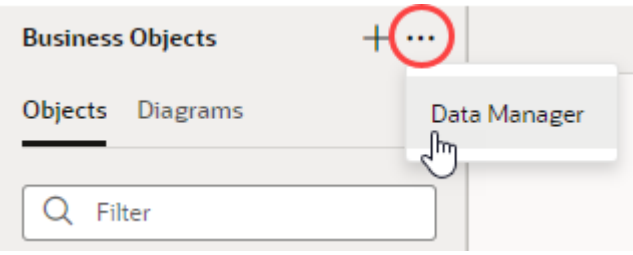
1. Select the business object marked as containing application setup data.
2. Click **Data**.
3. Click the option to reload setup data from application sources ( ).
4. After data is imported from the `entity-data.csv` file, click **Close**.

Your development database schema reflects the current version of the `entity-data.csv` file.

Work with the Data Manager

Visual Builder provides tools such as the Data Manager to help you manage the data stored in business objects while developing your visual applications.

To open the Data Manager, click **Menu** in the Navigator's Business Objects pane and select **Data Manager**.



Manage Data During the Development Lifecycle

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 to a File from the Data Manager and Import Data From a File Using the Data Manager.

> **Note:**
>
> If you want to access data in an existing database, see how you can bring your own schema when working with business objects.

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.

Phase	Description
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
Staging	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
Live	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

Import Data From a File Using the Data Manager

You can replace the data in one or more of your business objects by importing CSV files and Excel spreadsheets. Use the Import from File option 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 need one CSV file for each of the business objects that you want to update. 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 option to replace the data in a business object, all its data 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 some default fields—`id`, `creationDate`, `lastUpdateDate`, `createdBy`, `lastUpdatedBy`, and `versionNumber`—which are automatically created for you and assigned values. If you include `id` in your file (as shown here), those values are used:

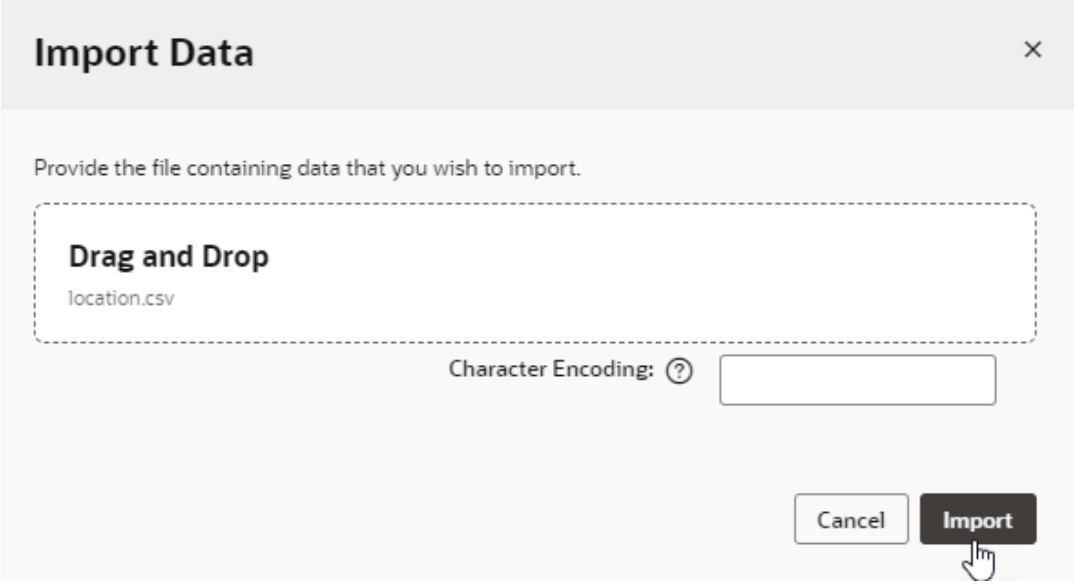
	A	B	C	D	E	F	G	H	I
1	Id	Name	Email	Hire Date	Country	Salary	Picture	Department	
2	1	Steven King	sking@examp	2003-06-17	US	5000	https://w	50	
3	2	Neena Kochhar	nkochhar@exa	2005-09-21	IN	4000	https://w	50	
4	3	Lex De Haan	ldehaan@exar	2001-01-13	DE	2000	https://w	50	
5	4	Alexander Hunold	ahunold@exa	2006-01-03	CZ	2000	https://w	50	
6	5	Bruce Ernst	bernst@exam	2007-05-21	US	3000	https://w	50	
7	6	David Austin	daustin@exan	2005-06-25	CA	3500	https://w	50	
8	7	Valli Pataballa	vpatabal@exa	2006-02-05	IN	3500	https://w	60	
9	8	Diana Lorentz	dlorentz@exa	2007-02-07	CA	4300	https://w	40	
10	9	Nancy Greenberg	ngreenbe@ex	2002-08-17	US	2000	https://w	60	

> **Note:**
>
> 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 (like `6/17/2006`). 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 States). Alternatively, edit the file in a text editor. You can't import a column that contains dates formatted `dd/MM/yy HH:mm`.

The import option does not create or remove fields for business objects. You can use the Business Objects editor to create or remove fields and to edit data. Note also that the import option replaces existing data in your database. If you want to append data to a business object, you can import a file using the import option in the Business Objects editor.

To import data into a database:

1. Click **Menu** in the Business Objects pane and select **Data Manager**.
2. Select the database that you want to update.
3. Click **Import from File**.
4. Drag the file into the drop area in the Import Data dialog. Alternatively, click and locate the file on your local system.



5. Click **Import**.

If the import is successful, you'll see a success message. If there are problems importing the data, you'll 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. Try to resolve the problem by comparing the fields in the schema to the columns in the CSV or spreadsheet, then either modify the schema in the Fields tab or modify the data in the file. Import the file again to correct the data.

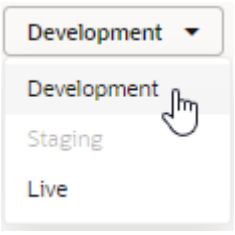
Import Data From a Database

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 **Menu** in the Business Objects pane and select **Data Manager**.
2. Select the database that you want to update in the drop-down list at the top of the page.



Unavailable databases are grayed out. After you select the database, the page will display the import and export tools for the database.

3. Click the tile to import data into your database (for example, **Import from Live**).

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 confirms that the data was successfully imported or warns you if there are problems.

Export the Data to a File from the Data Manager

You can export all the data contained in your database as CSV files. The export tool creates one CSV file for each of the business objects in your database and packages the files as a ZIP archive.

To export the database data as a CSV file:

1. Click **Menu** in the Business Objects pane 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 business object from the object's Data tab.

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. For files with string type fields, the data in each string field must not exceed 4000 bytes.

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.
4. Make sure that the MIME type of the file is compatible with the browser you're using. The following MIME type is accepted by most browsers:

```
[[["application/vnd.ms-excel","text/csv"]]]
```

Import and Export Data From the Command Line

You can perform bulk import and export of data from the command line using Visual Builder APIs.

The 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 daily `cron` job to import and export data and synchronize the data in your business objects with data in another table.

Import Data from the Command Line

To import data into a business object from the command line, you use a POST method with the `/resources/datamgr/import` endpoint.

You will need to upload `.csv`, `.xls`, or `.xlsx` files containing the data. The file name 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.

Query Parameter

The `import` endpoint only accepts the `append` parameter:

Parameter	Description
`append`	Boolean, which if `true` adds the rows in your file as new rows to the business object. The default value is `false`, which results in your file's data replacing existing data in the business object.

Syntax

To import data for all business objects in an application, you would use a POST method as follows:

```
POST https://host:port/ic/builder/design/appName/appVersion/resources/datamgr/
import
```

You can find your app's details on the Settings editor: look for *appName* in Application Name, and for *appVersion*, use the number that follows the dash in Application ID.

For example, to import data into all business objects for version `1.0` of the `MyApp` application, use:

```
POST https://myserver:myport/ic/builder/design/MyApp/1.0/resources/datamgr/
import
```

Remember to upload the data file you want to import and indicate its content type in the request body. If you were using REST API tools similar to Postman, for example, you would use the **Body** tab to upload the data file (say, `MyObject.zip`) and select **binary** as the content type.

To import data for a specific business object, you would use a POST method as follows:

```
POST https://host:port/ic/builder/design/appName/appVersion/resources/datamgr/
import/objectId
```

For example, to import data into the `MyObject` business object in version `1.0` of the `MyApp` application, use:

```
POST https://myserver:myport/ic/builder/design/MyApp/1.0/resources/datamgr/
import/MyObject
```

> **Note:**
>
> To import data to the *staging* or *live* database, replace `/design/` with `/deployment/` in the endpoint path.
> If your application is live, use the options menu to lock the application before you import data. Unlock the application after you finish.

See also Data Management REST Endpoints in *REST API for Oracle Visual Builder*.

cURL Commands

If you were using the cURL command-line tool to import *development* data for a business object named `MyObject` in version `1.0` of the `MyApp` application, you would use a POST method as follows:

```
curl -X POST -u user:password https://host:port/ic/builder/design/MyApp/1.0/
resources/datamgr/import/MyObject -H Content-Type:text/csv -T MyObject.csv -v
```

To import development data for multiple business objects in version `1.0` of the `MyApp` application, you would use a POST method as follows:

```
curl -X POST -u user:password https://host:port/ic/builder/design/MyApp/1.0/
resources/datamgr/import  -H Content-Type:application/zip -T myObject.zip -v
```

If you were using the cURL command-line tool to import data to the *staging or live* database for a business object named `MyObject` in version 1.0 of the `MyApp` application, you would use a POST method as follows:

```
curl -X POST -u user:password https://host:port/ic/builder/deployment/
MyApp/1.0/resources/datamgr/import/MyObject -H Content-Type:text/csv -T
MyObject.csv -v
```

Export Data from the Command Line

To export a business object's data from the command line, you use a GET method with the `/resources/datamgr/export` endpoint.

Syntax

To export development data from all business objects in an application, you would use a GET method as follows:

```
GET https://host:port/ic/builder/design/appName/appVersion/resources/datamgr/export
```

You can find your app's details on the Settings editor: look for *appName* in Application Name, and for *appVersion*, use the number that follows the dash in Application ID.

For example, to export development data from business objects in version `1.0` of the `MyApp` application, use:

```
GET https://myhost:myport/ic/builder/design/MyApp/1.0/resources/datamgr/export
```

Exporting the application's data results in a ZIP archive that contains a `.csv` file for each business object in the application.

To export development data for a specific business object, you would use a GET method as follows:

```
GET https://host:port/ic/builder/design/appName/appVersion/resources/datamgr/export/objectId
```

For example, to export development data from the `MyObject` business object in version `1.0` of the `MyApp` application, use:

```
GET https://myhost:myport/ic/builder/design/MyApp/1.0/resources/datamgr/export/myObject
```

Exporting the data in a single business object results is a `.csv` file containing the data in the specified business object.

> **Note:**
>
> To export data to the *staging* or *live* database, replace `/design/` with `/deployment/` in the endpoint path.
> If your application is live, use the options menu to lock the application before you export data. Unlock the application after you finish.

See also Data Management REST Endpoints in *REST API for Oracle Visual Builder*.

cURL Commands

If you were using the cURL command-line tool to export *development* data for version `1.0` of the `MyApp` application to a ZIP archive named `myapp.zip`, you would use a GET method as follows:

```
curl -u user:password https://host:port/ic/builder/design/MyApp/1.0/resources/
datamgr/export > myapp.zip
```

This command creates a `.zip` file, `myapp.zip`, which will contain a `.csv` file for each business object in `MyApp`.

If you were using the cURL command-line tool to export *staging or live* data for version `1.0` of the `MyApp` application to a ZIP archive named `myapp.zip`, you would use a GET method as follows:

```
curl -u user:password https://host:port/ic/builder/deployment/MyApp/1.0/
resources/datamgr/export > myapp.zip
```

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.

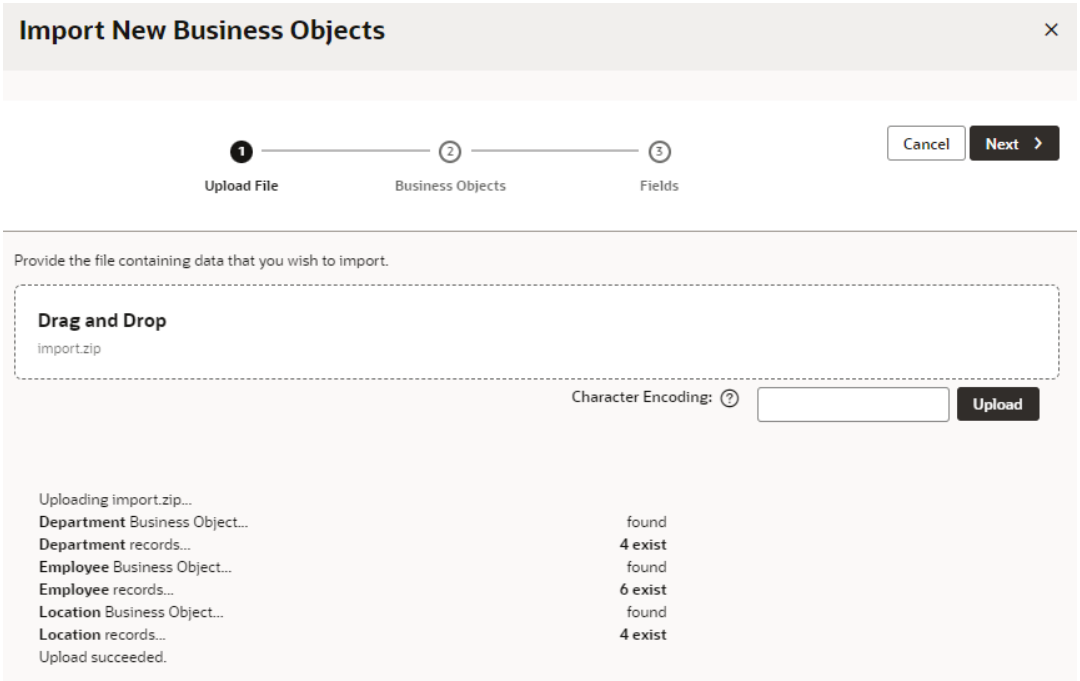
To create business objects in the Import New Business Objects wizard:

1. Click **Menu** in the Business Objects pane and select **Data Manager**.
2. On the Data Manager page, click **Import Business Objects** to open the Import New Business Objects wizard.
3. 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.

> **Note:**
>
> It is recommended that you keep the size of the file you upload small, representative of your entire data set and only what's required to create your business object model. Once your business object model is created, you can always import all your data, even large datasets, as described in Import Data From a Database.

The default character encoding format for an imported file is `UTF-8`. If a value isn't specified in the Character Encoding field or if the specified value is invalid, the default is used. See `https://www.iana.org/assignments/character-sets/character-sets.xhtml` for a list of encoding formats you can use.

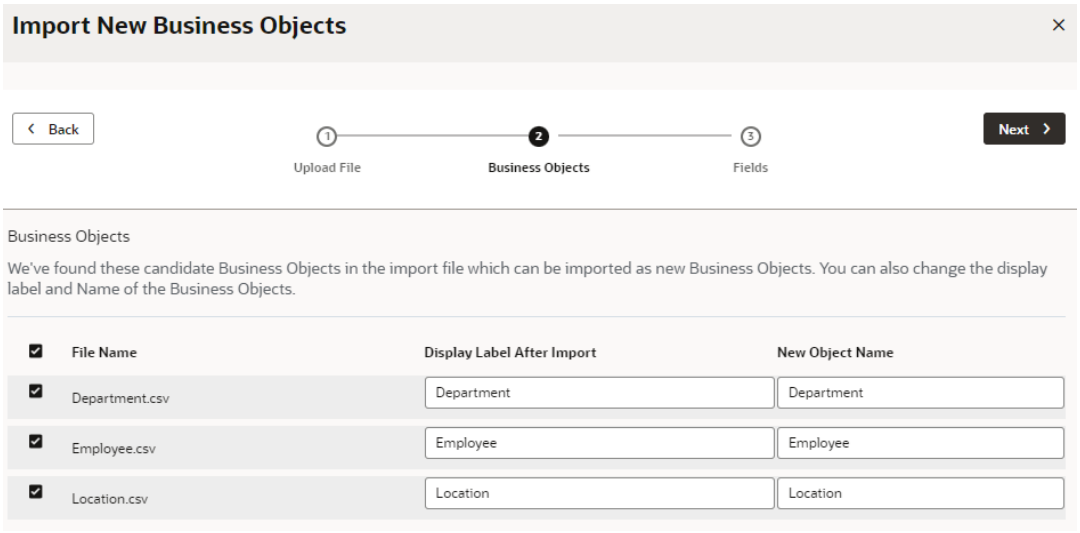
Click **Upload**. After the upload is complete, the wizard displays a list of the business objects and records found in the upload.



Click **Next**.

4. In the Business Objects step of the wizard, select the business objects that you want to create.

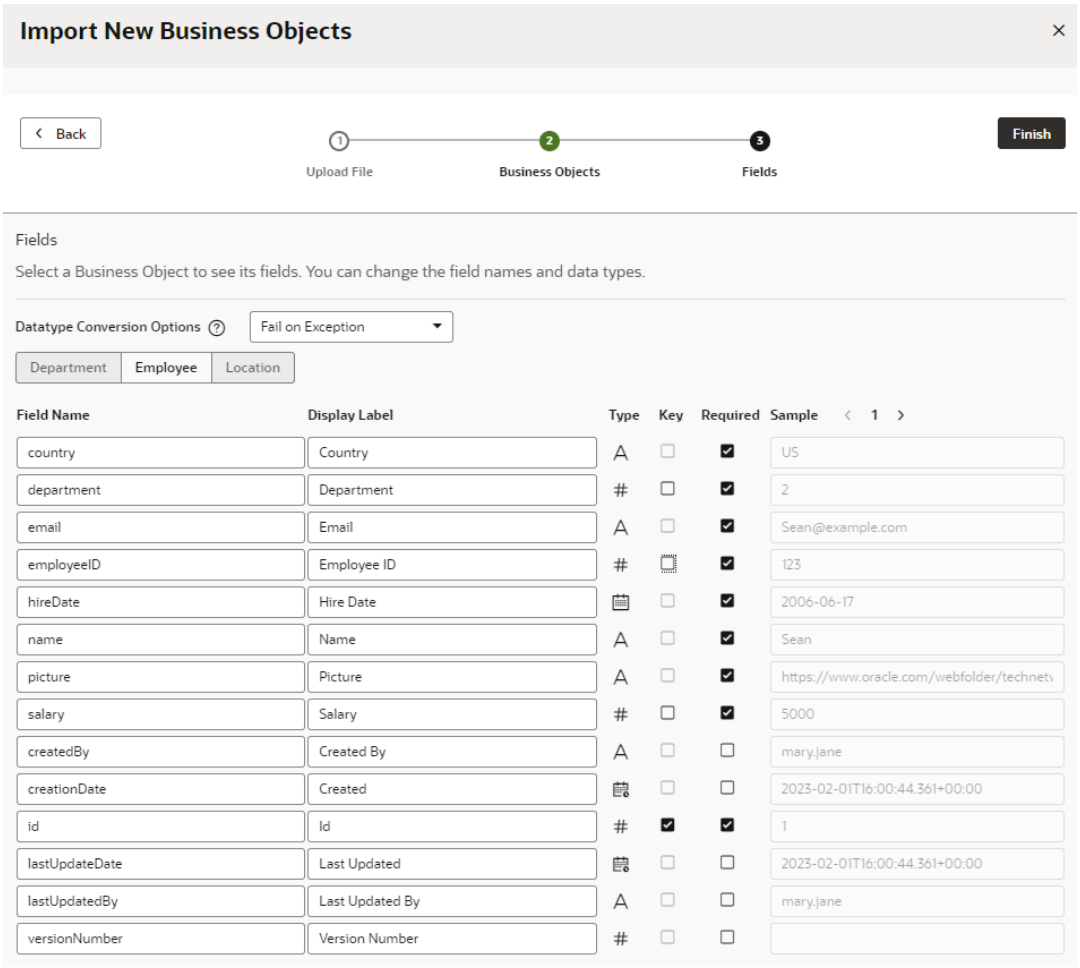
   The wizard displays a list of the business objects that can be created and the files in your upload that they are based on. You can select which business objects you want to create, and edit the display labels and names of the new business objects.



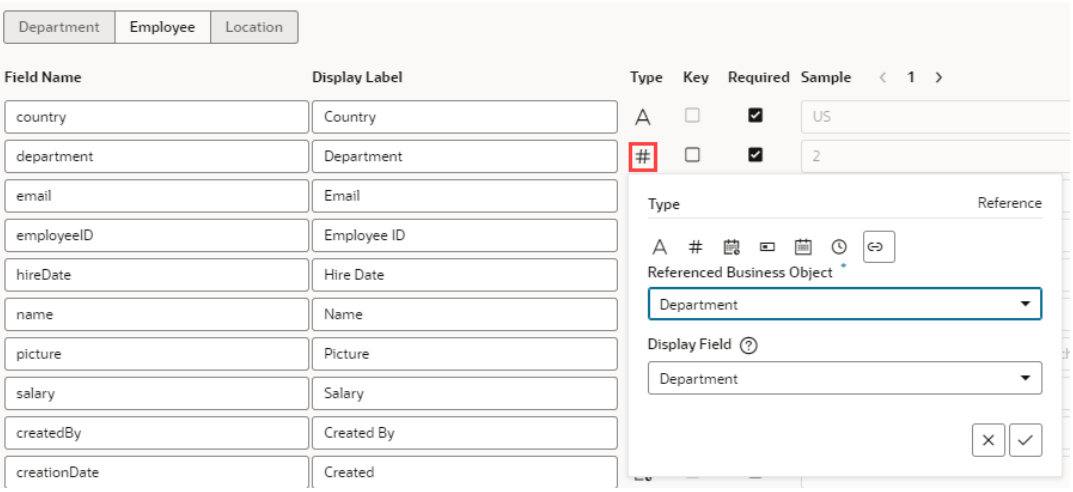
Click **Next**.

5. 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. You can edit each field name, display label, its data type, and specify whether it is required.

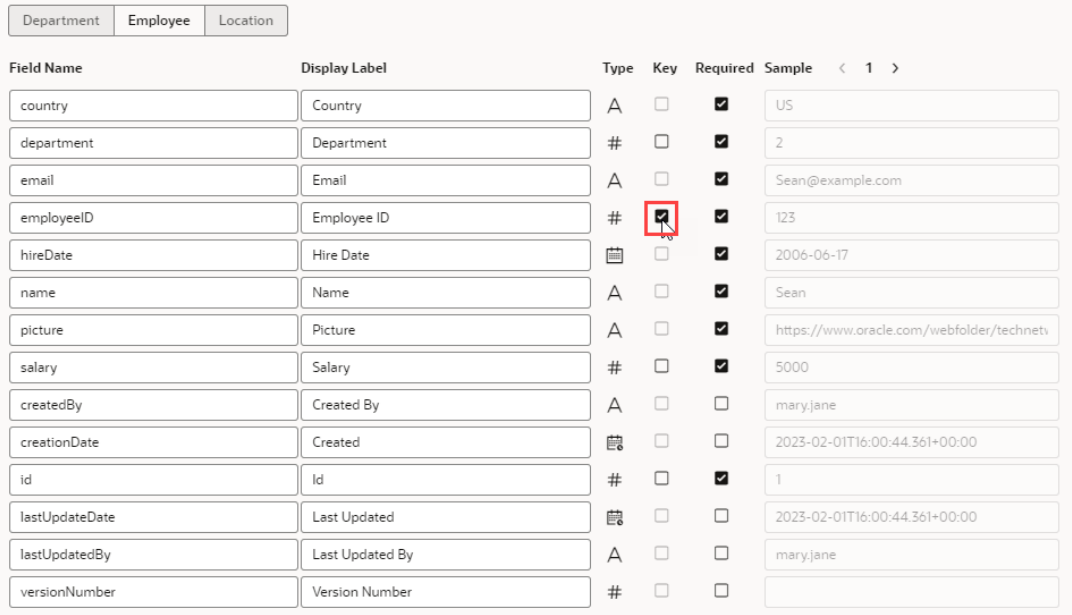


To change the data type, click the Type icon for each field to open a pop-up box where you can modify the type. For Reference fields, you can select the related business object from an existing business object or from those that you are importing, and select the field in the related object to display.



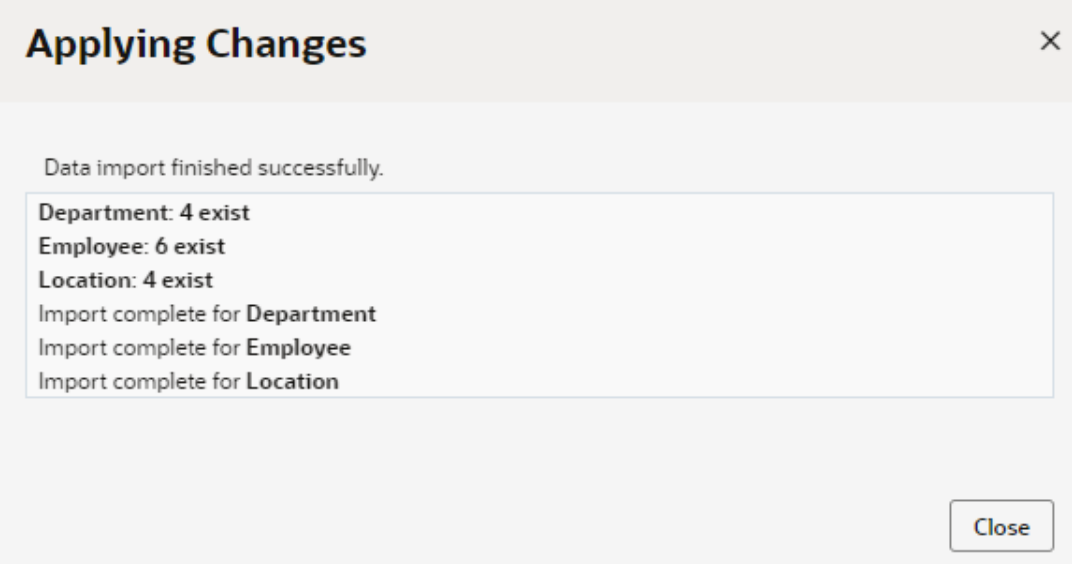
If your business object's definition contains a field to uniquely identify each record (the primary key field), you can use that field as the key instead of the system-generated `id`

field. To change the primary key field, select the Key icon for the field you want to use. This option is enabled only for non-blank data containing numeric fields. Once you switch, the default `id` field will no longer be selected; you can then safely delete this field after the business object is created.



Click **Finish**.

The wizard displays a list of the new business objects that were successfully imported.



Set Your Own Audit Fields For Imported Business Objects

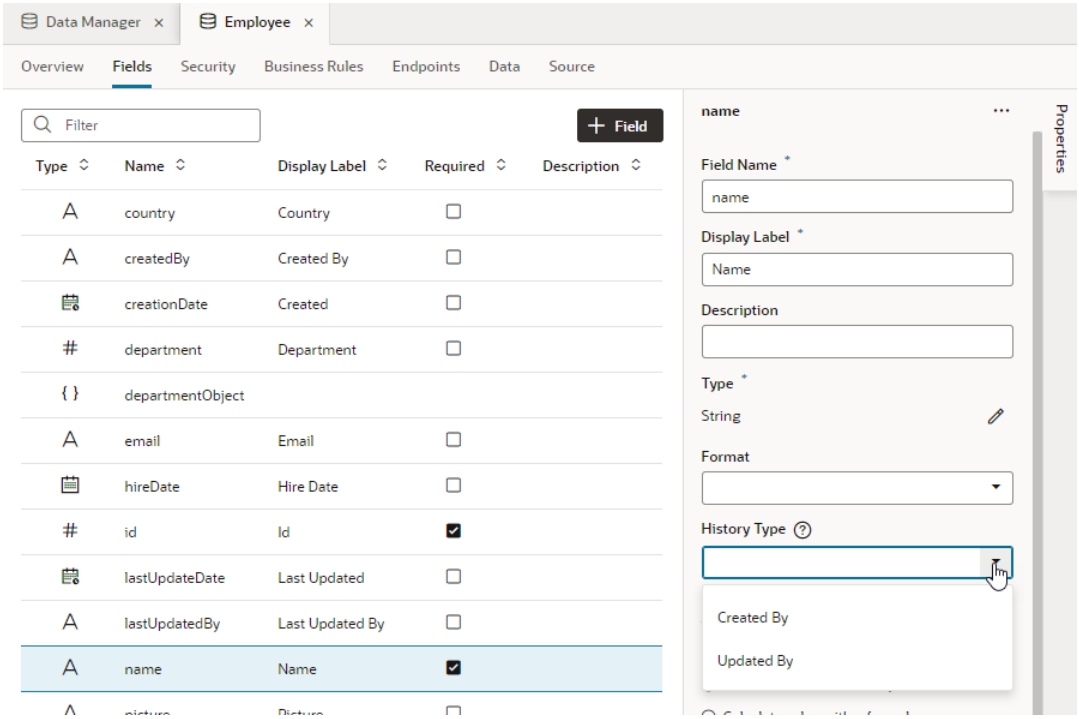
When you create a business object, including those imported from a file, fields that audit your business object's history are automatically created for you. If your file contains audit

information, you can use that data to track your object's history, instead of the default audit fields.

Audit fields maintain a history of changes made to your business object by tracking who created and updated an object when. They also include a version number used in the generation of an entity tag (ETag) which protects against users overwriting changes. Because audits can help you verify changes, you might want to set audit fields for each of these history types: `Created By`, `Updated By`, `Time Created`, `Time Updated`, and `Version Number`. A warning will be logged in your app's Audits pane for each history type that isn't assigned to a field. You'll also see an error if the same history type is assigned to more than one field.

To set a field for auditing:

1. Open your business object's **Fields** editor and select the field you want to use for auditing.
2. In the field's Properties pane, select an option in the **History Type** list.



Your options depend on the field's data type. For a string-type field, you see **Created By** and **Updated By** (as shown here). For a datetime-type field, you see **Time Created** and **Time Updated**, and **Version Number** for a number field.

Optimize Business Object Performance

When working with business objects, you might want to tune settings for your business object's fields and functions to optimize performance and enhance your application's user experience.

Override Default Timeout for Groovy Scripts

When you use Groovy code (whether short expressions or multi-line scripts) for business logic, you have the option of overriding the default timeout configured for Groovy processing. You

might want to do this if you expect your script's processing time to exceed the default Groovy timeout at runtime.

You can specify your own timeout for Groovy code that calculates a formula field's value or a field's default value. You can also change this setting in business rules when you define field-level or object-level validation rules, field or object triggers, and object functions.

The default timeout for each of these options varies and should work for most scenarios. Consider changing the default timeout only if your script's processing time will overrun the default duration and you run the risk of the connection timing out before your business logic can be executed. If you anticipate a fairly long processing time, you might also want to enable polling to prevent endpoint requests from being terminated because of (say) gateway or browser timeouts.

This table describes the default timeouts predefined for Groovy. To override the default timeout, you use the Timeout Override property, either in a field's Properties pane or in a business rule's editor.

Business Object Functionality	Default Timeout (in seconds)	To Change the Default Timeout, See:
Calculate a field's default value when using an expression	60 seconds	Set a Default Value for a Field
Calculate a field's value using a formula	5 seconds	Add a Formula to a Field
Object and field triggers	60 seconds	Object Triggers Field Triggers
Object and field validators	60 seconds	Object Validators for Business Objects Field Validators for Business Objects
Object function	60 seconds	Object Functions for Business Objects

Enable Polling for Endpoint Requests

If you run into timeout issues when working with business objects, you might want to enable polling for long-running endpoint requests.

Polling is useful in many contexts involving long-running processes, where you run the risk of breaking the client/server connection because of gateway or browser timeouts. A process can be long running, say, when your application integrates with external services, perhaps through a trigger that makes API calls to an external service. It can also involve endpoint requests that import a large volume of data from a file or from one database (development, staging, or live) to another during your application's lifecycle. Most data-related endpoint requests, including those to create, query, update, and delete business object data, are long-running processes that can benefit from polling.

You can enable polling by adding the `vb-poll=true` query parameter to an endpoint request URL. Now when the client makes an endpoint request, the server—instead of waiting for the request to complete and then return an HTTP response (status 200 or otherwise)—returns an HTTP response (status 202) with details of a new URL for the client to poll. This allows the server to continue processing the request in the background and the client to poll the new URL as and when it wants to find out if the request is complete and get the response (or error).

To enable polling for long-running endpoint requests:

1. Add the `vb-poll=true` query parameter to your endpoint request URL, for example:

```
POST https://server.example.com/ic/builder/rt/hrapp/1.0/resources/
data/Employee?vb-poll=true
```

2. When you receive the HTTP response with status 202, look for the `Polling-Location` header whose value will be the polling URL.

   The client can poll this URL and check the response, which will either contain the `Polling-Location` header to indicate that the process is still running, or will be the final response.

   When the long-running process completes, the response remains available for a limited time period, after which it is removed. The process itself is not affected by this, but the result is not available beyond this period—two minutes by default. The client should take this setting into account when determining the frequency of polling requests.

Control Data Caching for Business Objects

When application resources don't include sensitive data, you can define a caching strategy to safely store your business object's data and improve performance. Caching enables data to be retrieved from the browser cache, instead of the business object on the server, thus speeding up subsequent requests for this data.

Data Caching Options

Caching for each business object is controlled by its data caching strategy. You can choose predefined options (Sensitive, Private, or Public Shared) which combine flags used by the `Cache-Control` HTTP header to meet common caching scenarios. Or, you can use a combination of `Cache-Control` flags to build a custom caching strategy.

Here's a look at the different caching options in Visual Builder:

- **Sensitive**: Indicates that data must never be stored in any cache (private or otherwise). Select this option if your business object contains sensitive data, for example, banking transactions or personally identifiable information, that must not be cached and you want the latest most up-to-date information to be served every time.
  The **Sensitive** option is a combination of the `no-cache`, `no-store`, and `must-revalidate` flags that guarantee data is never cached. When a request is made for the data, the browser always checks with the server for the latest data.
- **Public Shared**: Indicates that data can be cached by the client's browser and other caches (like those from ISPs or other parties) for a specified duration. While this option is the least secure of all the available policies, select this one if your data is not sensitive and is not likely to change often, for example, country codes.
  The **Public Shared** option is a combination of the `public` and `max-age` flags that allow data to be stored in any cache for a maximum length of time.
- **Private**: Indicates that data can be cached, but only by the client's browser for a specified duration. Select this option if your data is user-specific, for example, a user's purchase order history that must not be stored in public caches but can stay in the client's browser cache. Remember though that anybody with access to the client would have access to the data as well.
  The **Private** option is a combination of the `private` and `max-age` flags that allow data to be stored only by the client's browser for a maximum length of time.
- **Custom**: (For advanced users) Specifies a custom option based on the `no-cache`, `public`, or `private` flags, each of which can be augmented by the `no-store`, `must-revalidate`, and `max-age` flags. Custom settings are not validated; select this option only if you are familiar with the `Cache-Control` header options and are confident of your choices.

Define a Data Caching Strategy

Configure the Resource Cache Control setting to define a caching strategy that safely stores a business object's data. Because all application data is deemed sensitive, by default, no data is cached.

Whether you cache data between the server and the browser or not at all depends on factors such as whether your data is sensitive, how often it is updated, and so on. Before you decide on a caching strategy, check whether it is safe to cache your data. If it is, decide on an option that makes the most sense for your data and specifies where, and for how long, the data can be cached. See Data Caching Options.

You can apply a different caching strategy for each business object, but be aware that the strategy applies uniformly to all its endpoints.

1. Select the business object's **Endpoints** tab.
2. Expand **Resource Cache Control** and select a suitable **Data caching strategy**:

Option	Steps
Sensitive	Select **Sensitive** if your data is sensitive and must never be cached (for example, banking transactions or personally identifiable information). This option is the default setting that disables caching.
Public shared	a. Select **Public shared** if your data isn't sensitive, is not likely to change often (for example, a list of country code values), and can be cached by the browser as well as other intermediary caches. b. In the **Max Age** field, enter how long the data can be cached. To cache your data for, say, a month, enter `2629746` as the max age in seconds. After this duration, the data will be considered stale and if a user is offline, they will continue to see the stale data. If you don't want this behavior, use the **Custom** option and add **must-revalidate** in addition to **max-age**. In this case, after the **max-age** duration, the data must be re-validated or downloaded again from the origin server.
Private	a. Select **Private** if your data is user-specific (for example, someone's purchase order history that must not be in public caches but can be stored in the client's browser cache). b. In the **Max Age** field, enter how long the data can be cached. To cache your data for, say, 24 hours, enter `89999` as the max age in seconds. After this duration, the data will be considered stale and if a user is offline, they will continue to see the stale data. If you don't want this behavior, use the **Custom** option and add **must-revalidate** in addition to **max-age**. In this case, after the **max-age** duration, the data must be re-validated or downloaded again from the origin server.

Option	Steps
Custom	a. Select **Custom** to build your own caching strategy. b. Select one or more of the available flags: **no-cache**, **public**, **private**, **no-store**, **must-revalidate**, and **max-age**. c. If you chose **max-age**, enter a value in the **Max age** field. **Note:** Custom settings are not validated. It is your responsibility to ensure the flags you select are valid combinations.

Your caching strategy is applied to data requests at runtime, when your web server adds the `Cache-Control` header to each endpoint response.

Note:

You can see your caching setting take effect only when an application is staged or published, not when you preview the application during development.

Index a Field to Speed Up Searches and Sorts

To improve the performance of searches and sorts for a business object's field, you can index the field, as explained by Index a Field.

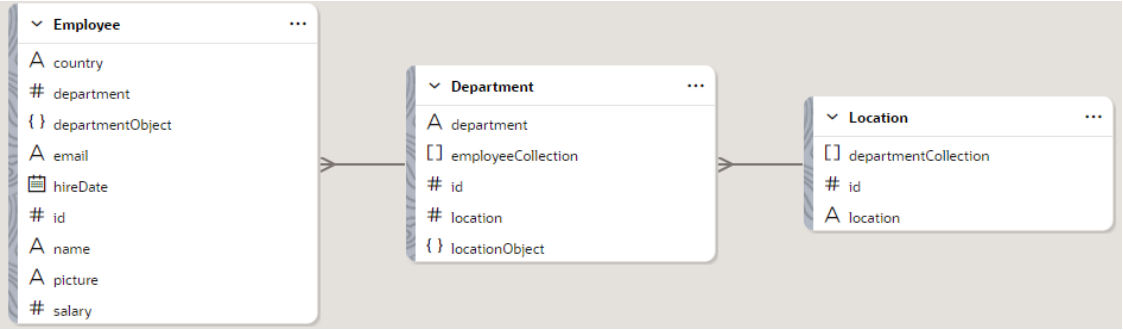
Work with Business Object Diagrams

When working with business objects, it's helpful to visualize business objects and their relationships. You can do this by creating diagrams for your business objects to show their fields and relationships.

To create a Business Object Diagram:

1. In the Business Objects pane, click **+** and select **Business Object 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 the Properties pane.
3. In the diagram's Properties pane, select 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.

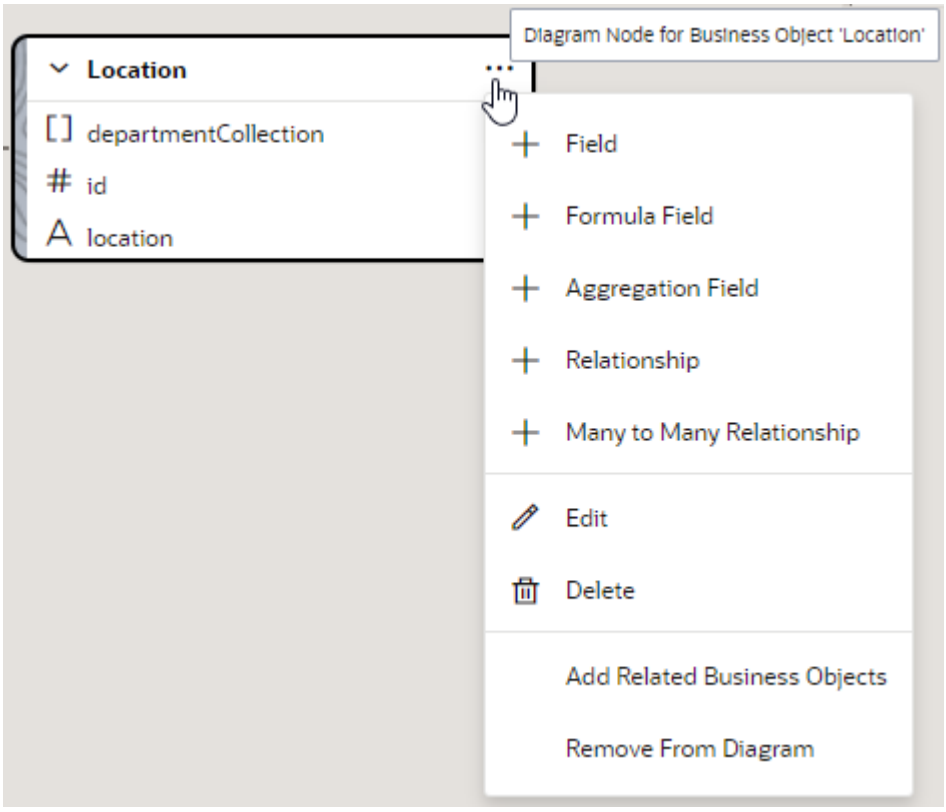
Here's an example that shows three selected business objects: Employee, Department, and Location. Each object's fields display in alphabetical order, along with their type. The relationships between the objects and the object accessor names through which you access referenced business objects are also shown:



What Can I Do With a Diagram?

You can use a business object diagram to complete many different tasks:

- Click the name of a business object to see the object's Properties pane, where you can view or edit overview information. Double-click the business object name to go to the object's Overview tab.
- Click a business object field to see the field's Properties pane, where you can edit its properties.
- Right-click a business object (or click Menu) to see a menu that lets you create a new field or a new relationship, edit or delete the business object, add related business objects, or remove the business object from the diagram:



- Click a relationship line between two business objects to see the Properties pane for the relationship, where you can edit or delete the relationship.

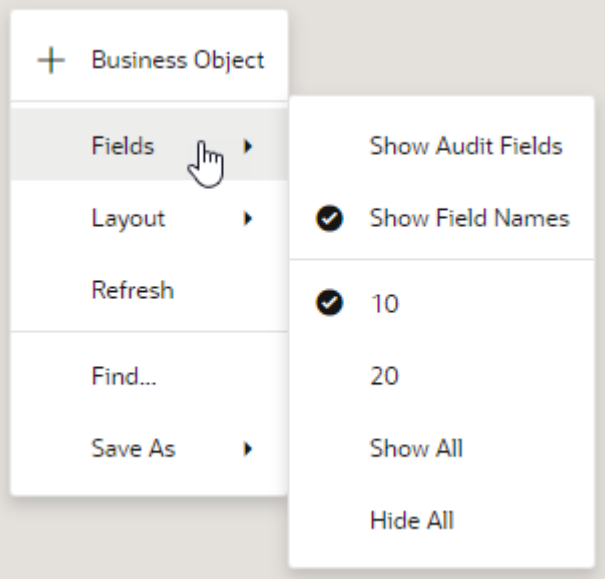
- If the diagram displays many business objects, right-click the canvas outside a diagram and click **Find** to select one of them.
- To refresh a diagram, right-click the canvas outside a diagram and click **Refresh**.
- To export the diagram to your file system, right-click the canvas outside a diagram, click **Save As**, and select either **PNG** (Portable Network Graphics) or **SVG** (Scalable Vector Graphics) as the format.

To delete a diagram, right-click the diagram in the Diagrams tab and select **Delete**.

How Do I Customize the Display?

Here's how you can change a business object diagram's display to suit your preferences:

- To change the default display from horizontal to vertical, right-click the canvas outside a diagram, click **Layout** and select **Vertical**.
- To show or hide a business object's fields, click the down-arrow next to the object's name.
- To change how business object fields are displayed, right-click the canvas outside a diagram and select **Fields**:



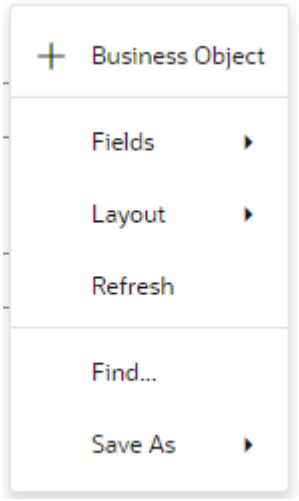
By default, reference fields are displayed (along with their object accessor names), and field names that match what you'll use in REST API calls or Groovy code are shown instead of display labels. The number of fields displayed is 10 (though you can scroll to view more). If you want to change this, from the **Fields** menu:

- 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 **Hide All** to hide all fields).
- Deselect **Show Field Names** to display fields by their display labels.

Create Business Objects with the Diagrammer

To create new business objects from a diagram:

1. Right-click in the canvas area outside a diagram and select **+ Business Object** from the menu:



   Alternatively, click **Menu** in the diagram's Properties pane and select **+ Business Object**.

2. In the New Business Object dialog, enter the business object name in the Name field. The Display Label is filled in automatically based on the Name you enter; change it, if you want. Click **Create**.

3. To add a field to the new business object, right-click the object and select **+ Field**. Enter the field name in the Label field and select the type, then click **Create Field**.

   To create a field based on a formula, make sure you select **+ Formula Field**, then define your field as described in Add a Formula to a Field.

   To delete a field, right-click the field and select **Delete**.

4. Click a field to open the field's Properties pane, where you can edit its name, data type, and other properties.

Create Relationships with the Diagrammer

To create new relationships between business objects from a diagram:

1. Right-click a business object and select **+ Relationship**.

   A dotted red line appears in the business object.

2. Drag the red line to the business object you want to connect to.

   The Create Relationship dialog appears, with a new Reference field in the source business object that includes the name of the target business object.

3. Select the cardinality for each object in the relationship (Many or One).

   In addition to many-to-one and one-to-many, you can specify cardinalities of one-to-one and many-to-many between business objects. If you specify Many-to-Many, an intersection business object with two reference fields is automatically created. Its default name is a concatenation of the two business objects; change this name as you want for better usability. For an intersection business object, you can't deselect the Required checkbox 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 checkbox, 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**.

   A relationship line in "crow's feet" notation that matches the cardinality you selected appears between the object nodes. Mouse over the relationship line to see a description of the relationship.

Right-click a relationship line and select **Edit** to modify 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.

Switch to Your Own Database Schema for Business Objects

If you connected your Visual Builder instance to an Oracle database (instead of the default embedded database), you can use your own schema in the tenant database to create business objects.

When you switch to a database such as DBaaS or Autonomous Transaction Processing (ATP), Visual Builder automatically manages the schemas and tables it uses for apps and business objects in your DB. In addition, this will let you build apps that can connect to a schema that already exists in the database and create business objects based on existing DB tables and views.

At the instance level, you can have some apps based on internal tables and some based on external tables, but within a visual app, you can have only one type of business object: either native business objects or those based on existing DB tables and views.

Business objects based on existing DB tables (including those backing materialized views) and views are, in general, similar to native business objects. You can create fields for all existing columns of supported data types (CHAR, VARCHAR, VARCHAR2, NCHAR, NVARCHAR2, NUMBER, BINARY_DOUBLE, BINARY_FLOAT, DATE, and TIMESTAMP without timezone specifications)—though foreign key columns that reference other DB tables need the related business object to be created first. Any foreign key column that links to another DB table can be used to build business object relationships. It's also possible to add calculated fields, business rules, cache control definitions, and security settings, much like what you'd do when working with native business objects.

However, keep these restrictions in mind when creating business objects based on existing schema:

- The `PRIMARY KEY` constraint, used to manage a table's primary key, is limited to a single column. Similarly, only single-column unique and foreign keys are supported.
- Audit fields (that track who created or updated a field when) and the ETag mechanism (that protects against users overwriting changes) are not added automatically as they are in native business objects, but you can set your own fields for auditing based on existing columns.

Create a Business Object Based on a DB Table or View

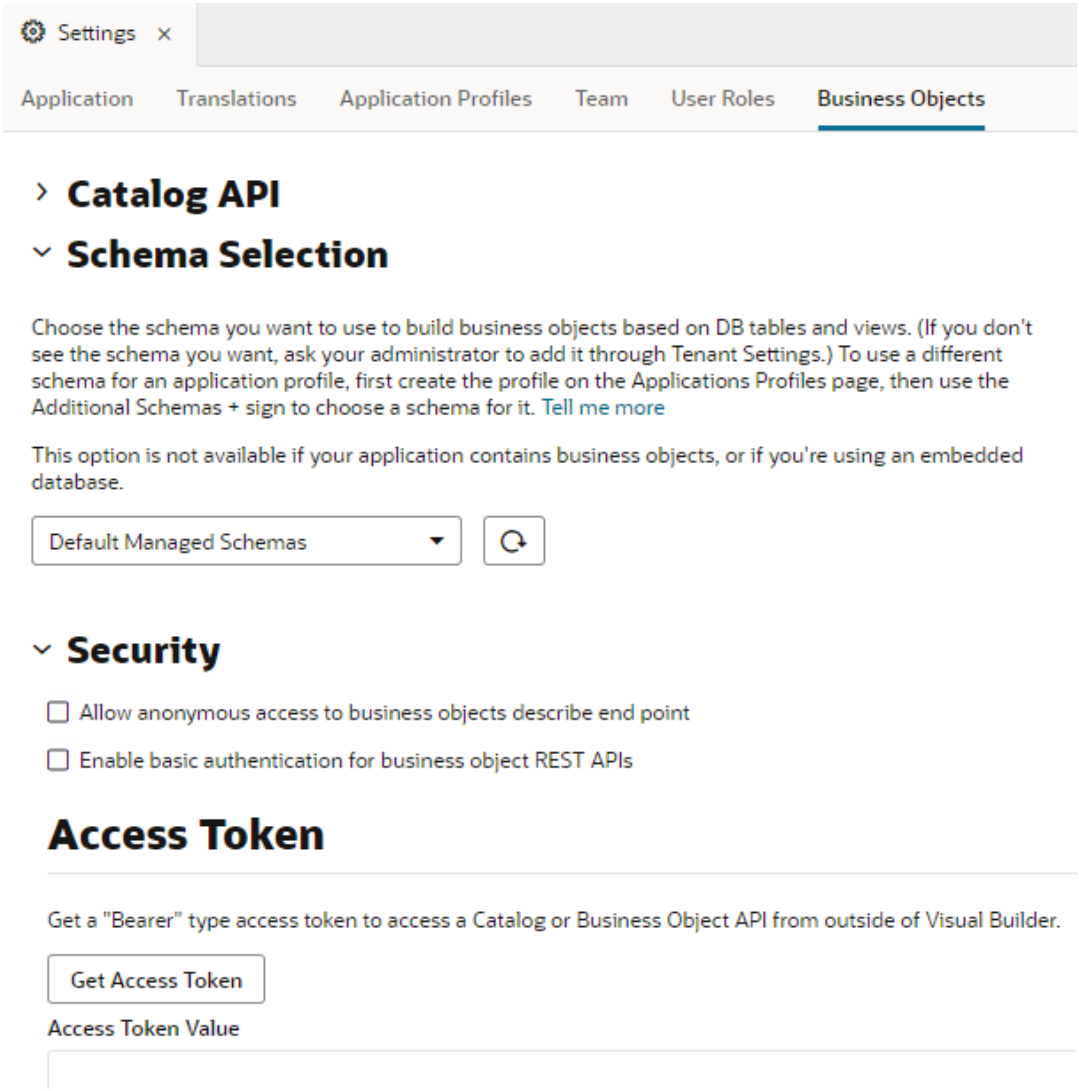
To create business objects based on tables and views in your DB schema, you switch your visual application's database schema, then use the Create Business Object wizard to create a business object based on an existing table or view.

Before you begin:

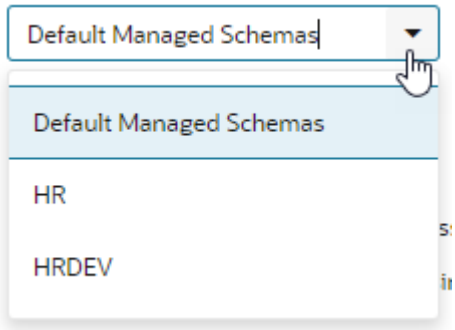
- Your Visual Builder instance must be configured to use another Oracle DB and the schema must already exist in the tenant DB. Talk to your administrator to confirm this setup.
- Your visual application must not have any business objects defined.

To create a business object based on an existing table or view:

1. Click **Menu** in the upper right corner and select **Settings**, then click **Business Objects**.



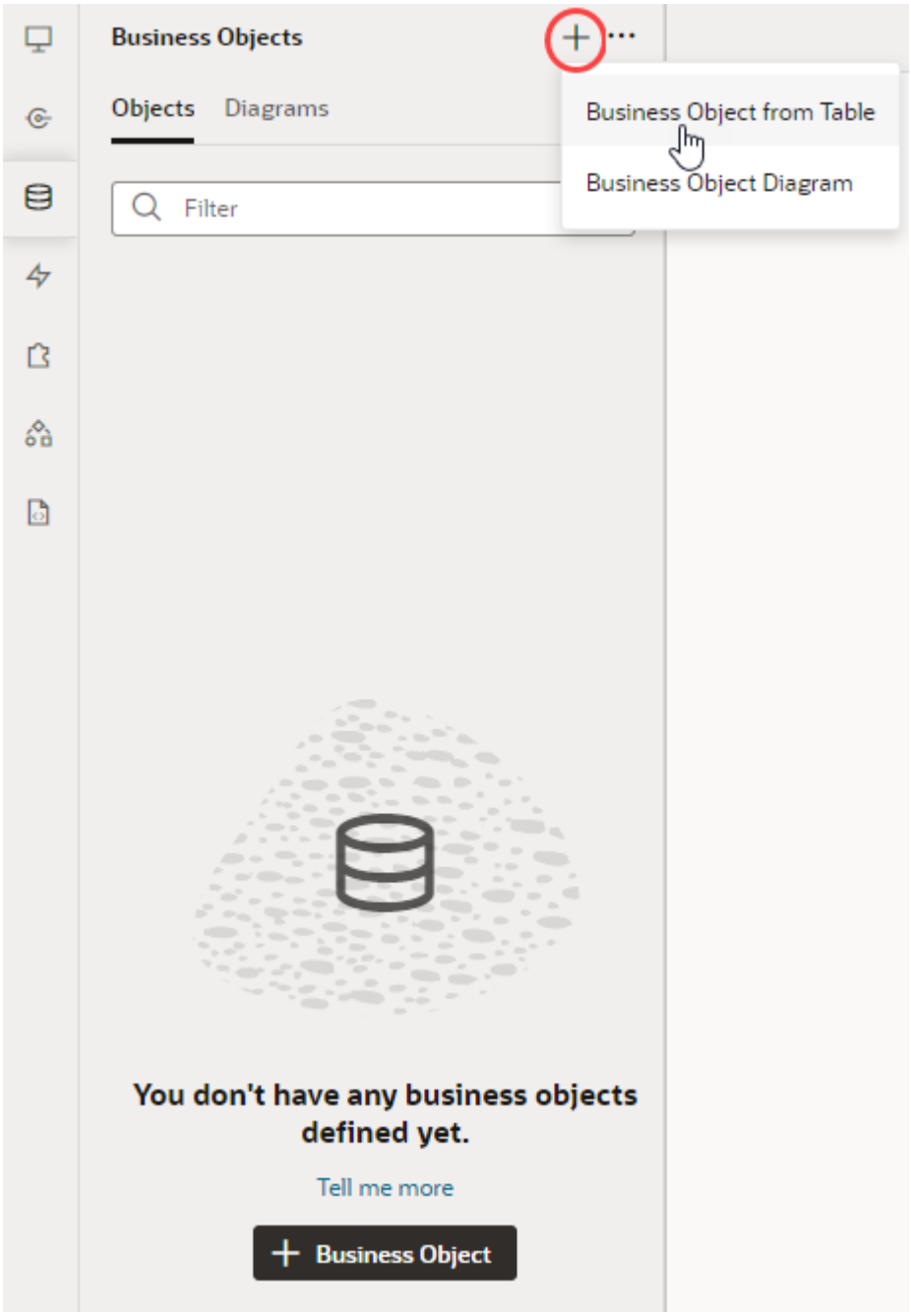
2. From the Schema Selection list, select your schema.



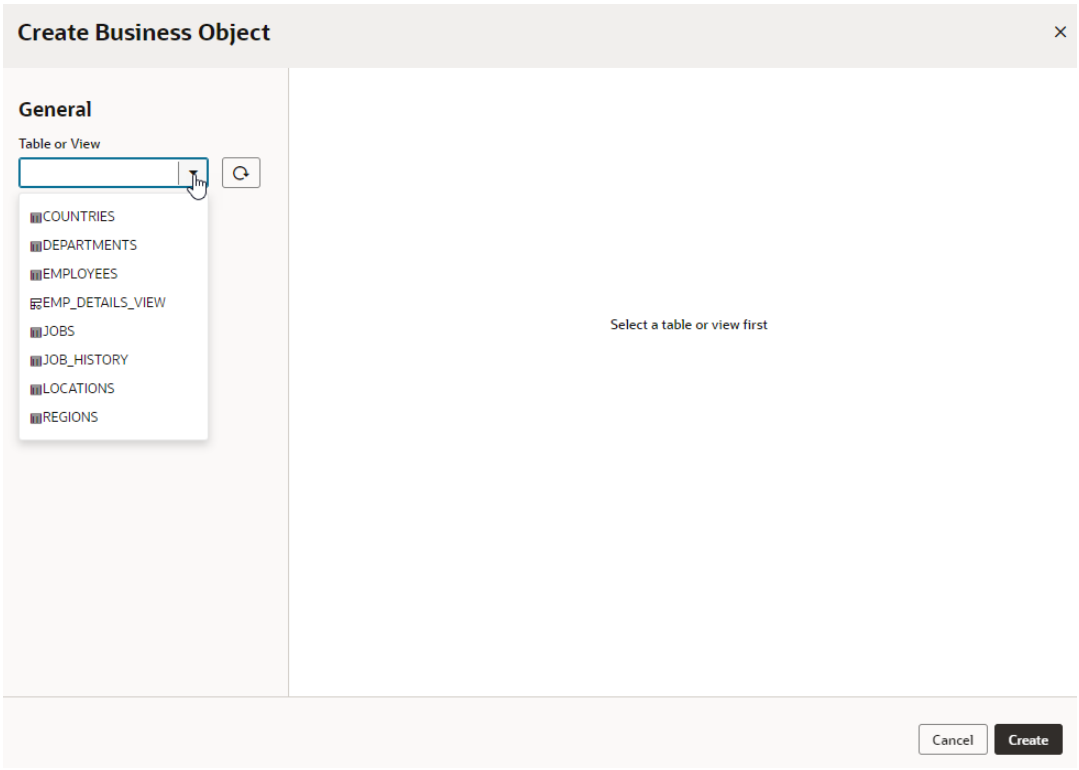
The list of schemas available to you is defined by your administrator in the Tenant Settings of your Visual Builder instance. If a particular schema isn't listed, talk to your administrator to add it as an available schema for your instance.

> **Note:**
>
> Details of the tables and views available for use in a schema are fetched once and cached. If the schema (or tables/views within it) was changed, click to pick up the latest updates for a particular schema. You can also do this when creating a table or view for a schema during development.

3. Optional: To simplify the task of changing schemas when your app is deployed to different environments like development, test, or production, you can add additional available schemas and associate them with different application profiles (see Switch Schemas Used During an App's Lifecycle). If you don't add additional schemas, the schema you selected in the previous step is used as the default schema for all deployments.
4. Click **Business Objects** in the Navigator, then click **+** and select **Business Object from Table**.



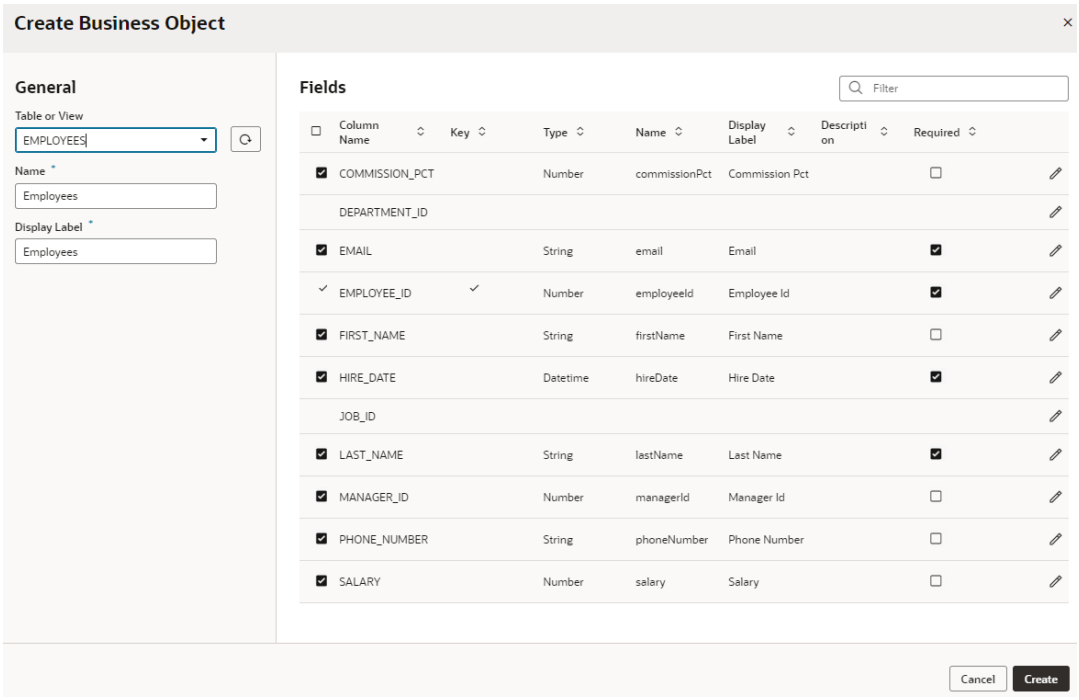
5. In the Create Business Object wizard, select a table or view.



6. Look over the table and column data that displays based on your selection.

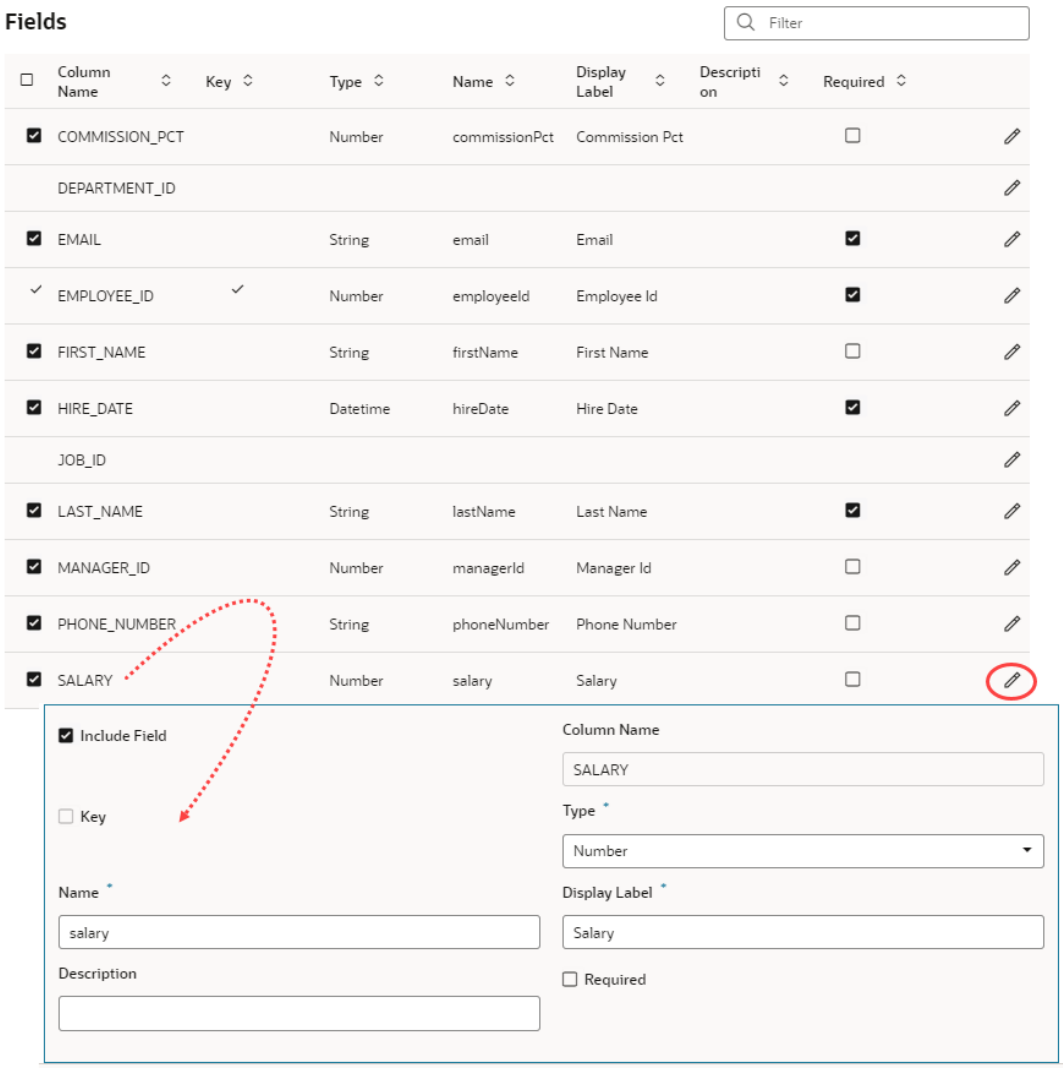
   The Business Object Name and Display Label fields under General are populated based on your table's name and can be edited. If you selected a view or if the primary key cannot be identified for some reason, select the field that should be the primary key in the **Select Primary Key** field. The primary key does not have to be numeric, but it must be a column that contains unique data and no NULL values.

   A list of Fields on the right shows the table's columns that you can add as fields to your business object. A check mark in the first column indicates the column's fields that will be added.



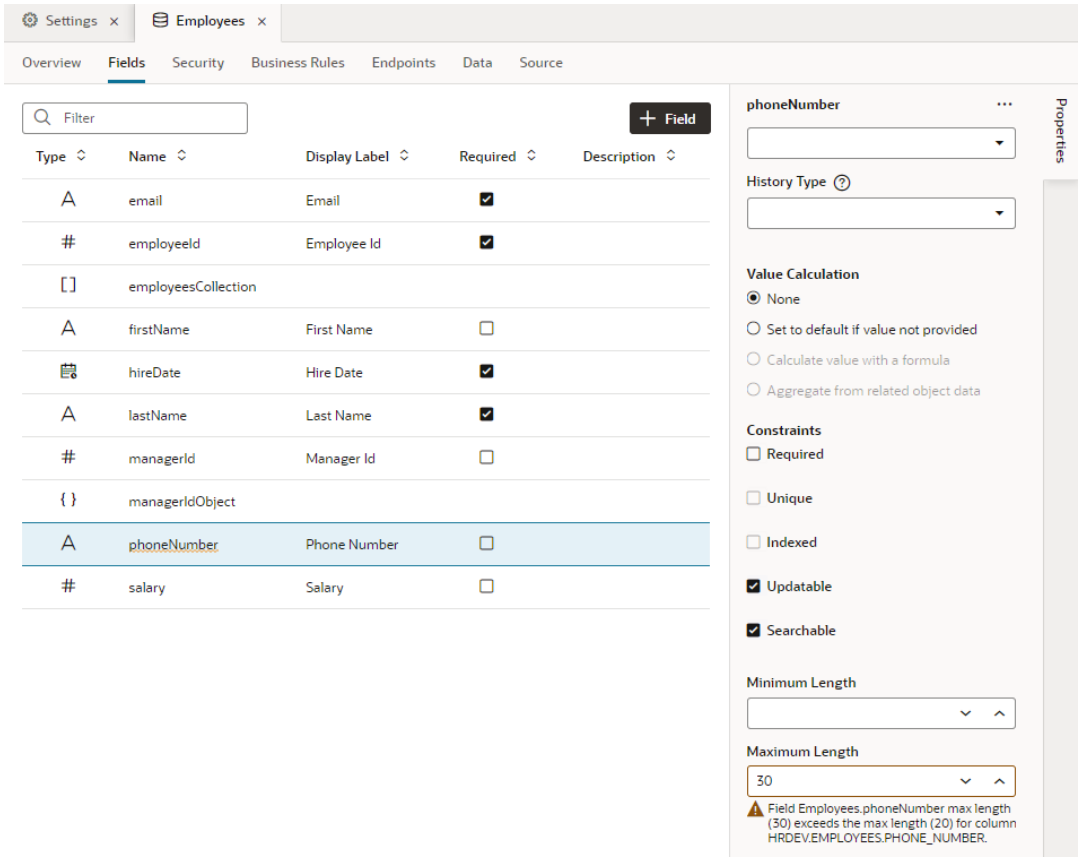
This Fields list does not include unsupported data type columns or foreign key columns that link to tables that don't yet have a business object; for example, the DEPARTMENT_ID column in the image can't be added as a field because a business object based on the referenced DEPARTMENT table doesn't exist yet.

The Field Name and Display Label are based on the table's column name. If you want to change the default values, double-click the field or click in the row and make changes. You can also update the Required property. If you don't want a particular column to be included as a field, deselect the check mark in the first column, or deselect **Include Field** when the row is in edit mode:



7. Click **Create**.

   A business object created based on your schema shows in the Business Objects tab. You can add new fields to this object (including calculated fields) based on the existing table/view and modify existing fields. If a field's metadata is not compatible with the table/view column properties, you'll see validation messages, similar to those for native business objects. For example, when a field is based on a VARCHAR2(20) column but the Maximum Length property is set to a higher value, you'll see a warning around the property and in the Audits pane:



You can also set up business rules, security, and call REST API endpoints from components. In general, you manage these business objects similar to how you'd manage a native business object, but because you cannot use the Business Objects editor to make changes that would (in native business objects) update the schema, some options in the editor won't be available for objects based on existing DB tables or views:

Option in Business Objects Editor	Native Business Object	Business Object Based on Own Schema
New Relationship (**+**) button in Overview tab	✓	X
Contains Application Setup Data property in Overview tab	✓	X
Unique and **Indexed** fields in Properties pane	✓	X
Convert an existing field to a calculated field: **Calculate value with a formula** and **Aggregate from related object data** fields in Properties Pane	✓	X
Change cardinality, target business object, or Delete Rule in the Relationship editor	✓	X
Data Manager in the Business Objects pane's Options menu	✓	X
Import From File and **Export CSV** options in Data tab	✓	X

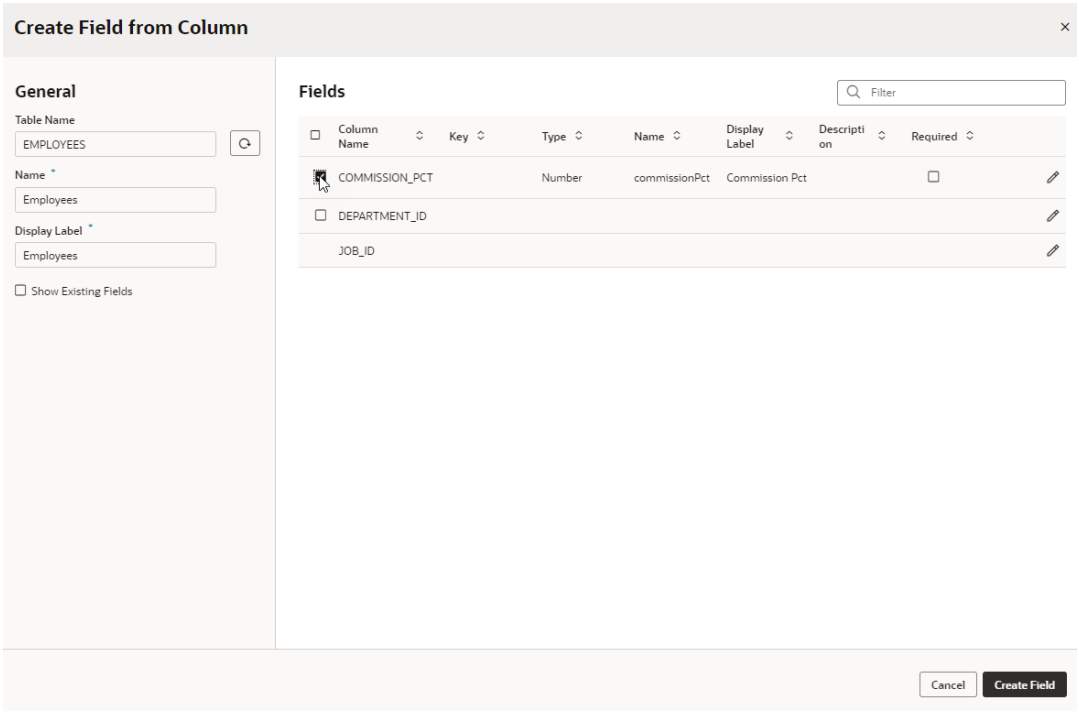
Add Fields to a Business Object Based on a DB Table or View

After a business object has been created from an existing table or view, you can add fields based on other existing columns in the table or view. You typically do this when the table or view has been modified, or because a referenced business object is now available.

> **Note:**
>
> You cannot use the Business Object editor to add fields that would require a new column to be created. To add such a field, you must add a new column to the database table (or redefine the view), then add the field for that column.

To add a field to a business object based on an existing table or view:

1. Open the Business Objects page and click your business object's **Fields** tab.
2. Click the **+ Field** button, then select **Field From Column**.
3. In the Create Field from Column wizard, look for the field you want to add. Fields that don't already exist and can be added show in the wizard automatically. Select the check mark in the field's first column, or click and select **Include Field**; update other values as needed.



If you want to see the fields already added to the business object, select **Show Existing Fields** in the left pane.

Click **Create Field**.

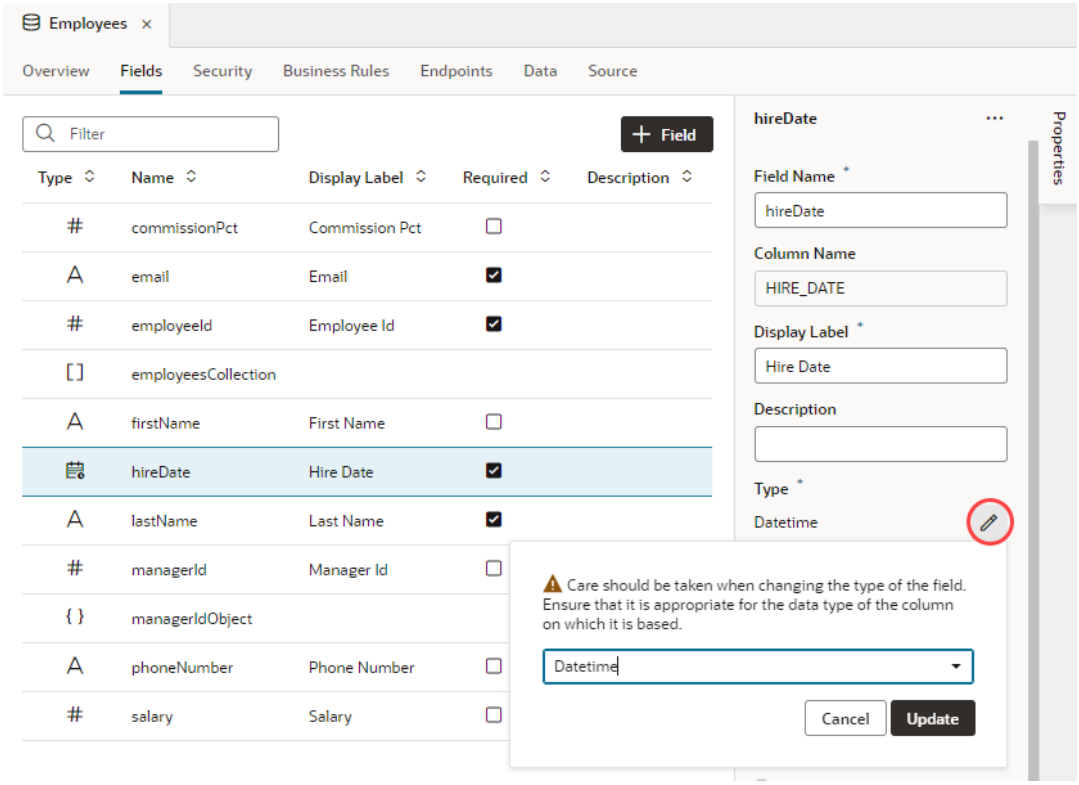
Change the Data Type of a Field Based on a DB Column

When you add a field based on an existing DB column, the field's type is set to the column's default data type. You can change this data type after creation if needed. For example, when you have a field of type Datetime (where a date or time is stored in a TIMESTAMP or DATE column), you might want to change the field's type from Datetime to Date.

> **Caution:**
>
> Before changing an existing field's data type, ensure that the new type is appropriate for the data type of the column on which it is based.

To change the default data type of a field added to a business object based on an existing table or view:

1. From your business object's Fields editor, select the field whose type you want to change.
2. Under **Type** in the Properties pane, click next to the current data type.



3. Select the type you want to change to and click **Update**.

Create Calculated Fields for a Business Object Based on a DB Table or View

You can create formula and aggregate fields for a business object based on an existing DB table or view to calculate a field's value.

- To create a field that calculates its value based on a formula:
  1. Select **+ Field**, then **Formula Field** from your business object's Fields editor.
  2. In the Formula Field dialog box, enter a label for the formula field you want to create, then select the field's Type.
  3. Enter a valid expression for the formula in the text area. 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. Click an operator in the toolbar to add it to the formula.

     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`).
  4. Click **Create Field**.
  5. Optional: When you define a Groovy expression, you can click the icon in the Properties pane to override the default Groovy timeout.
- To create a field that aggregates the data of related business objects:
  1. Select **+ Field**, then **Aggregation Field** from your business object's Fields editor.

     The **Aggregation Field** option won't show if your business object doesn't have incoming relationships. For example, if you created a field in the Employee business object that refers to the Department object, Employee is considered the source object and Department the target object. Department, in this case, is said to have an incoming relationship with Employee.
  2. In the Aggregation Field dialog box, enter a label for the aggregation field you want to create.
  3. Select the object to aggregate (for example, the Employee business object which uses Department as a target business object), the aggregation function (which can be Average, Count, Maximum, Minimum, or Total), and the field to aggregate (for example, Salary).
  4. Click **Create Field**.

Set a Field for Auditing

Audit your business object's history using fields based on existing columns in your DB table or view. For example, let's say your table has the DATETIME_CREATED, DATETIME_MODIFIED, USER_CREATED_BY, USER_MODIFIED_BY, and VERSION_NUM columns; you can add these columns as fields to your business object and use them to track history.

Audit fields maintain a history of changes made to your business object by tracking who created and updated an object when. They also include a version number used in the generation of an entity tag (ETag) which protects against users overwriting changes. Because audits can help you verify changes, you might want to set audit fields for each of these history

types: `Created By`, `Updated By`, `Time Created`, `Time Updated`, and `Version Number`. A warning will be logged in your app's Audits pane for each history type that isn't assigned to a field. You'll also see an error if the same history type is assigned to more than one field.

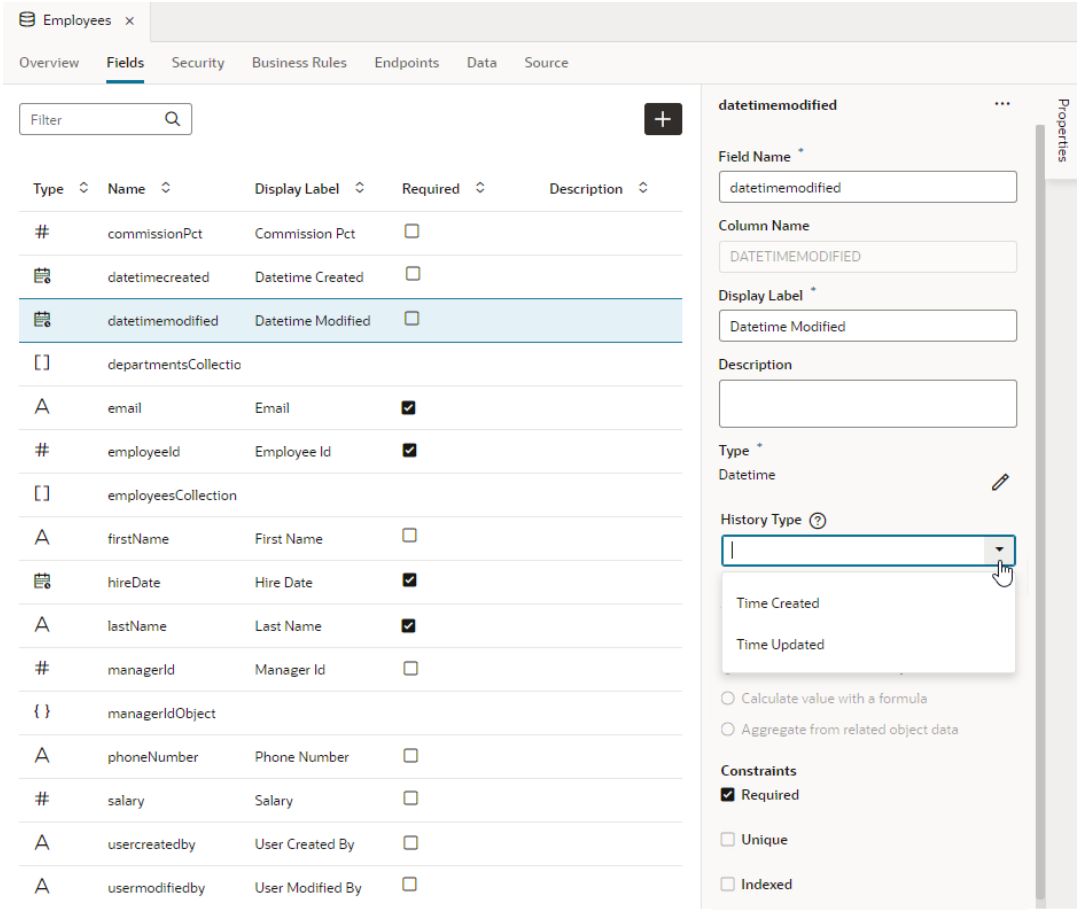
Before you use your own fields for auditing, make sure the column data types used by those fields are compatible:

Purpose	Column Data Type
Created By	VARCHAR2(200)[1]
Creation Date	TIMESTAMP(6)
Updated By	VARCHAR2(200)[1]
Updated Date	TIMESTAMP(6)
ETag	NUMBER

1 200 is an example.

To set a field added to a business object based on an existing table or view as an audit field:

1. From your business object's Fields editor, select the field you want to use for auditing.
2. In the field's Properties pane, select an option in the **History Type** list.



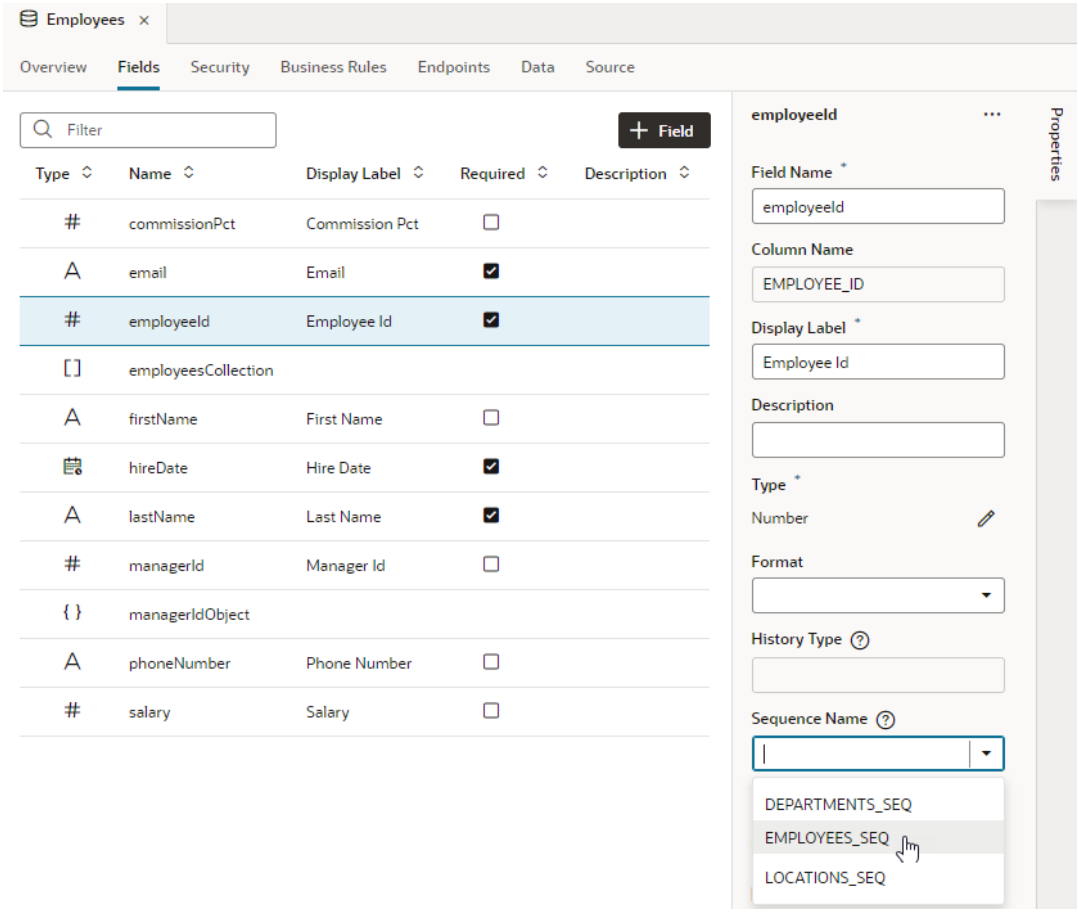
Your options depend on the field's data type. For a datetime-type field, you see **Time Created** and **Time Updated** (as shown here). For a string-type field, you see **Created By** and **Updated By**, and **Version Number** for a number field.

Use a Sequence for a Primary Key Field

The primary key in a DB table is a single field that uniquely identifies a record. When your primary key column is not an Identity Column or if a "Before Insert" database trigger that could be used to populate its value isn't defined, you can specify a database sequence to populate the primary key field's value.

To set a sequence to populate the primary key field:

1. From your business object's Fields editor, select the primary key field. If necessary, click **Properties** to open the Properties pane.
2. From the Sequence Name list, select the sequence you want to use to populate the field. The Sequence Name field shows the list of all available sequences in the schema.



Switch Schemas Used During an App's Lifecycle

When you use your own DB schema, you can switch the schema that your app connects to during its lifecycle. For example, you might want to use a particular schema during the

development phase, but switch to another one for testing or for production. You do this using *application profiles*.

Application profiles help to define combinations of servers and security settings for every environment that you want to deploy your visual app to. You define multiple profiles for an app, with each one containing a different configuration appropriate for the environment. For example, you can have a profile for testing that connects to a test environment, then associate it with the schema used to access testing data. Similarly, you can associate a production profile with a schema for live data.

If a profile isn't explicitly mapped to a particular schema, it will always use the default schema.

To associate a schema with a profile:

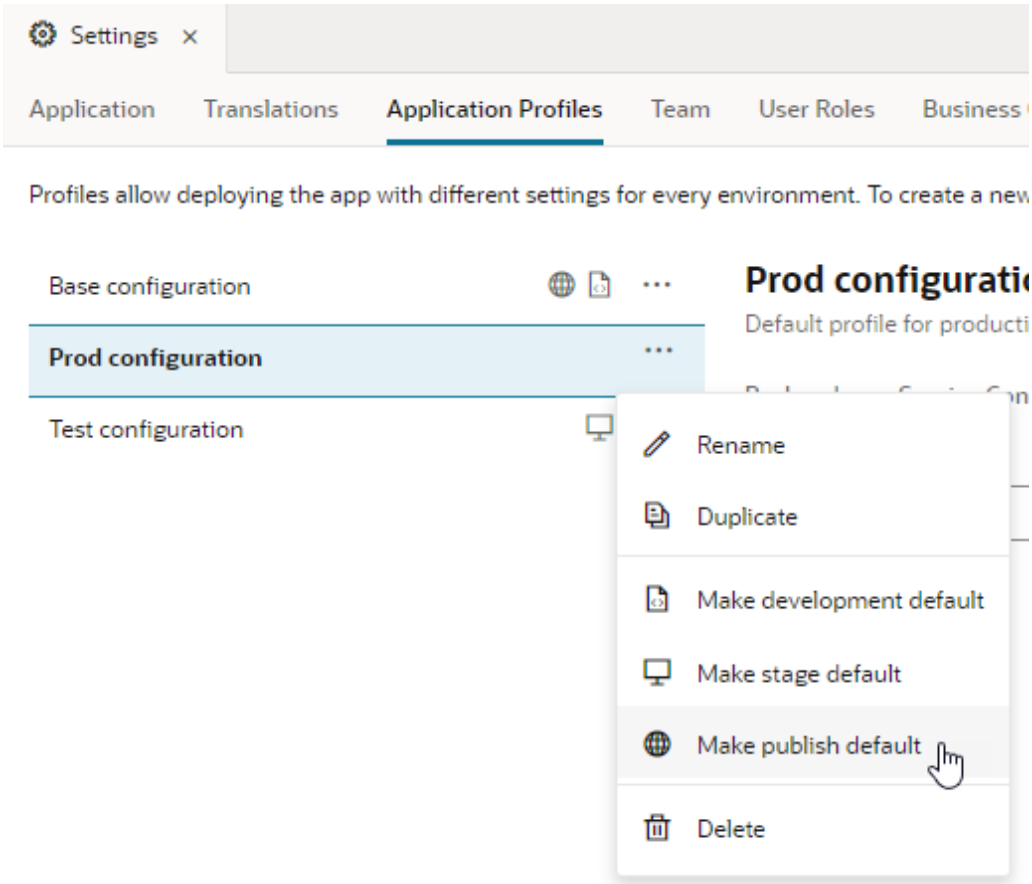
1. Create the application profile you want to use with your schema.

   Visual Builder provides a ready-to-use application profile (Base configuration) that is used by default for development, stage, and production deployments. You can duplicate the default one to create another profile, then associate it with the schema you want to use. If you've already defined the profile you want to use, skip this step.

   a. Click **Menu** in the upper right corner and select **Settings**, then click **Application Profiles**.

   b. From the Base configuration's menu, click **Duplicate**, then provide a new name (for example, Prod configuration) and description in the Duplicate Application Profile dialog. Click **Duplicate**.

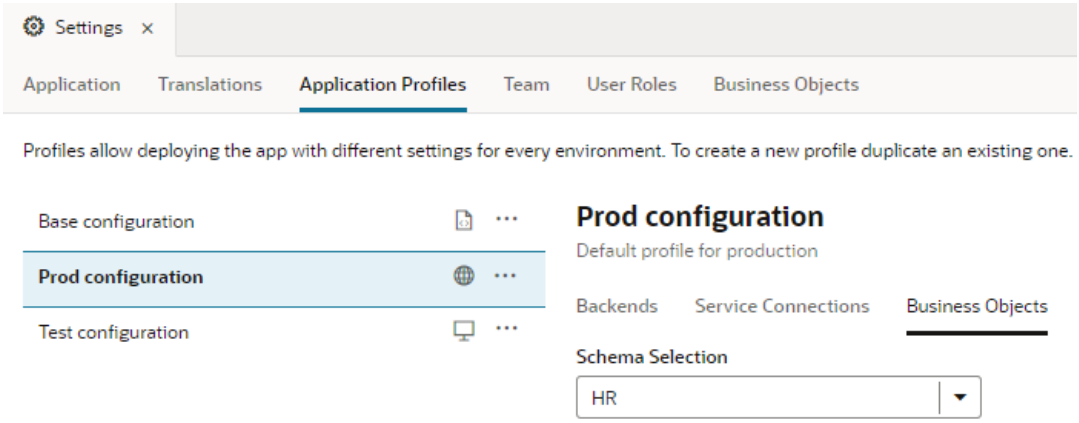
   c. From the newly created profile menu, select **Make publish default** to use this profile as the default when the app is published.



2. Associate the profile with the schema to be used during a particular phase.

> **Note:**
>
> Make sure the schema you want to use is defined in the Tenant Settings of your Visual Builder instance. If it isn't listed, talk to your administrator to add it as an available schema for your instance.

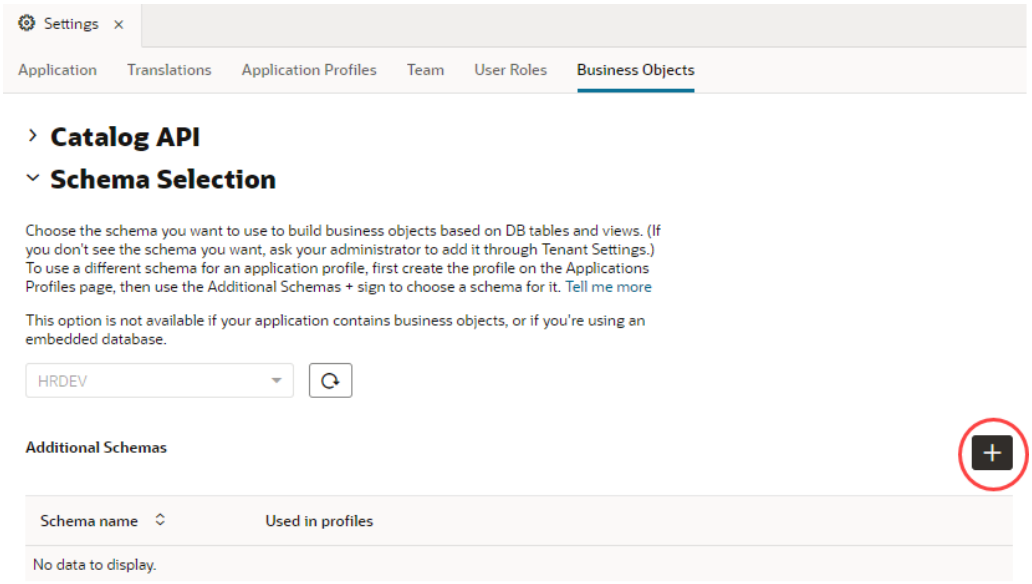
It's possible to map a profile to a schema in the Application Profiles page by selecting the profile you want to manage, then using the profile's **Business Objects** tab to associate a schema, as shown here:



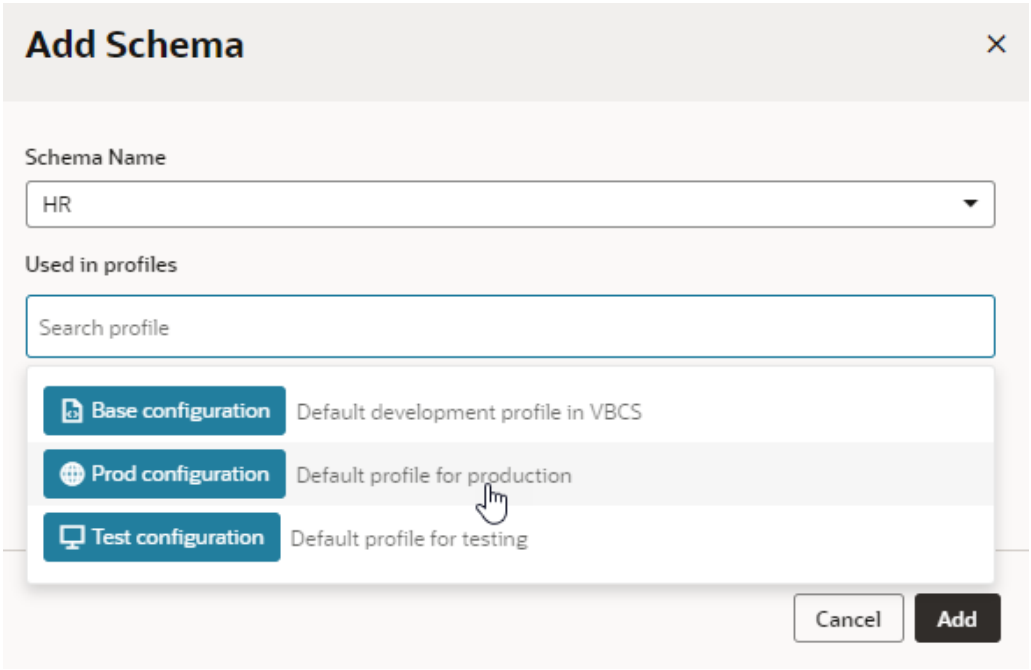
But the **Business Objects** tab on the Settings editor provides a more visual way to map schemas to profiles, as described here:

a. Click **Menu** in the upper right corner and select **Settings** (if necessary), then click **Business Objects**.

b. Under Schema Selection, with your schema already selected, click next to Additional Schemas.

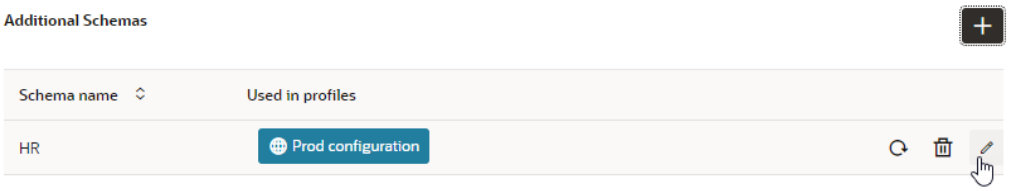


c. Select an available schema (for example, HR), then from the **Used in profiles** list, select the profile you want to associate the schema with. You can add more than one profile if you want.



d. Click **Add**.

The schema (mapped to the profile) appears under Additional Schemas. If you want to update the profile, click in the schema's row and make changes in the Used in profiles column. You can change the selected profile as well as add other profiles.



Note:

Details of the tables and views available for use in a schema are fetched once and cached. If the schema (or tables/views within it) was changed, click to pick up the latest updates for a particular schema.

5

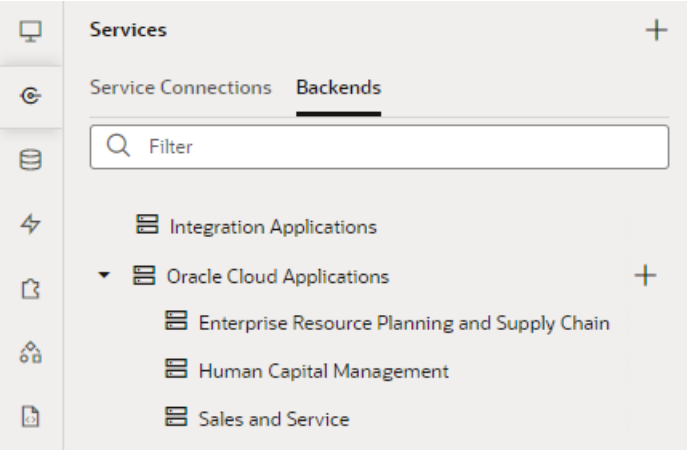
Work With Services

Many of the UI components in your application are bound to data that comes from REST APIs, which in turn are provided by *services*, like Oracle Cloud Applications. These services are made available to your visual application through backends and service connections.

A *backend* is a representation of an external system whose data you wish to consume. A *service connection* provides basic information about a service, including the REST endpoints and schema provided by the service, in an Open API-compliant format.

Service connections are defined at the visual application level, which means that all the web apps in your visual application can use them.

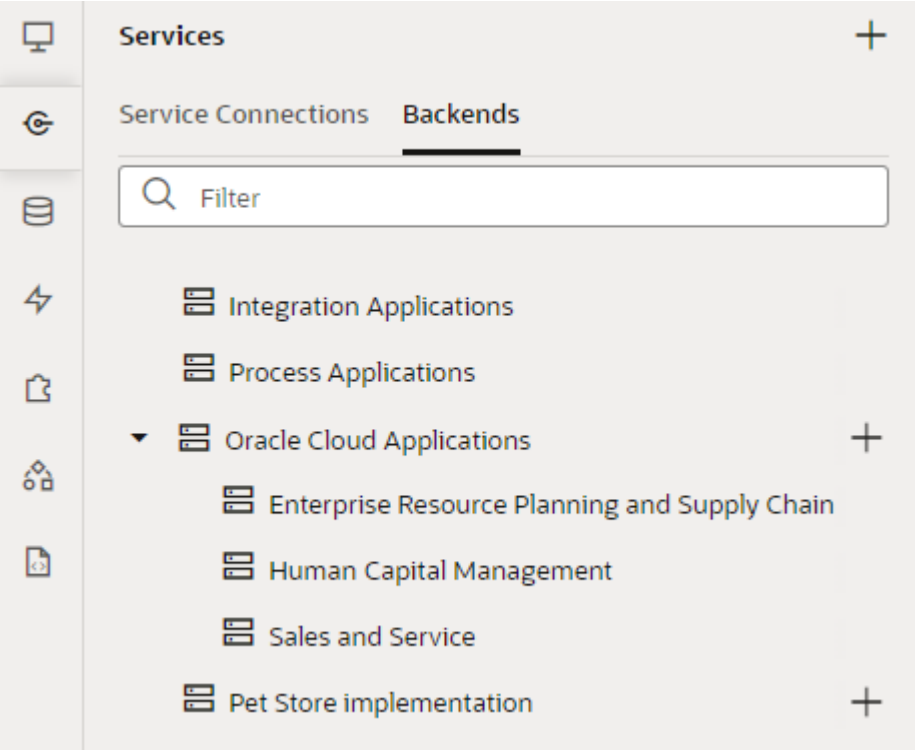
Backends help you manage the servers on which the REST APIs are hosted. By gathering server details together in one place, backends make it easier to create service connections, as well as to manage the server details themselves. For example, the "Oracle Cloud Applications" backend, which is automatically provided for most visual applications, points to the servers hosting the Enterprise Resource Planning and Supply Chain APIs, the Human Capital Management APIs, and more:



Manage Backends in Your Visual Application

Backends define servers that your visual application can access. You can view and manage the backends available to your application on the Backends tab, which you access from the Services tab in the Navigator.

The following image shows backends—for example, Integration Applications and Oracle Cloud Applications—that have been defined in the tenant-level catalog:



The administrator for this Visual Builder instance has configured the Oracle Cloud Applications backend to connect to an Oracle Cloud Applications instance that provides access to the three child backends shown under the top-level backend: Enterprise Resource Planning Supply Chain, Sales and Service, and Human Capital Management. You can create custom backends to map to other types of servers, either by providing an OpenAPI/Swagger file or by pointing to the URL of a custom Oracle ADF Describe.

If a backend hasn't been defined for your Visual Builder instance or you want to override the settings for your Visual Builder instance and create a catalog just for a particular application, use the Services tab to create a new backend or edit an existing backend.

For each backend, you can use the following tabs to view and edit the backend's details:

Tab	Description
Overview	Displays the name and type of the backend (which can be Integration Applications, Oracle Cloud Applications, or a custom backend). You can use the **+ Service Connection** button to create a service connection based on the backend. If any service connections have been defined for the backend, they'll be listed below the **+ Service Connection** button. Dynamic service connections, which are defined in the `catalog.json` file, are automatically loaded and displayed below the button. Static services can be loaded by selecting the **Load more related connections** link, after which all `openapi3.json` files will be loaded and parsed, locating all services connections that were defined from the backend.

Tab	Description
Servers	Displays the servers associated with the backend and includes the instance URL and the application profile associated with the instance. You can add, edit, or remove backend servers. You may have one or more servers if the backend is hosted on different instances. You use the **+ In-Source Server** button to add new servers where you specify details such as: • The application profile to associate with the server • Headers • Security and connection details
Headers	Displays the static headers defined for the backend at the server level. You can add and edit headers in the tab.
Source	Displays the description of the backend stored in the tenant-level `catalog.json` file. If you override the tenant-level definition, this file shows the contents of the application-level catalog from the `services/catalog.json` file. You can edit the entries in the Source tab, if you want.

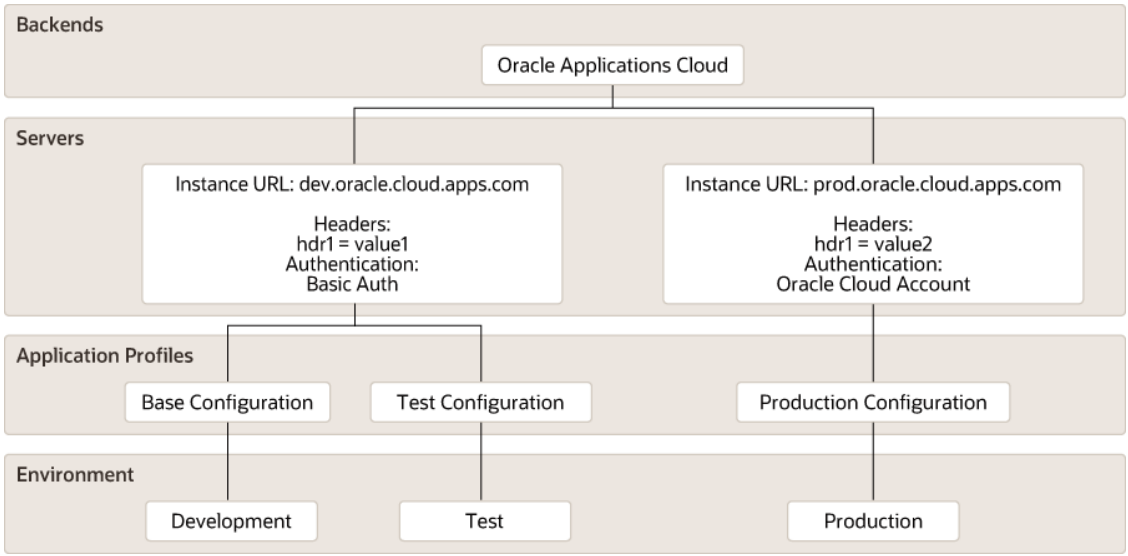
> **Note:**
>
> If you do not see any services in your catalog after confirming the URL and authentication method are correct, contact your administrator to confirm that you have the proper credentials and that your user role is authorized to access services from the backend instance.

What Are Backends?

A backend is a collection of servers that your visual application uses to access REST endpoints in a known resource, like Oracle Cloud Applications or Oracle Integration.

Each backend is associated with one or more servers, and each server has different instances for development, test, and production. These backend instances are defined in the Tenant Settings of your Visual Builder instance by your administrator and provide the out-of-the-box catalog that all visual applications in your instance can access. Your application uses these backend definitions (instance URL, authentication type, and so on), in association with application profiles, to connect to the right server, so you can successfully develop, test, and

deploy your application, as shown here:

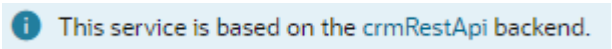


A typical backend catalog includes the built-in Integration, Oracle Cloud Applications, and Process Automation backends. You can also create backends that map to other types of custom servers.

The catalog is described in an OpenAPI-compliant file called `catalog.json`, which contains the list of backends and their details at the tenant level. Because this definition applies to all visual applications, you can easily connect to REST APIs from the predefined catalog without having to specify instance URLs, authentication, and other details every time you create a visual application. You can choose to override the tenant-level definition for a single application, either to customize an existing backend (say, connect to a different instance or change the authentication method) or to create your own—all of which is then stored at the application level in the `services/catalog.json` file.

When you create a service connection from a catalog, the service connection resolves using the backend configured at the application level and, if not present, it resolves to the backend configured at the tenant level.

All new service connections will require a backend. If a backend doesn't exist, you'll be prompted to create one. Service connections created previously without a backend will continue to work and their configuration settings can be changed as well. You can create a custom backend when you create a service connection or create the backend first, and add the service connections to it later. You'll know when services are "derived" from a backend because you'll see something like this:



Clicking the backend will take you to the Backends editor, where you can edit the backend's details if you want.

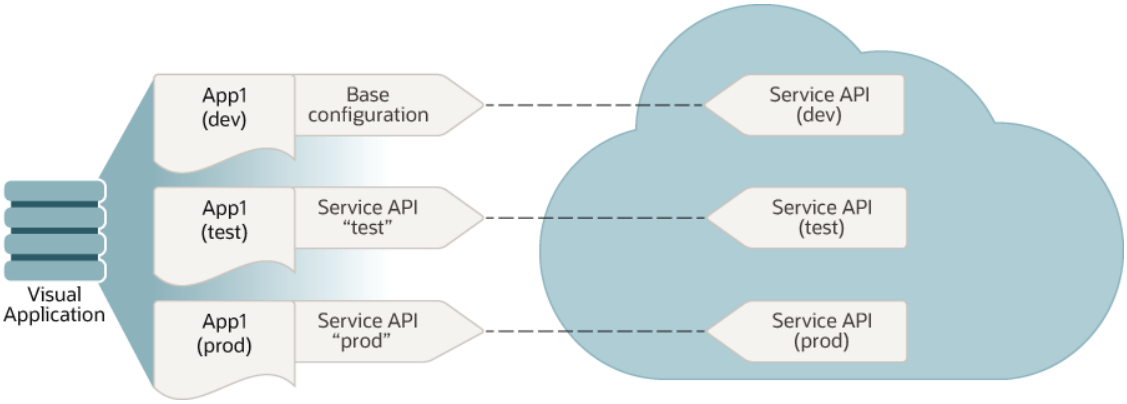
What Are Application Profiles?

Application profiles in Visual Builder facilitate the development of applications that consume REST services.

As you develop your web (and mobile) apps, you need access to the REST services that your app will consume when it goes into production. Given that it may not be possible or appropriate

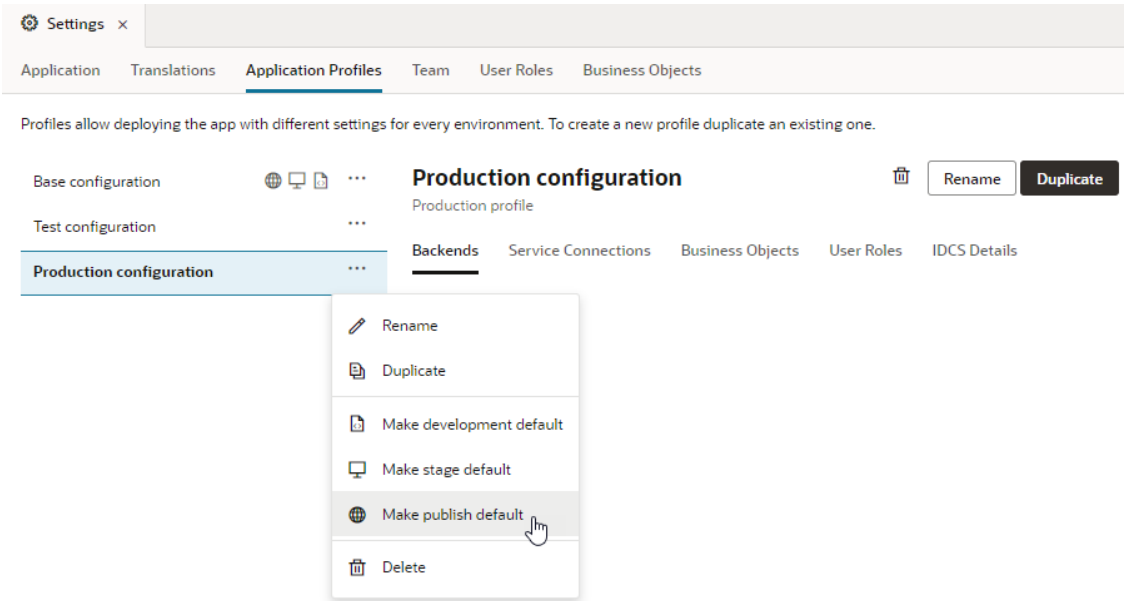
to use production instances of the service as you develop your app, you use development and test instances that provide the same APIs as the production instance of the service for the development and test phases of application development.

To make the task of changing between development, test, and production instances easier, Visual Builder provides you with the ability to define application profiles in your visual application where you specify the appropriate details to use for an app when you deploy it to a development, test, or production environment. For example, basic authentication may be acceptable to use during the development of your application, but it is not recommended that you use this authentication type when you deploy your app to production. In this scenario, you can configure an application profile for development where you use the basic authentication method that is less onerous to implement. The application profile that you use to deploy the app to production uses a more secure authentication method.



You access application profiles from the Application Profiles tab of your visual application's Settings page. Visual Builder provides one ready-to-use application profile (Base configuration) that is the default application profile to use when you deploy web (and mobile) apps in this visual application to development, stage, and production.

To create additional application profiles, click **Duplicate** and provide a new name (for example, `Production configuration`), ID, and description in the Duplicate Application Profile dialog. Having created the new profile, you can make it the default to use when you stage or publish your visual app by selecting the appropriate option from the menu that appears to the right of the newly created application profile:



The application profiles that you create can be associated with the various servers (dev, test, and prod) used by the REST service that your app consumes.

In the following example, a backend that returns contact information (**contactsBackend**) lists four servers that host the relevant REST service. The development server uses the Development Configuration application profile, while the staging and production servers use their corresponding application profiles.

Set the Backend's Authentication Method and Connection Type

A backend represents an external system that has REST APIs you want to use in your web app, and to access them, you'll need to specify the required authentication method and connection type for the access.

The *authentication method* determines how to obtain permission to access the REST APIs. The *connection type* determines how the REST APIs are reached: either directly through JavaScript, or through a server-side component called a *proxy*.

The dialogs for creating and editing a backend's server details allow you to:

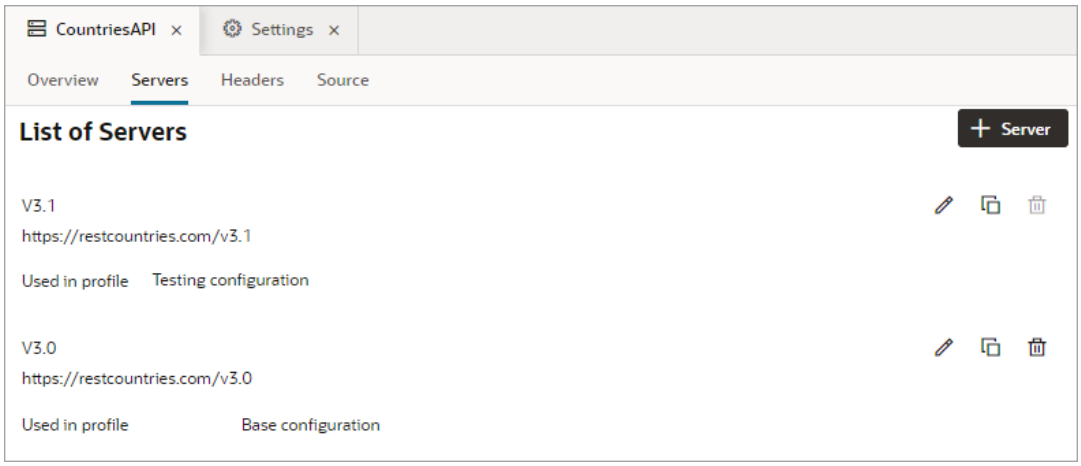
- Override authentication settings set at the environment's tenant settings.
- Manage the credentials for accessing the service (if credentials are required).
- Manage identity propagation of the end user logged into the web app (if the service supports the standard IDCS OAuth flows).
- Manage how your application connects to the service (via proxy or via Direct call); to bypass CORS, the "Always use proxy, irrespective of CORS support" connection type is provided.
- Manage how anonymous users can access the application.

To connect to a service that is available through HTTPS, authentication is not required, and there's no CORS requirement. The default setting of **None** for the **Authentication** field is sufficient. In this case, any end user (anonymous or authenticated) of the web app can access the service.

To connect to a service that requires authentication, you need to select the appropriate authentication method from the **Authentication** list. There are two types of authentication methods that can be used for a backend, distinguished by whether the user's identity is passed to the service (identity propagation) or not (fixed credentials).

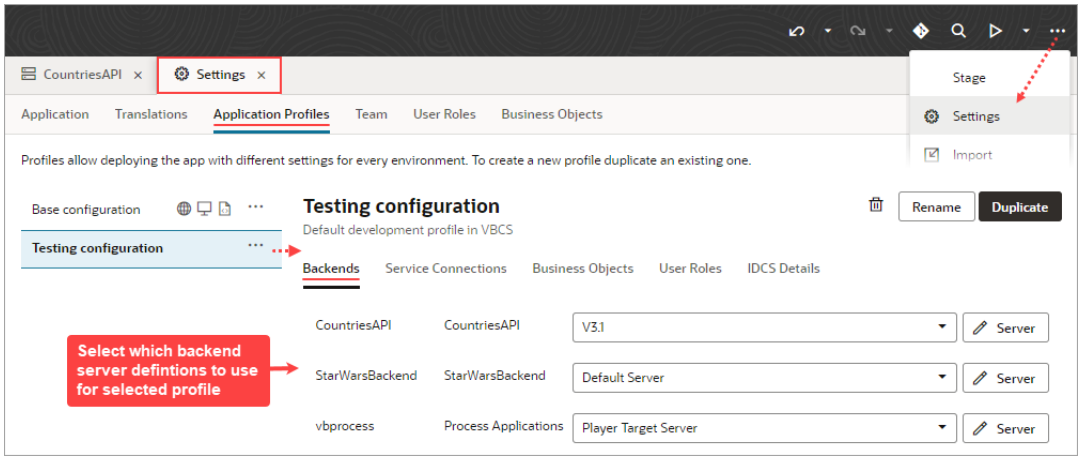
To use different server details (URL, authentication methods, and connection types) for a service during development, staging, and publication:

1. Create the different server definitions that you'll need for the backend. Shown here are the server definitions and their associated application profiles, after the profiles have been created in step 2:



2. Create the application profiles you'll need, and set which backend server definition to use for each.

   To create a new application profile and set which backend server definitions to use for it, go to your visual application's Settings editor. Use the **Duplicate** button to duplicate an existing profile and provide a name and description for the new profile. Select the new profile and use the drop-downs to select which server definitions to use:



3. Switch between the application profiles as needed, to switch between the different backend server definitions.

You can also select authentication methods for logged in users (authenticated users) and non-logged in users (anonymous users). For a description of user roles, see Authentication Roles Versus User Roles.

> **Note:**
>
> Before you connect to a service, an administrator may need to configure the Visual Builder environment settings for backend services, or the settings for the external service or identity provider.

How Does the Identity Propagation Authentication Method Work?

Authentication methods that use identity propagation pass the identity of the logged in user to the service for authentication.

To use identity propagation, the service must be able to understand the IDCS identity token coming from Visual Builder and extract the user (or subject) from it. Visual Builder supports JWT tokens issued by IDCS procured using OAuth 2.0-user assertion flows.

Tokens are a way of encoding the calling user identity into a string according to different specifications, like SAML or the JWT format. For example, if the user is John.Doe, the corresponding JWT token takes the format <*header.body.signature*> and looks like this:

Decoding the body of the token reveals details about the user identity and possibly the resources to which that user is allowed access. The signature part is encrypted by the authority that authenticated the user, and can be easily verified by using the authority's public key. A valid user's identity is encoded into the token so services (namely REST APIs) that receive this token can consider the user as authenticated. This token is usually passed to REST services by passing it as a "*Bearer <token>*" in the Authorization header.

Here are the authentication methods that use identity propagation:

Authentication method	Description
Oracle Cloud Account	Select this method to communicate with Oracle Cloud Applications services. When you select this method, the user must sign in with the credentials of a valid account in the Oracle Identity Cloud Service (IDCS) associated with Visual Builder. The user's identity is converted into a user assertion, then into an IDCS-issued JWT token for the scope that is equivalent to the base URL of the service being called. For example, if the service's URL is `https://fa.oraclecloud.com/myservice`, the token is created for the scope of `https://fa.oraclecloud.com`.
OAuth 2.0 User Assertion	Select this method to call an external system's services that can be represented as a resource app with a particular scope in Oracle Identity Cloud Service (IDCS). This also requires the user to sign with a valid Oracle Identity Cloud Service user account. As with Oracle Cloud Account authentication, the user's identity is first converted into an assertion, then into an IDCS-issued JWT token for the configured scope. The difference is that with this method you can specify your *own* scope, rather than using the service's URL.

Authentication method	Description
Delegate Authentication	This method was previously called Propagate Current User Identity. With this method, the service connection doesn't have its own authentication, but delegates the authentication to the web (or mobile) app that is calling it. The following apply: • A web app can only use the Oracle Cloud Account option for its security (set with web app > Settings > Security). When the service is called from a web app, by default, Oracle Cloud Account authentication applies. The user's identity is converted into an assertion, then into an IDCS-issued JWT token. If the web app has the "Enable implicit grant for Service Connections" option enabled, the resulting JWT token is procured by an OAuth 2.0 implicit grant flow instead of through an assertion. • If the service is called from the Service Tester (from the Service Connection's Test tab), the identity of the user that's logged into Visual Builder is converted into a token and passed to the service. This is similar to what is done with Oracle Cloud Account authentication. Since the actual authentication used at runtime depends on the calling web app's security settings, it is only recommended for use with web apps that use OAuth-based implicit grant authentication.

How Does the Fixed Credentials Authentication Method Work?

An authentication method that uses fixed credentials passes a fixed identity to the service, ignoring the signed-in end user's identity or credentials. All requests to the service use the same app ID for authentication.

For example, if the logged-in user is `abc.xyz`, but the backend is using Basic Auth with the user credentials `def.xyz`, the REST APIs connected by the backend will see only `def.xyz` in their Authorization headers.

Here are the authentication methods that use fixed credentials:

Authentication method	Description
None	Select this for services that don't need authentication and don't accept Authorization headers.

Authentication method	Description
Basic	Select this for services that require a fixed username and password for authentication. The signed-in user's credentials aren't used for authentication. This option uses the Visual Builder authentication proxy, irrespective of the connection type you choose. Due to the limitations of basic authentication, it's recommended that you use this method during development only, and here's why: Suppose you set basic authentication with a particular username and password, and later need to revoke the basic authentication for one specific application. Your only option is to revoke that particular user, which affects all applications that use basic authentication for that user. OAuth-based methods use scopes (with the client identifier and client secret) to offer you better control for managing credentials. **Note:** Basic authentication is not supported for service connections in Visual Builder Gen 2 instances in government realms. If your instance is in a government realm, and your apps use Basic Auth to connect to Oracle Cloud Application services, you must update the apps' service connections to use a supported Oracle Cloud authentication mechanism (for example, OAuth 2.0 Resource Owner Password) before the instance is migrated. If you don't update the service connections before migration, the apps might fail. You can log a Service Request in My Oracle Support for assistance with updating your apps.
OAuth 2.0 Client Credentials	This method is recommended if you want to use a fixed credentials method and the service supports OAuth 2.0 Client Credentials. This method is part of the OAuth 2.0 grant types and is used for application-to-application authentication scenarios where 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 are interpreted as the visual application's client ID and secret, and the token URL is interpreted as IDCS's token URL.
OAuth 2.0 Resource Owner Password Credentials	This method is part of the OAuth 2.0 grant types and is used when you 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 are interpreted as the visual application's client ID and secret, and the token URL is interpreted as IDCS's token URL.

Authentication method	Description
OCI Signature Authentication	This method uses a signature method to create an Application ID flow using a single Oracle Cloud Infrastructure (OCI) user to connect to OCI endpoints. All requests go through a proxy because of the requirement to sign the outgoing message. To use this authentication in Visual Builder, you'll need these user details from the OCI console: • Fingerprint of the public key associated with your OCI account, available on the **Profile** > **User Settings** page. Click **API Keys** and copy the fingerprint value. • User's OCID, available on the **Profile** > **User Settings** page. The OCID is shown under **User Information**; click **Copy** to copy it to your clipboard. • Tenancy's OCID, available on the **Administration** > **Tenancy Details** page. The OCID is shown under **Tenancy Information**; click **Copy** to copy it to your clipboard. • The contents of your certificate's private key in PEM format. For more information, see Request Signatures in the OCI documentation. Once you have the details of the OCI user you want to use to connect to OCI endpoints, set up authentication in Visual Builder as follows: 1. Select **Oracle Cloud Infrastructure API Signature 1.0** as the authentication method. 2. Click **Enter the API Key and Private Key** ( ). 3. Construct the API key in the following format, then copy and paste it as the **API Key**: *tenancy-ocid/user-ocid/fingerprint* 4. Paste the PEM file contents as the **Private Key**. Copy the entire file, starting with `-----BEGIN PUBLIC KEY-----` right up to `-----END PUBLIC KEY-----`. 5. Select **Always use proxy, irrespective of CORS is selected** as the connection type.

The Connection Type indicates how the actual REST API should be connected to: directly from the web app's JavaScript or from the server proxy. You should make this decision based on the CORS support your REST API has.

Note:

When using the fixed credentials authentication method, keep in mind these limitations:

- When using the OAuth 2.0 authentication method, which uses token relay, you're limited by what the browser can send and receive.
- When using a proxy you're limited by the browser as stated above, with the exception of Oracle Cloud Infrastructure API Signature 1.0. For this method, the maximum message body size is two gigabytes (because the proxy needs to cache the entire message to sign it).
- When using a proxy, the REST call will time out after 234 seconds if no data has been sent.

What is CORS?

CORS (Cross-origin request sharing) is a mechanism that restricts a web page on one domain calling APIs hosted on another domain from browser-based JavaScript.

To allow such calls, the server hosting the APIs has to allow the domain of the web page to call by enabling CORS for the web page domain. The rules for CORS are built within the browser and are used to secure the user from arbitrary hosted scripts calling any APIs. See this blog for more details. If the API server has not enabled CORS for the web page, then calling the API via Fetch or AJAX from the browser results in a JavaScript error.

Use an Appropriate Connection Type to Handle CORS for REST Services

Web apps can call external REST services *directly* or through the *Visual Builder proxy*. The value you choose for the connection type and your choice of authentication method when you create or modify a service connection determines which option your app uses.

Direct calls

In this method, the REST service is called directly from the browser JavaScript using the browser's Fetch API. This method can have a performance benefit, as the call is routed directly from the browser to the REST service in question without any intermediary. However, the external REST service that you call must add your app's domain to its CORS allowlist. A direct call can only be made for the following authentication types:

- Oracle Cloud Account
- Delegate Authentication
- OAuth 2.0 (all types)
- None

Calls via proxy

The Visual Builder proxy is a trusted server-side component that calls external REST services hosted on external domains on behalf of your app. One benefit is that you, or a system administrator, don't need to configure CORS settings for the external REST service. However, you incur the cost of an extra network call, as each request and response must first go through the Visual Builder proxy.

All authentication types can be used with the Visual Builder proxy. When you use some authentication types, such as Basic authentication and OCI Signature, authentication is always routed through the Visual Builder proxy. (Note that the proxy times out if the REST API doesn't respond within 234 seconds.)

Connection type options

The type of REST service call to make (either direct or through the Visual Builder proxy) is controlled by the option you choose in the **Connection Type** drop-down menu (located in the dialog where you edit or add a server to a service connection) and includes:

- **Dynamic, the service supports CORS**: Visual Builder decides the best route to connect to the external REST service assuming that the external REST service has enabled CORS for the Visual Builder domain. If the authentication supports direct calls (None, Oracle Cloud Account, Delegate Authentication, OAuth 2.0), the external REST service is called directly from the app.

- **Dynamic, the service does not support CORS**: Visual Builder decides the best route to connect to the external REST service assuming that the external REST service has not enabled CORS for the Visual Builder domain. All calls are routed through the Visual Builder proxy.
- **Always use proxy, irrespective of CORS support**: REST service calls from your apps always go through the Visual Builder proxy.

Review this table to understand what happens when you use different combinations of authentication and connection types:

Application type	Authentication type	Connection type	Result
Any	Any	Always use Proxy	Through Visual Builder proxy
Visual Builder Design time environment	Any	Dynamic – Service doesn't support CORS	Through Visual Builder proxy
Web app running on browser	Any	Dynamic – Service doesn't support CORS	Through Visual Builder proxy
Progressive Web App	Any	Dynamic – Service doesn't support CORS	Through Visual Builder proxy
Any	All except basic authentication and OCI Infrastructure API Signature 1.0	Dynamic – Service supports CORS	Direct
Any	Basic authentication or OCI Infrastructure API Signature 1.0	Dynamic – Service supports CORS	Through Visual Builder proxy

Here's an example of how different choices work when your application calls Sales and Service (CRM) REST APIs from an Oracle Cloud Applications instance. For this scenario, let's say your Visual Builder instance is hosted on `vb.oracle.com` and the crmRestApi is hosted on `fa.oraclecloud.com`.

If you chose **Dynamic, the service supports CORS** and the selected Authentication Type supports direct calls, the direct call goes like this: `vb.oracle.com (browser) → fa.oraclecloud.com`.

If you chose **Dynamic, the service does not support CORS** or **Always use the Proxy**, the call is routed through the proxy and goes like this: `vb.oracle.com (browser) → vb.oracle.com proxy → fa.oraclecloud.com`.

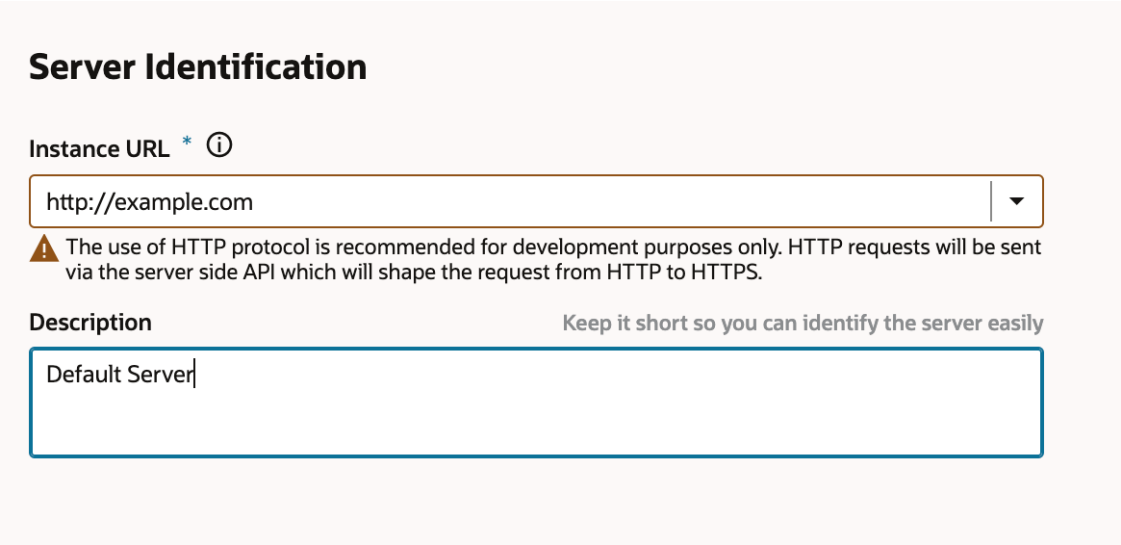
It's recommended that you use the **Dynamic** options, where Visual Builder decides the best route based on whether the external service supports CORS or not for this particular domain. However, you have the flexibility of choosing **Always use the Proxy** to route all requests via the proxy.

If you don't want to use the Visual Builder proxy, you must enable CORS in Oracle Cloud Applications for your particular domain (`vb.oracle.com` in our example) by adding the `Access-Control-Allow-Origin` CORS header for the CORS profile option. To view the profile option, go to the **Setup and Maintenance** work area and use the **Manage Applications Core Administrator Profile Values** task in the Application Extensions functional area. See Configure Cross-Origin Resource Sharing in *REST API for Common Features in Oracle Fusion Cloud Applications*.

Work with HTTP-based Endpoints

Visual Builder uses HTTPS for its applications, and we recommend also using HTTPS for all service connections to external endpoints.

If you use an HTTP-based URL to create a service connection for development purposes, it will always be routed via the VB Proxy irrespective of the connection type and then to the actual HTTP service. This is because visual applications run on HTTPS and are not allowed to call an HTTP endpoint directly, that is using browser JavaScript. From the VB Proxy to the HTTP endpoint, the call is routed through HTTP. You will get a warning in the Server tab (shown below) when you provide an instance URL that uses HTTP.



Use of HTTP is discouraged because the credentials that access the HTTP service are visible over the network. These credentials are not encrypted, as in case of HTTPS). This makes HTTP services vulnerable.

Allow Anonymous Access to the Service Data

Select the **Allow anonymous access to the Service Connection Infrastructure** checkbox in the Edit Server dialog 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:**
>
> If you allow anonymous access to a backend and its service connections by specifying credentials for anonymous access, be aware that those backend systems will be accessible through a public URL without any authentication. If you need anonymous access for a backend or service connection, consider:
>
> - Creating a backend specifically for the service connections to which you want to allow anonymous access (allowing anonymous access is inherited by a backend's service connections).
> - Adding only the minimal credentials required to access just those service connections (for example, read-only credentials).

OAuth and JWT Token Validity

The validity of an OAuth token for a particular service connection is determined by the system that issues the token. In the case of JWT tokens, the expiry date and time are encoded within the token itself. Regardless of the token type and the issuing system, the VB (service) infrastructure will ensure that the token is reissued when the current one expires, provided the user is still logged into IDCS and the calling REST API complies with the OAuth specification. If the user is no longer logged into IDCS, any attempts to reissue a token will redirect the user to the login page.

Create a Backend

If your environment does not include Oracle-provided backends such as Integration Applications, Oracle Cloud Applications, or Process Automation, you can add these backends to create an application-level catalog for your visual application. You can create only one instance of each backend in your visual application.

The high-level steps to add a backend are the same for each backend and involve providing the URL of the instance that hosts the backend service, authentication method to access the instance, and so on.

Let's say you want to add the Oracle Cloud Applications backend: You supply the URL of your Oracle Cloud Applications instance (for example, `https://my-oracle-cloud-app-instance.example.com`). This will automatically discover the `interfaceCatalogs` endpoint (typically *`Oracle Cloud Applications Base URL`*`/helpPortalApi/otherResources/latest/interfaceCatalogs`) of your Oracle Cloud Applications instance and retrieve the list of available services from the most recent ADF Describe.

To create a backend that connects you to the Integration Applications, Oracle Cloud Applications, or Process Automation catalog:

1. Click **Services** in the Navigator.
2. In the Services pane, click **+** sign and select **Backend**.
3. In the Create Backend wizard, select the backend you want to create:
   - To register an Oracle Cloud Applications instance as a backend, click **Oracle Cloud Applications Instance**.
   - To register an Integrations instance as a backend, click **Integrations**.
   - To register an Oracle Process Automation instance as a backend, click **Oracle Process Automation**.

4. Enter the instance URL and other information, such as authentication details and headers, that your visual application requires to successfully connect to the backend.
5. Click **Create**.

The newly registered backend shows up in the Backends tab.
It is possible to edit all the server details (instance URL, authentication method, headers, connection type, and so on) after the backend is registered. You can also specify more than one server that provides access to the backend by adding additional servers that host the backend. See Edit a Backend.

To create a service connection to your backend, click **+ Service Connection**. See Create Service Connections from the Oracle Cloud Applications or Integration Applications Catalog.

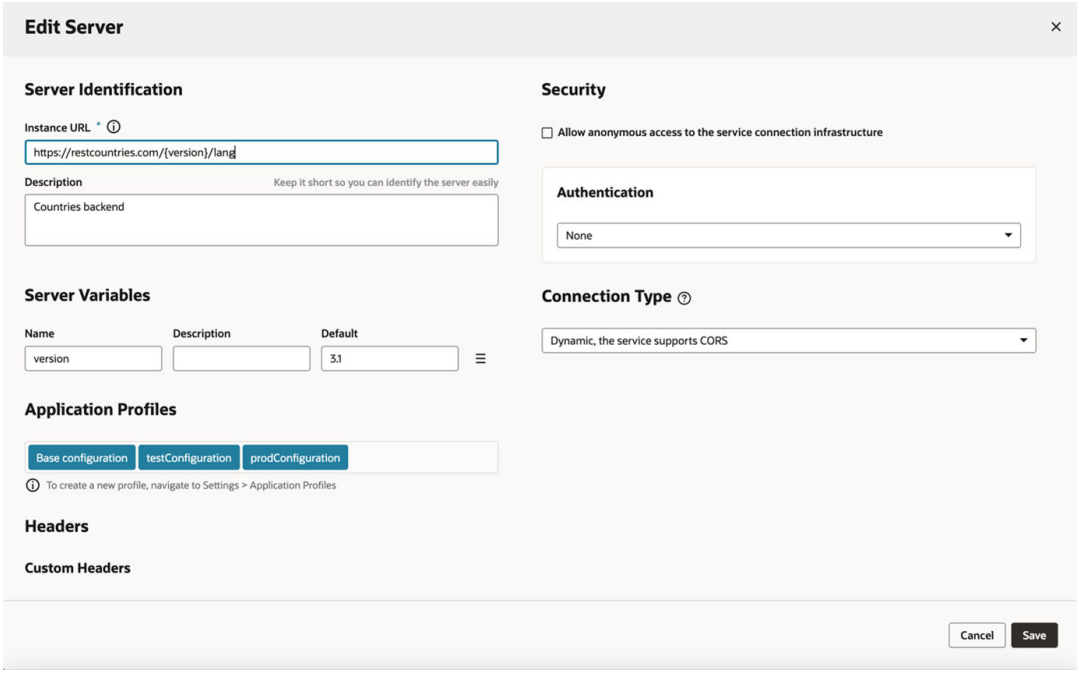
Add Server Variables for Backends

When creating or editing a backend, you can add server variables to the instance URL on the Servers tab.

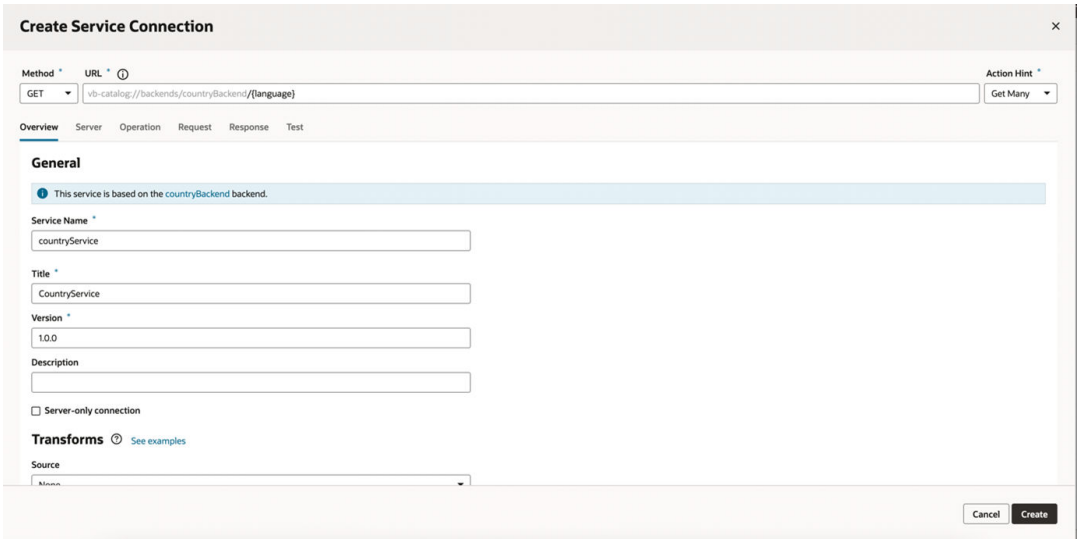
When you enter a valid URI template expression, such as `{version}`, in an instance URL, a server variable will be created automatically and displayed in the Server editor's **Server Variables** section. You can also create, edit, or reorder a list of variables. A default value must be set for each server variable.

Let's take a look at how this works, using `https://restcountries.com/v3.1/lang/german` as an example:

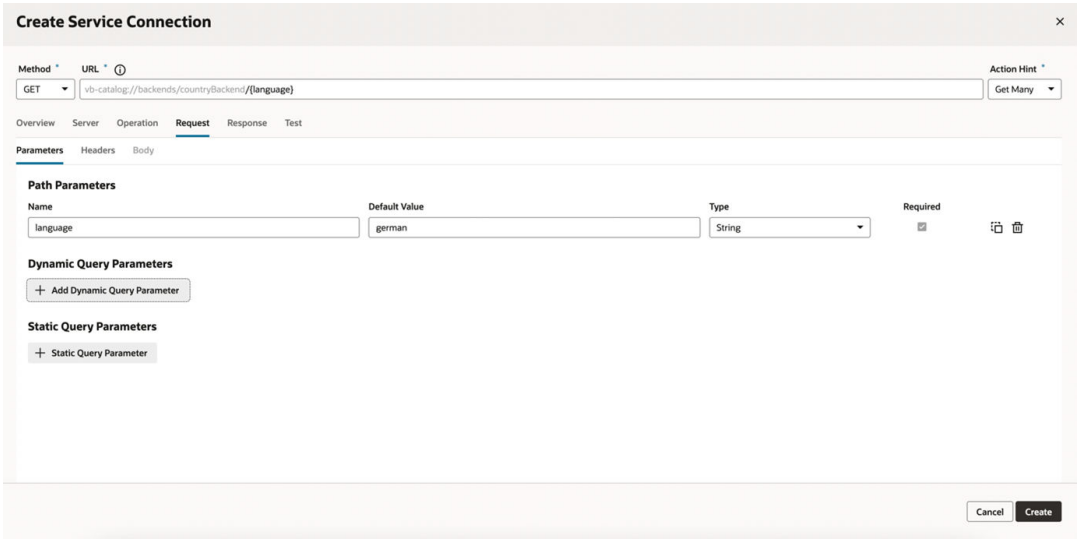
1. The server instance URL can be represented as `https://restcountries.com/{version}/lang`.
   Here, there's just one server variable, `{version}`, which has a default value of `v3.1`, as shown in the server of a backend called **countryBackend**:



2. The endpoint is `/{language}`.
   We define the service connection on the **countryBackend** by defining the endpoint and adding `/{language}` to the URL path.



In this example, `language` is a path parameter, because it's a part of the endpoint path. It can't be a server variable, in this example, because it's outside the instance URL. Its default value is `german` and its type is string.



3. The full URL becomes `https://restcountries.com/{version}/lang/{language}`.
   After substituting (`v3.1` for the server variable `{version}` and `german` for the `language` path parameter), this represents the instance URL we started with, `https://restcountries.com/v3.1/lang/german`.

Here's how you can add or change a variable (or multiple variables) in a server URL:

1. Open the **Servers** tab of the backend.
2. Click **Edit** to the right of the server instance to which you want to add a new server variable (or modify an existing one) to the instance URL.

   The Edit Server dialog is displayed.
3. In the Instance URL field, add a variable where one is needed.

For example, you may want to replace a version number in the URL, such as "1.2" with the variable {version}. After you do this, you'll see the variable you added to the URL displayed in **Server Variables**, below the instance URL's **Description** field. The variable you added is shown in the **Name** field.

4. Enter a description in the **Description** field and set the default value in the **Default** field.

   Click **Done** if you are finished or click **+ Add Value** to add another value. To create a list with multiple values, click **Menu** ☰ and select **Create Value List**. Then, after adding values, use the ⌃ or ⌄ controls to set the default value or reorder the list of values. The default value will be the one at the top of the list. You can use **Delete** 🗑 to discard values you don't need.

5. Add another variable in the URL, if needed.

   Note that you must set a value for each variable you define. If you don't, you'll see a message indicating that a value is required. Enter the default value and click **Done**.

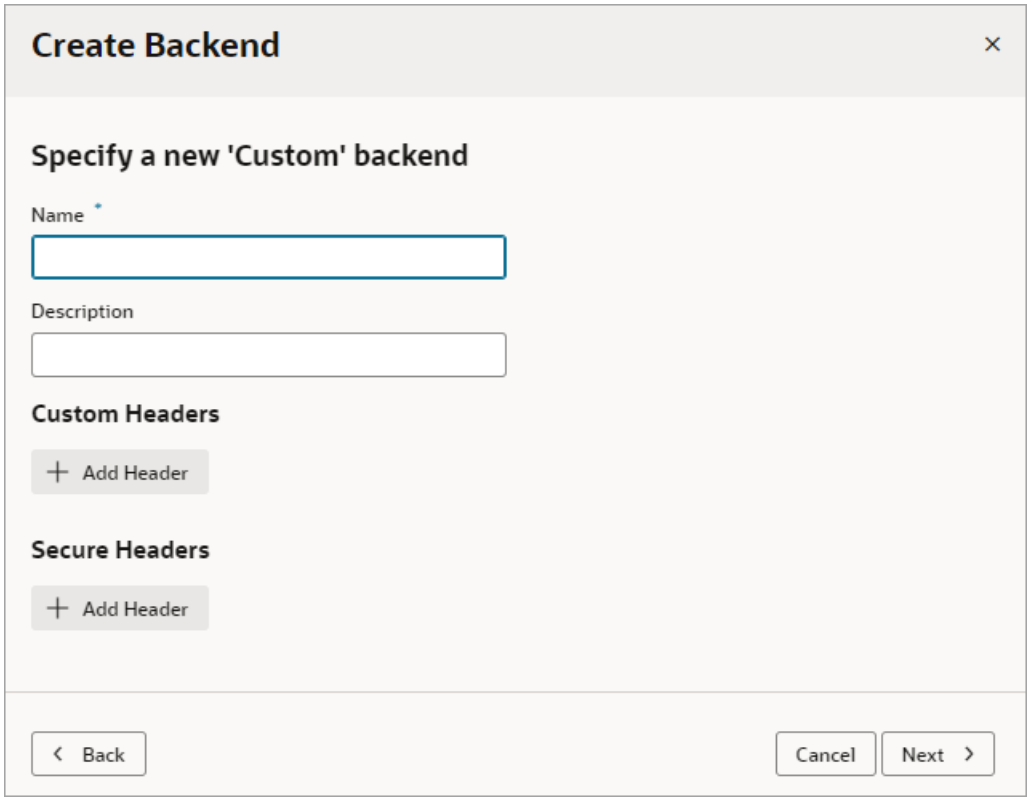
6. Click **Save**.

Create a Custom Backend

You can create your own backend to map to a custom server other than the built-in Integration Applications, Process Automation, and Oracle Cloud Application backends.

A custom backend lets you access a service when you know its URL. You can create a **custom** backend with a free-form URL, or create a **custom ADF backend** when you know the Describe URL that points to an ADF Describe service.
To create a custom backend:

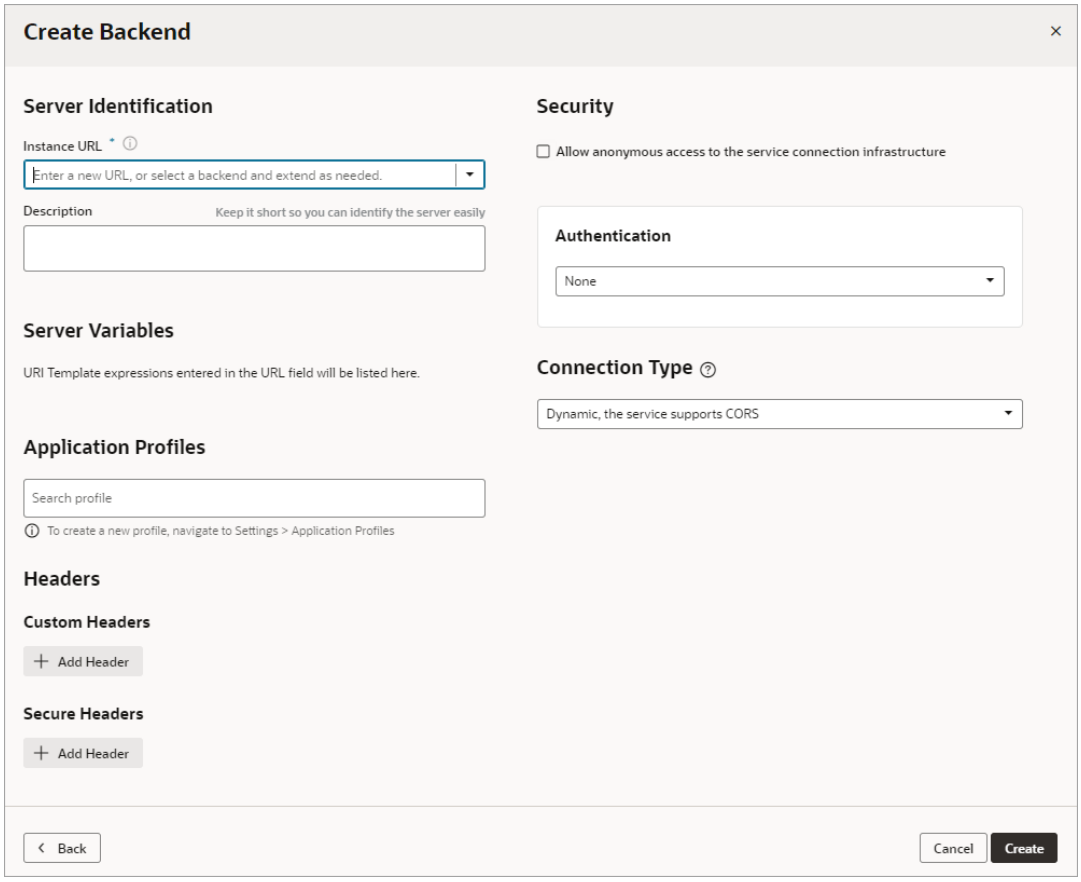
1. Click **Services** in the Navigator.
2. In the Services pane, click **+** sign and select **Backend**.
3. In the Create Backend wizard, select the type of backend you want to create:
   - To create a backend with a free-form URL, click **Custom**.
   - To create a backend with the Describe URL of an ADF service, click **Custom ADF Describe**. Use this option only when your custom ADF Describe endpoint does not have any child backends.
4. Enter a name and description for the custom backend. Your description will be the backend's display name in your application.



5. Optionally, add static headers:
   a. To add a static header that is passed from the browser to the service (for example, a REST-Framework-Version header), click **Add Header** under Custom Headers. Enter a name for the header and its value, then click **OK**.

      Custom headers become available to you from the browser's Developer Tools console.
   b. To add a static header that isn't passed from the browser to the service and is automatically inserted only at the server, click **Add Header** under Secure Headers. Click the **Create Secure Header** icon, enter a name and value for the header, then click **Save**.

      Secure headers are useful when you want to apply a credential to a header and never want it to be available on the browser.
6. Click **Next** to display the second page in the wizard:



7. Enter the instance URL and other information that your visual app requires to connect to the custom backend.

> **Note:**
>
> If you're using an IP address instead of a proper DNS-based URL in a production environment, you're probably using self-signed certificates. Be aware that certificates based on IP addresses are less secure and more difficult to maintain than DNS-based URLs. To avoid potential security issues, self-signed certificates should never be used in production environments. For this reason, an audit warning is displayed whenever an IP address-based service connection is used.

8. If you need more information about the options in the Authentication drop-down, see Set the Backend's Authentication Method and Connection Type.
9. To add server variables, see Add Server Variables for Backends
10. Click **Create**.

A new custom backend displays in the Backends tab on the Services pane. Click the newly created custom backend to view and edit its details. See Edit a Backend.

Now that your custom backend is registered, you can click **+ Service Connection** to create a service connection to your backend, either by providing a service specification document or by pointing to the URL of a service endpoint. See Create a Service Connection from a Service Specification or Create a Service Connection from an Endpoint.

Create a Child Backend

You can create child backends to extend the functionality provided by custom backends that have been registered in your extension, or by the Oracle Cloud Applications backend.

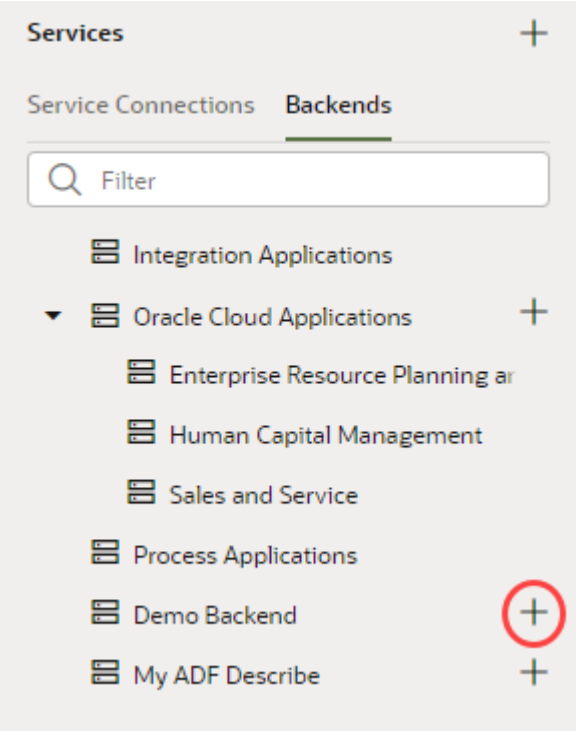
A child backend inherits the parent backend's definition, which you can override as required. Its server URL is derived from the top-level backend, with `vb-catalog://backends/` as the base URL. For example, the Sales and Service backend, which is a child of the Oracle Cloud Applications backend, has the server URL `vb-catalog://backends/fa/crmRestApi/resources`, where `fa` represents the parent backend.

Child backends can be created only for custom backends, or for Oracle Cloud Applications.

Create a Child Backend for a Top-Level Custom Backend

To create a child backend for a custom backend:

1. Open the **Backends** tab from the Services tab in the Navigator.
2. Click the **+** sign next to the intended parent backend:



3. Select **Custom ADF Describe** to create a backend with an ADF Describe URL. For a backend that doesn't have a Describe URL, select **Custom**.
4. Enter a name and description for the child backend. Optionally, click **+ Add Header** to add custom static headers. To add a static header that won't be available in the browser and will only be inserted at the server, click **+ Add Header** under Secure Headers. Secure headers are useful to apply a credential to a header without it being available in the browser.
5. Click **Next**.

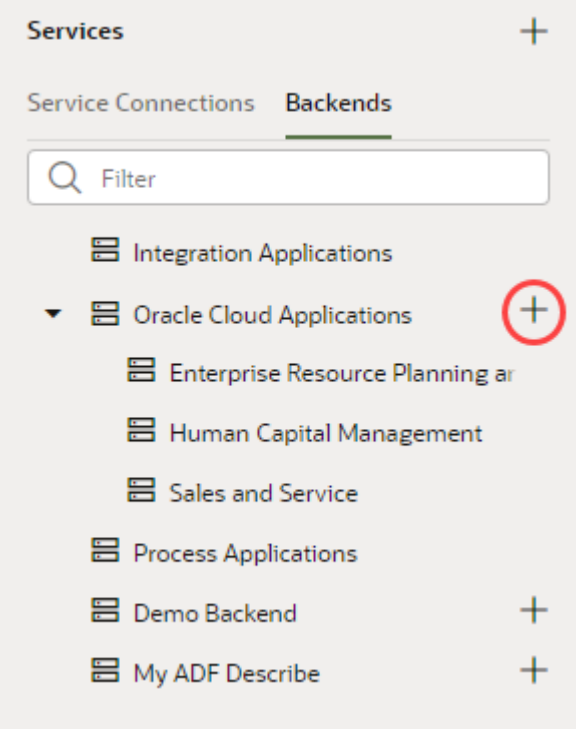
6. Enter the instance URL for the child backend, beginning with the slash (/) that follows the grayed out portion of the inherited parent URL.
   You can click the **Info** icon to see the complete URL that the backend resolves to.
7. Enter other settings, as needed.
8. Click **Create**.

Create a Child Backend for the Oracle Cloud Applications Backend

The process for creating a child backend based on the Oracle Cloud Applications backend is slightly different, as the parent and the child backends are in different extensions.

To create a child backend for the Oracle Cloud Applications backend:

1. Open the **Backends** tab from the Services tab in the Navigator.
2. Click the **+ sign** for the top-level Oracle Cloud Applications backend:



3. Select **Custom ADF Describe** to create a backend with an ADF Describe URL. For a backend that doesn't have a Describe URL, select **Custom**.
4. Enter a name and description for the child backend. Optionally, to add custom static headers, click **+ Add Header**. To add a static header that won't be available in the browser and will only be inserted at the server, click **+ Add Header** under Secure Headers. Secure headers are useful when you want to apply a credential to a header without it being available in the browser.
5. Click **Next**.
6. For **Instance URL**, enter the rest of the path for your new backend after the slash (/) that follows the grayed out portion of the parent.
   Click the **Info** icon to see the complete URL that the backend resolves to.
7. Enter other settings, as needed.

8. Click **Create**.

Edit a Backend

You can edit any backend that you've added, and for the backends that were automatically added to your visual application upon its creation, you can override their configurations set at the tenant level for your Visual Builder instance.

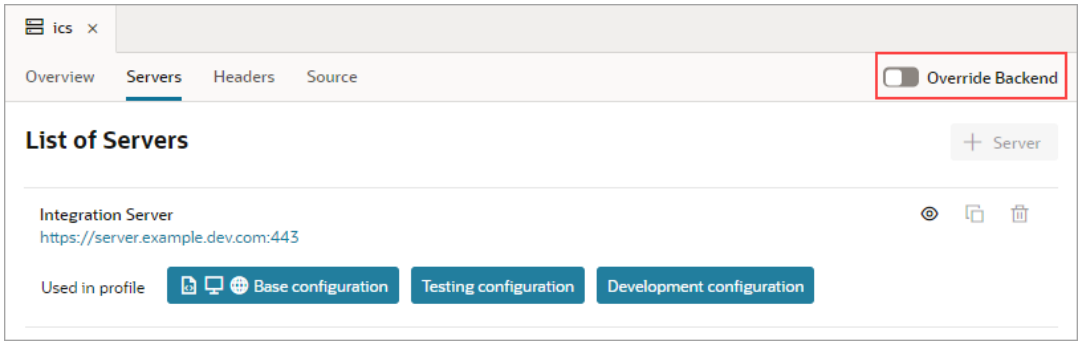
When you override a provided backend's configuration, you are essentially adding a backend with the same ID from the environment catalog to the application catalog and customizing some detail (for example, the authentication method specified for a backend server) just for that particular application.

To edit and customize a backend service:

1. Open the **Backends** tab from the Services tab in the Navigator.
2. Select the backend you want to modify, then click the **Servers** tab.

> **Tip:**
>
> Click the **View Information** icon ( ) to view backend details, such as server and authentication inherited from your Visual Builder instance's tenant-level settings.

3. To edit an Oracle cloud services backend, you'll need to click **Override Backend** to modify the settings specified by your administrator:



4. Edit the backend service's details as required:
   - Use the **Overview** tab to change the description, if desired, or to change the transforms attached to the backend (see Add Transforms for more information).
   - Use the **Servers** tab to add a new server that hosts the backend service, or modify an existing server. In both cases, a dialog appears where you can identify the server, specify an application profile that will use the server, add headers, and configure security options. These options are the same as those you configure when you define a service connection.
   - Use the **Headers** tab to add static headers.

     Under Custom Headers, you can add headers that are passed from the browser to the service (for example, a REST-Framework-Version header). Under Secure Headers, you can add headers that are inserted only at the server end (for example, a credential that you never want to be available on the browser).

- Use the **Source** tab to view the contents of the `services/catalog.json` at the application level, and to edit the file if needed.
  You can also access this file under `services` in the Navigator's Source View tab.

Add Transforms

Transforms are JavaScript functions that transform the format of data and parameters for REST requests and the format of data from REST responses. Request transforms are typically for sorting, filtering, and so on, while response transforms are typically for formatting data and retrieving paging metadata.

Transforms perform these actions:

- Filtering, to specify the data to be returned and displayed.
- Sorting, to sort items returned from a collection resource.
- Pagination, to limit the number of records that are displayed on a page.

A transform can be a:

- Require.js module, such as vb/BusinessObjectTransforms (default transform for an Oracle Cloud Applications backend)
- File path that's relative to the service connection, such as `./transforms.js`
- URL, such as `https://cdn.oracle.com/static/vb/businessobjecttransforms`

The Oracle Cloud Applications backend has built-in business object REST API transforms (`vb/BusinessObjectTransforms`) that are applied by default. When you create a related service connection using the **Select from Catalog** option of the Create a Service Connection wizard, the service connection and its endpoints inherit the backend's built-in transforms. Similarly, ADF Describe backends have pre-defined transforms that are applied by default.

Transforms can be applied at the *service level* or the *instance level*:

- **Service Level Transforms**: Applied to a backend, child backend, service connection, or endpoint.
- **Instance Level Transforms**: Applied to a Service Data Provider (SDP) or a Call REST action as individual functions.

For the sake of consolidation and simplification, it's best to use only service level transforms, with one exception. Visual Builder business objects are pre-configured with out-of-the-box transforms (vb/BusinessObjectTransforms) and can't be changed. If you need to change a transform for a business object, you can use instance level transforms.

You can find more details about adding transforms to an SDP or a Call REST action here:

- SDP Request Transformation Functions
- SDP Response Transformation Functions
- Call REST Action

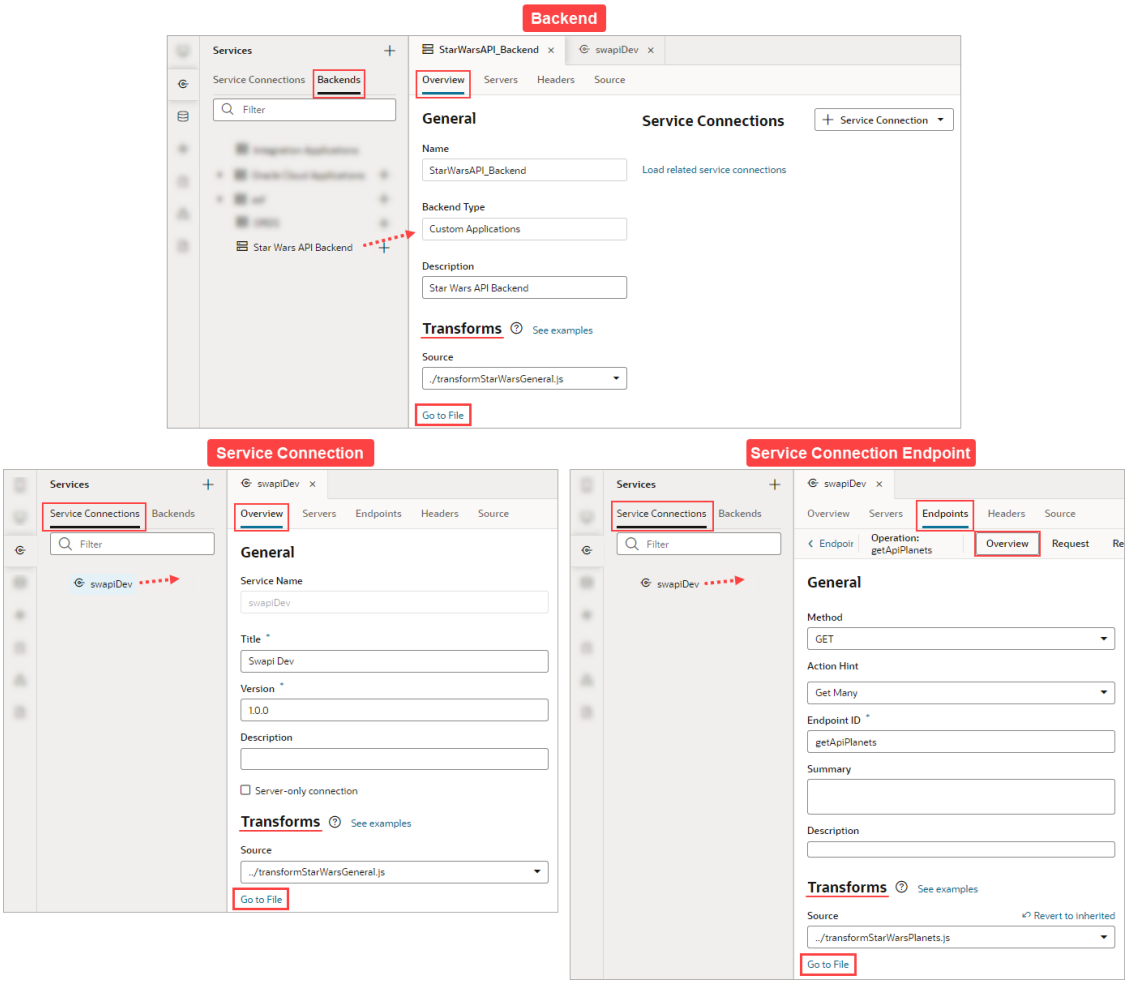
Service Level Transforms

Service level transforms are typically applied to a backend so that they can be inherited by the backend's service connections and by the service connection's endpoints. However, if you need to differentiate between the REST resources or between the endpoints (GET vs POST) of

a system, for the service connection or its endpoints, you can choose a different `.js` transforms file or create a new one.

Within the hierarchy of service level transforms, an endpoint is first checked for a transform file, then the service connection, then the child backend (if there is one), and lastly, the backend. The first transform file that is found is used; any at the upper levels are ignored.

To see a backend, service connection, or endpoint's provided transform functions, navigate to its Overview tab, then click the **Go to File** link in the Transforms section. To create new transform functions, click the **Source** drop-down list and select **Create**.

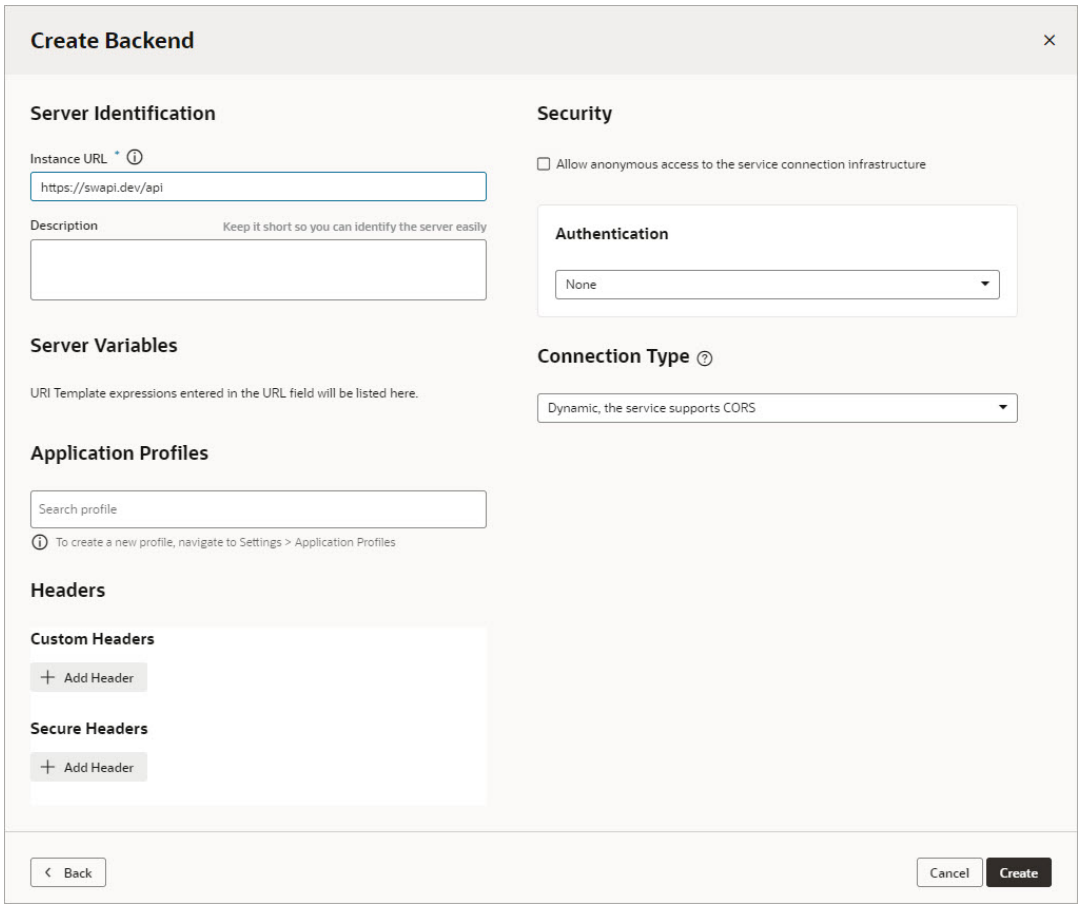


Example of Creating Service Level Transforms

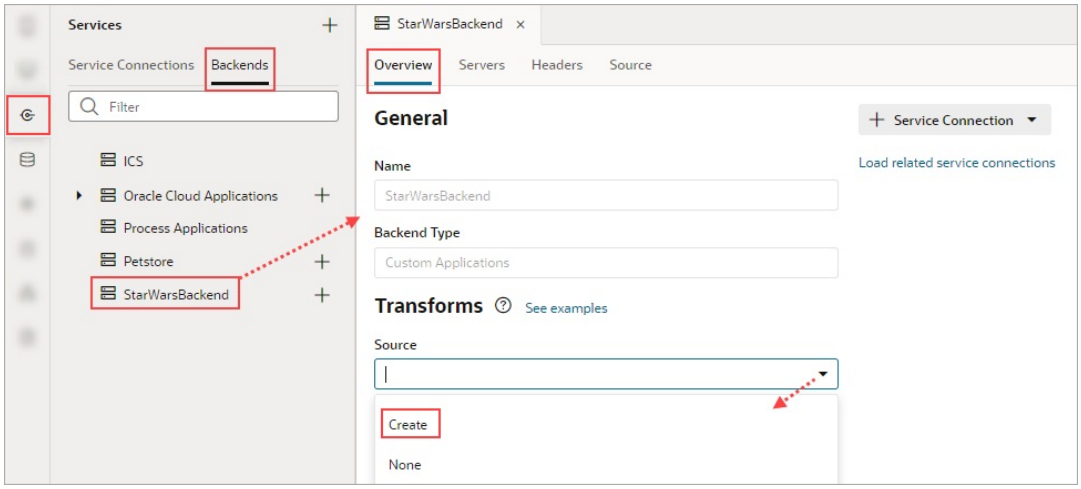
These steps show how to create a backend for service connections and how to create the file with the template transform functions for your new backend and its service connection endpoints:

1. Open the **Services** panel, select the **Backends** tab, click the Add (+) icon and select **Backend**.
2. In the Create Backend window, select **Custom** to create a custom backend.
3. Provide a name for your custom backend, and optionally, enter its description.

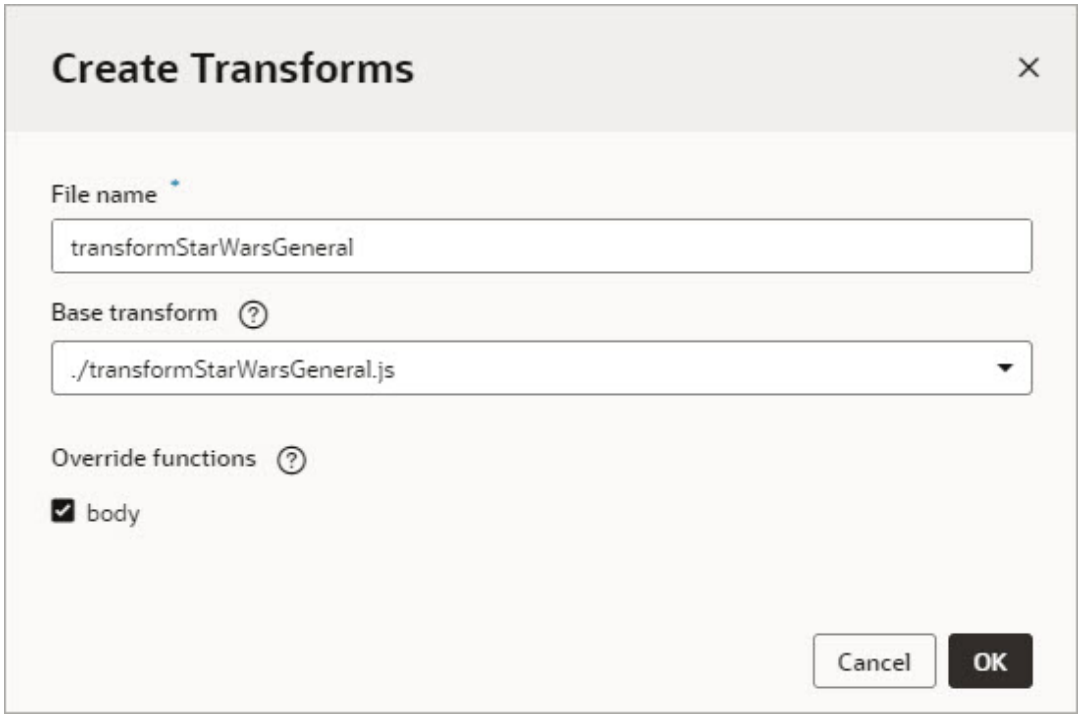
4. Enter the instance URL and any other required details. Later, when creating a service connection for the backend, this URL is used as the base for defining the endpoint. Click **Create** to create the backend.



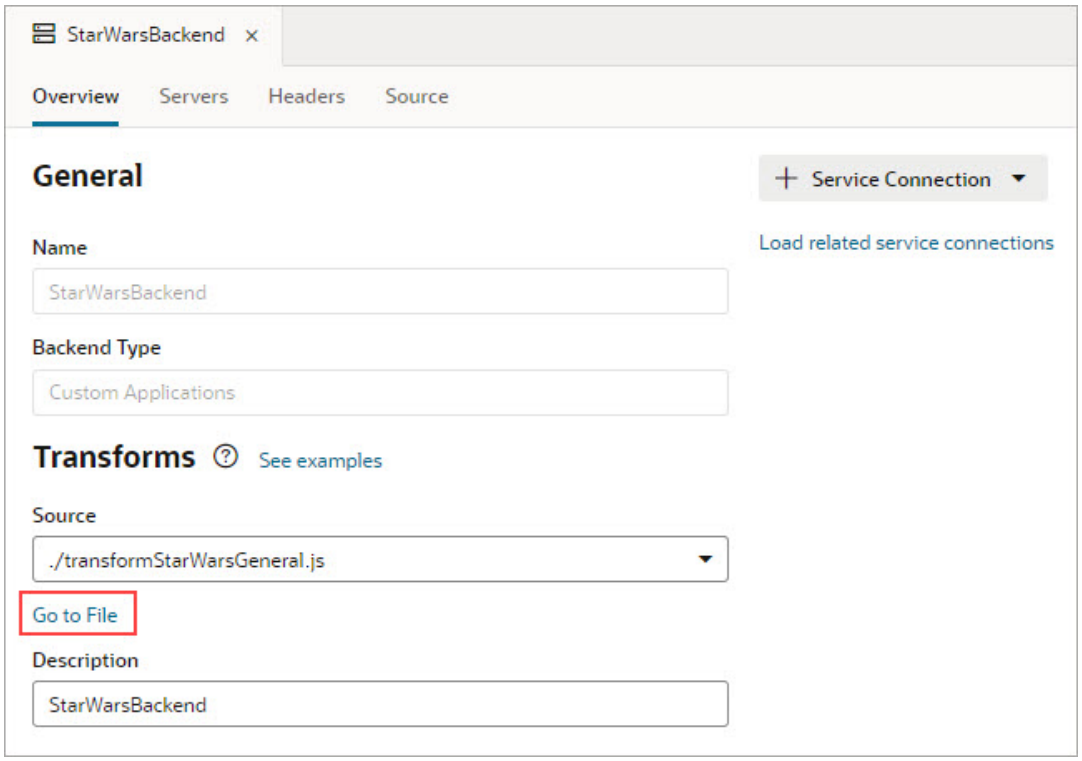
5. You can now create the file with the template transform functions for your new backend. On the **Overview** tab, in the **Transforms** section, click the **Source** field's down-arrow and select **Create**:



6. Provide the file name for the transforms file:



7. For the **Base transform** dropdown, select the file to supply the base code for all of your transform functions. For ADF REST APIs, this file is always chosen for you.
8. In the **Override functions** select the function(s) you want to add to the transforms file and click **OK**. Functions that you don't select won't be added to the transforms file.
9. To open the transforms file, click **Go to File** under the **Source** field:



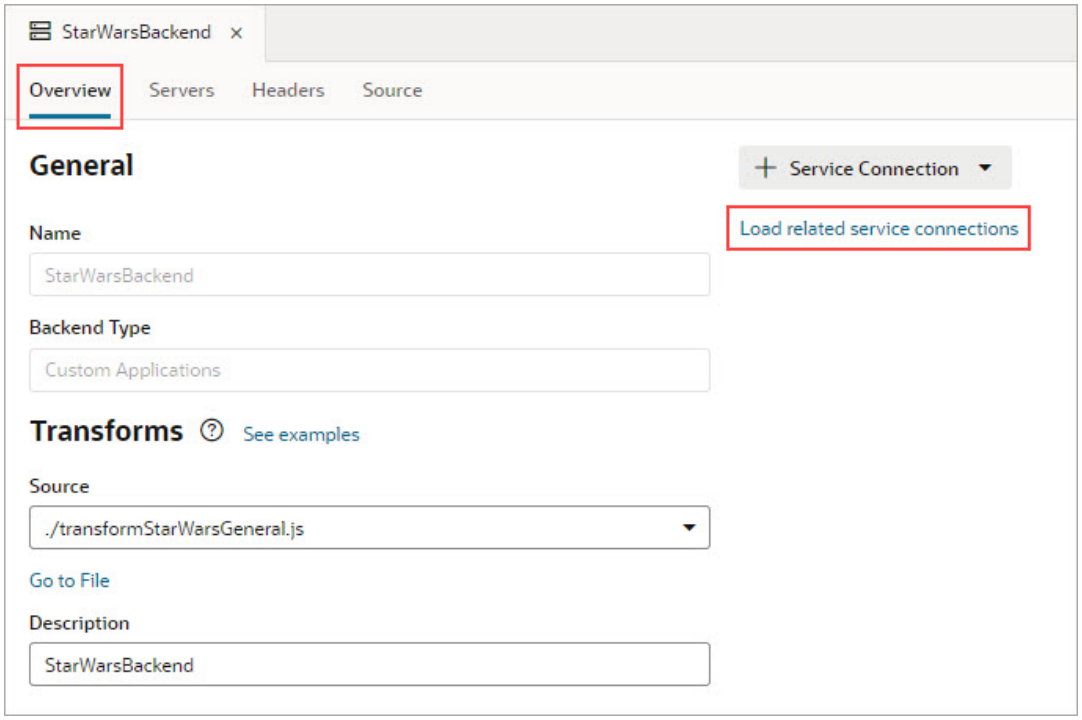
10. Override the relevant transform functions as required. The provided built-in transform functions are: sort, filter, paginate, select, query (commonly used with GET collection endpoints), body (commonly used with POST and PATCH endpoints), and fetchByKeys. The `transformsContext` input parameter is a context object that is passed to every transform function by the associated SDP, so that you can retrieve and store contextual information for the current REST call. SDP variables have a transformsContext property that is passed to the transform functions. For more about an SDP's `transformsContext` property, see Oracle Visual Builder Page Model Reference - transformsContext SDP Property.

```
class Response {
    /**
     * @typedef {Object} PaginateResponse
     * @property {number} totalSize optional what the totalSize of the result is (the total count of the records in
     * the service endpoint).
     * @property {boolean} hasMore usually required, the paginate response transform function is relied upon to
     * inform the ServiceDataProvider when to stop requesting to fetch more data. Indicates whether there are more
     * records to fetch
     * @property {String} 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.
     */

    /**
     * paginate is called with the response so this function can process it and return an object with
     * properties set.
     * @param {object} result
     * @param {TransformsContext} transformscontext
     * @return {PaginateResponse}
     */
    /*paginate(result, transformscontext) {

        const tr = {};

        if (result && result.body) {
            const cb = result.body;
            if (cb.totalCount) {
                tr.totalSize = cb.totalCount;
            }
            if (cb.totalCount >= 0) {
                tr.hasMore = !!cb.hasMore;
            } else {
                tr.hasMore = false;
            }
        }
        return tr;
    }*/

    /**
     * body is called last, after all the other response transforms have been called. It is a hook for authors
     * to transform the response body or build an entirely new one.
     * @param {object} result
     * @param {TransformsContext} transformscontext
     * @return {object}
     */
    body(result) {
        // TODO: Replace example code
        let tr = {};
        if (result.body) {
            tr = result.body;
        }

        // as a example store some random aggregation data
        tr.aggregation = { example: 4 };

        return tr;
    }
}
```

Here's an example for the `body(result)` transform function, which adds text to all of the `mass` and `height` fields:

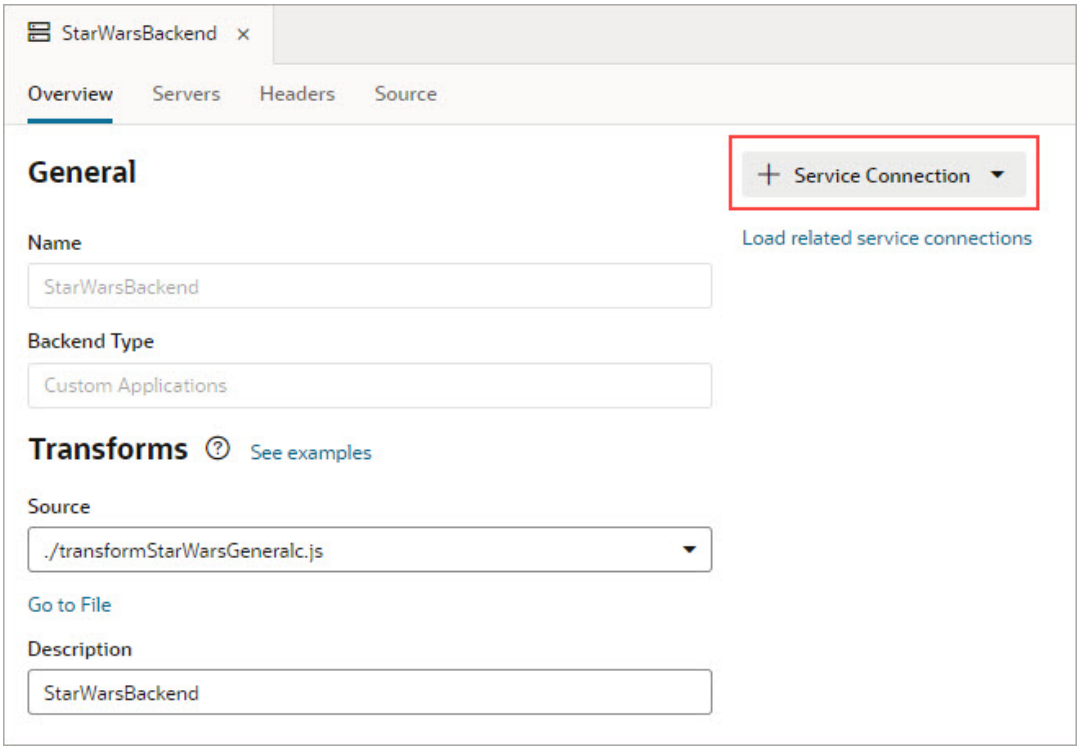
```
body(result) {
    let tr = {};
    if (result.body) {
        tr = result.body.results;
        for (let i = 0; i < tr.length; i++) {
          tr[i].mass = tr[i].mass + " kilos (kg)";
          tr[i].height = tr[i].height + " cm";
        }
    }
   // return tr;
}
```

The created transforms file is inherited by its related service connections, which are created by using the **+ Service Connection** button on the **Overview** tab. To view the related service connections, click the **Load related service connections** link:

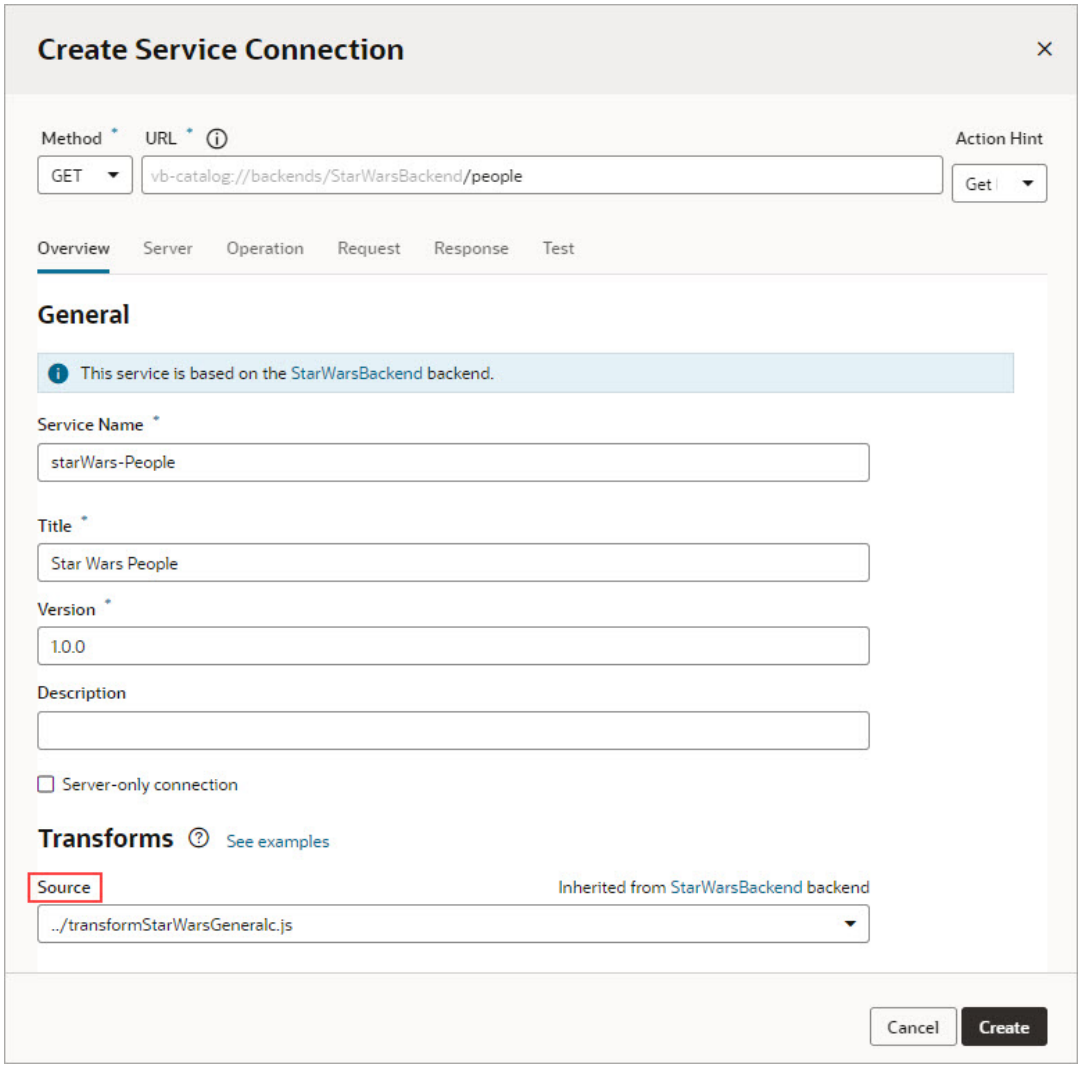


For details about creating a service connection from an endpoint, including how to get an example of the body of the response, see Create a Service Connection from an Endpoint.

11. To add a related service connection for your backend, click the **+ Service Connection** button:



12. Complete the **URL** for the service connection endpoint and provide the required details.
13. To create a transform file for the endpoint, click the **Source** field's down-arrow and select **Create**:



14. To open the transforms file, click **Go to File** under the **Source** field.

Examples for Endpoints that Are Not from Oracle Cloud Applications

Here are examples of request and response transform functions for endpoints that aren't from Oracle Cloud Applications:

Request
A request transform function is called right before a request is made to the server/endpoint. It provides a chance for page authors to transform the options (filter, paginate, sort, and so on) and build the final request configuration. Here are some samples of how to write `filter`, `paginate`, and `sort` transform functions:

filter

```
/**
 * 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;
};
```

paginate

```
// paginate Transform Function
// 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;
};
```

sort

```
/**
 * Sort Transform Function Implementation
 * @param configuration
 * @param options the JSON payload that defines the sortCriteria
 * @param context an object to store/retrieve any contextual information
for the
 *  current request lifecycle.
 * @returns {object} configuration object. the url looks like ?
orderBy=foo:asc
 */
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;
};
```

Response

A response transform function is called right after a request returns successfully. Here is a sample of how to write a `paginate` transform function:

```
// paginate() Response Transform Function
PageModule.prototype.paginateResponse = function (result, context) {
  const ps = {}; const tr = {};

  if (result.body) {
    const rb = result.body;
    if (rb.totalCount) {
      tr.totalSize = rb.totalCount;
    }
```

```
    if (rb.totalCount > 0) {
      tr.hasMore = !!rb.hasMore;
    } else {
      tr.hasMore = false;
    }
  }
  return tr;
};
```

Manage Service Connections

To access external REST APIs in your visual application, you create connections to the services that provide access to these API endpoints.

What Are Service Connections?

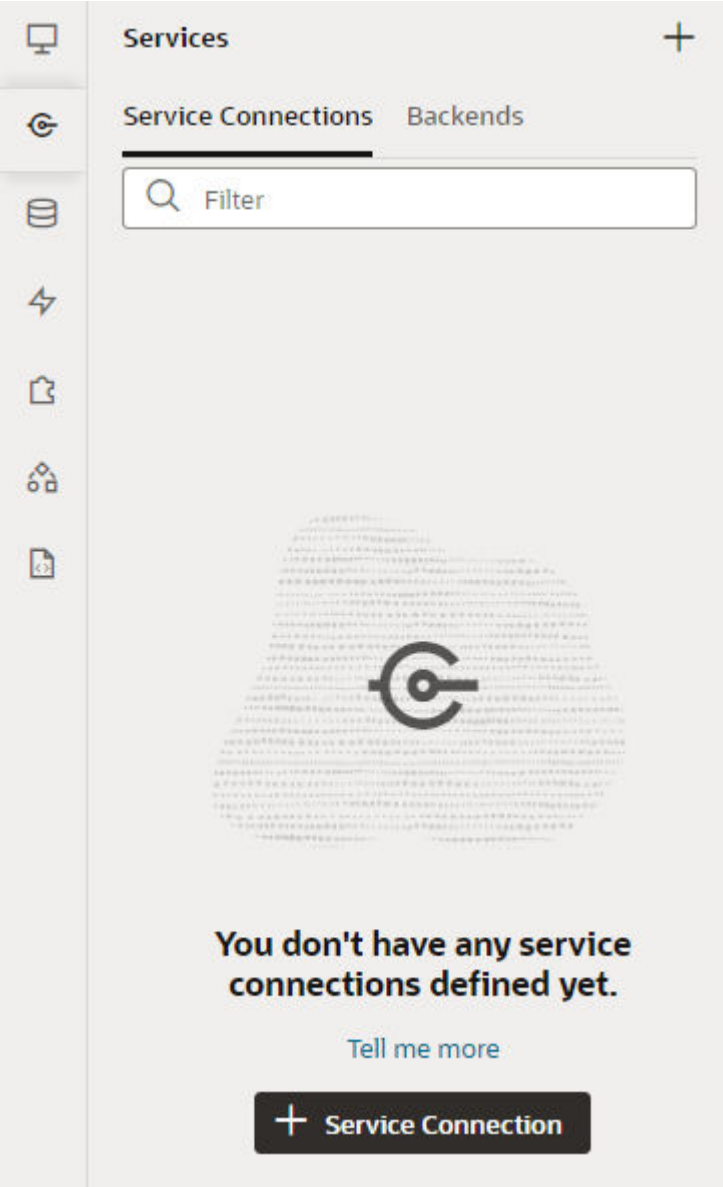
To work with an external service's REST API, Visual Builder needs basic information about that service. A service connection provides this information by describing the connection to the service, including connection details, properties, and the REST endpoints provided by the service that you want to use in your application.

You can create a service connection in the following ways:

- By selecting a service from a **catalog** of preconfigured Oracle SaaS/PaaS REST services
- By providing a Swagger or ADF-Describe **specification** that describes the external service
- By providing the location of a REST **endpoint** for the external service

A Visual Builder catalog of predefined services includes backends, such as Oracle Cloud Applications and Integration Applications. These backends expose REST APIs that your visual application can consume right out of the box. You can also create custom backends to access services that aren't listed in this catalog, either by providing the service specification or by providing the URL of a REST endpoint of the service.

Use the Services tab in the Navigator to add new service connections and manage your existing ones. You can also manage your application’s catalog of backends here:



Service Connections: Static Versus Dynamic

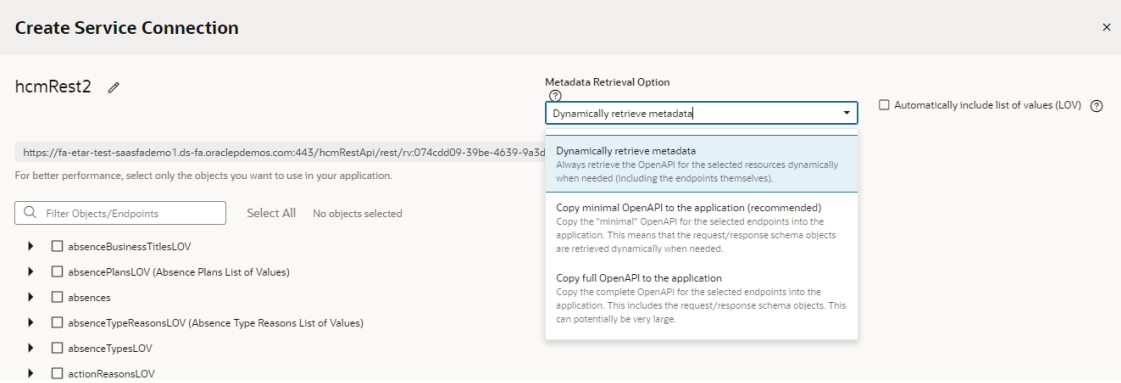
Service connections in Visual Builder are defined by the service's OpenAPI metadata, which describes the available endpoints and details required to connect to the service. How you want your connection to retrieve this service metadata—either statically or dynamically—is entirely up to you. Both options have their advantages and disadvantages.

Let's say a service's OpenAPI definition is located at `https://service.com/openapidef`. You can set up your service connection to retrieve this definition in one of two ways:

- Copy the service definition from `https://service.com/openapidef` at the time of development and save it to the visual application's sources. That is, statically retrieve the service metadata saved to the application when the service connection was first created—a **static** service connection.
- Always get the service definition from `https://service.com/openapidef`. That is, dynamically retrieve the service metadata from the source URL each time you open the

application—a **dynamic** service connection. The service definition in this case is only a "reference", a pointer to the OpenAPI document located outside Visual Builder.

In the service connection wizard, you can use the **Metadata Retrieval Option** to specify how you want to retrieve service metadata:



Here's what each option provides:

- Choose **Dynamically retrieve metadata** if your service definition changes frequently and you want to include these updates, especially during development, when things are rapidly evolving. In other words, by creating a dynamic service connection, you guarantee that a new field added to the service definition will be available to your application, because the service metadata is fetched every time the application is opened. This picks up all changes made to endpoints, including the new field. The metadata will always be up to date.

  > **Note:**
  >
  > You can dynamically retrieve metadata only for service connections that provide a service specification or those derived from a catalog, specifically the Oracle Cloud Applications backend.

- For better runtime performance, consider the **Copy full OpenAPI to your application** to your application option. Dynamic service connections pull entire resources, not just individual endpoints, which can make the `openapi3.json` metadata file unwieldy. If you check the **Automatically include list of values (LOV)** checkbox as well, the size of the file increases even more. Coupled with factors such as complexity of the schema and network latency, dynamically retrieving metadata may take a while to retrieve and process. Using the **Copy full OpenAPI** option and selecting just the endpoints you need for your application can improve performance, as your endpoint definitions are stored locally in the application for faster retrieval.
- While you might consider creating a dynamic service connection for development, then converting the connection to a static one when your application is ready for production, there's a better option: **Copy minimal OpenAPI to the application**. This option provides the best of both worlds when you create a service connection that points to an ADF Describe.

The **Copy minimal OpenAPI to the application** option is available only for services that are based on ADF Describe (like Oracle Cloud Applications' ADF BC-based REST APIs) and have a minimal describe endpoint to get limited metadata. This option stores service metadata for the endpoints you select in your extension's sources, much like a static service connection, but it only copies the minimal describe for those endpoints. It also dynamically retrieves the

parameter or request/response schema similar to a dynamic service connection. But it does this only when required (say, when a user tries to bind a table with an endpoint's response), not every time the extension is opened. Because only the minimal OpenAPI is copied to your application only for the endpoints you select, the size of the metadata file is reduced. And because the schema object is still referenced, the latest service definition is dynamically retrieved whenever required. For optimal performance, this is the recommended option for ADF Describe-based services.

Here's a quick breakdown of the advantages and disadvantages of each metadata retrieval option:

Service Connection Option	Advantage	Disadvantage
Copy full OpenAPI to your application (Static)	Better performance as service metadata is retrieved locally from the application's sources	Application is out of sync from the latest service definition and might not have recent customizations. Also, because full service metadata (including child objects) is saved to the application's sources, runtime performance may be impacted if this metadata is very large. Optimizing your resources can help in reducing the fetch time of the static OpenAPI.
Dynamically retrieve metadata (Dynamic)	Provides the most up-to-date service definition for your application	Performance may be impacted, though not always

Service Connection Option	Advantage	Disadvantage
Copy minimal OpenAPI to the application (recommended) (Static + Dynamic)	Optimal performance as minimal service metadata is stored and can be retrieved faster from the application's sources; granular endpoint selection is also possible. Dynamically referenced schema objects provide the ability to retrieve the most up-to-date request/responses schema.	Performance may be impacted when schema objects are retrieved

Note: Available only for ADFDescribe-based services l

Service Connection Option	Advantage	Disadvantage
ikeOracleCloudApplications		

No matter which option you choose to create your service connection, you can switch it up any time you want, as described in Convert a Service Connection (Static to Dynamic or Dynamic to Static). You can also change things when you edit a service connection to add endpoints, as described in Add More Endpoints to a Service Connection.

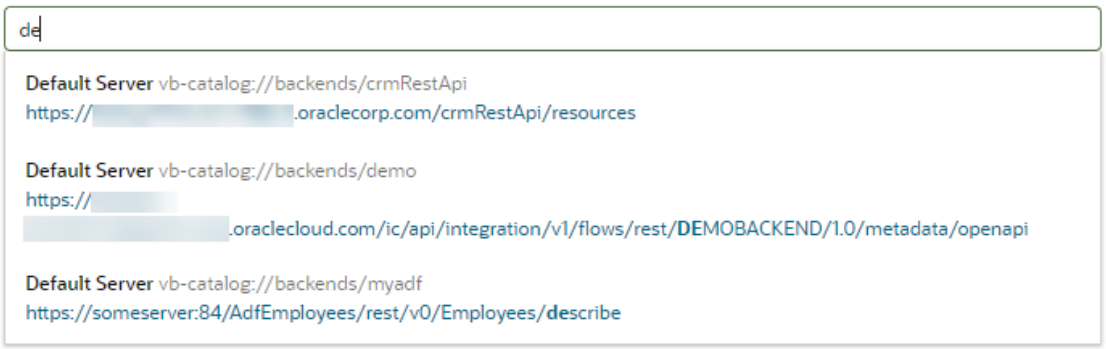
Create a Service Connection

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.

You can create service connections to REST services that support both the OpenAPI 3.0 and Swagger 2.0 specifications. If the service description that you use to create the service connection includes an error, Visual Builder displays it.

Your service connection can be derived from an existing backend (either predefined or custom) or you can register a backend when you create a service connection. To re-use existing backends, choose from a list of existing backends that is displayed based on what you enter in

the URL, as shown here:

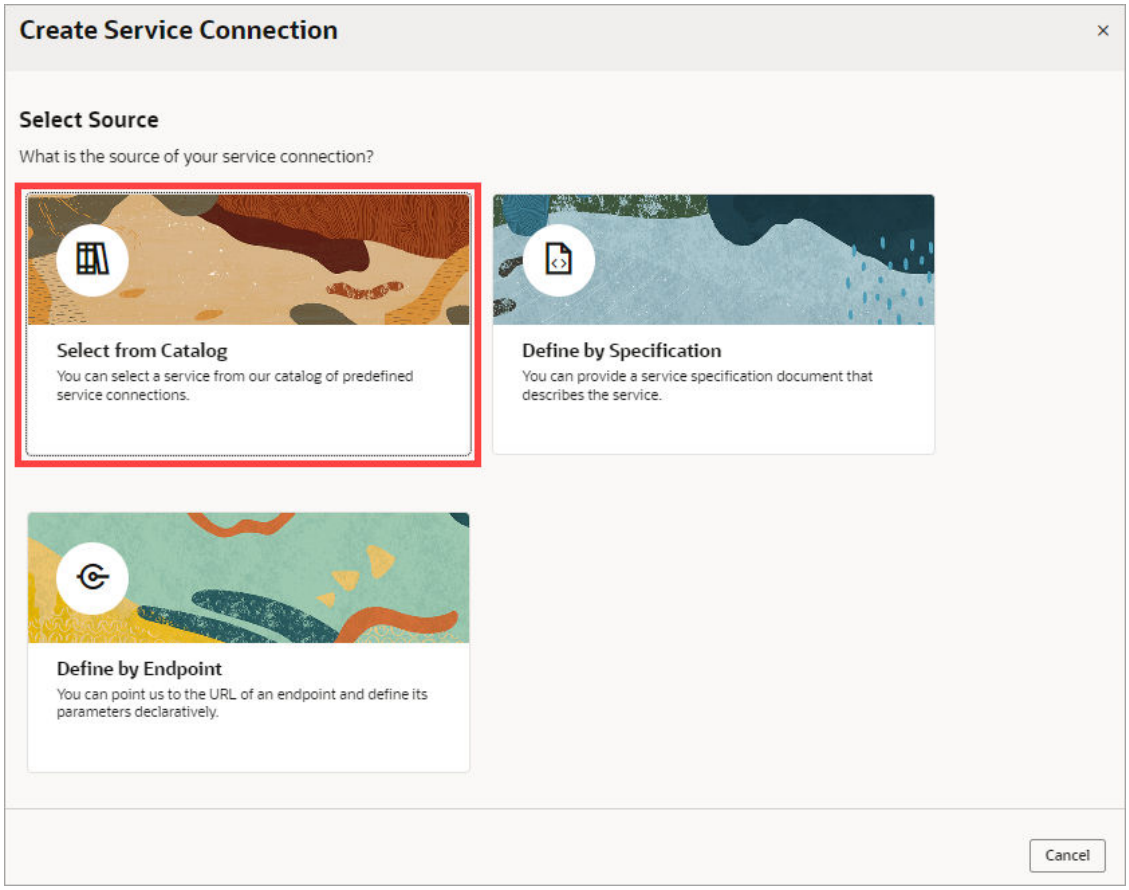


You'll see this list when you register service connections via a service specification document or an endpoint URL. For connections to services in a catalog, the backends list appears only when you choose custom backends registered by specifying a service specification document.

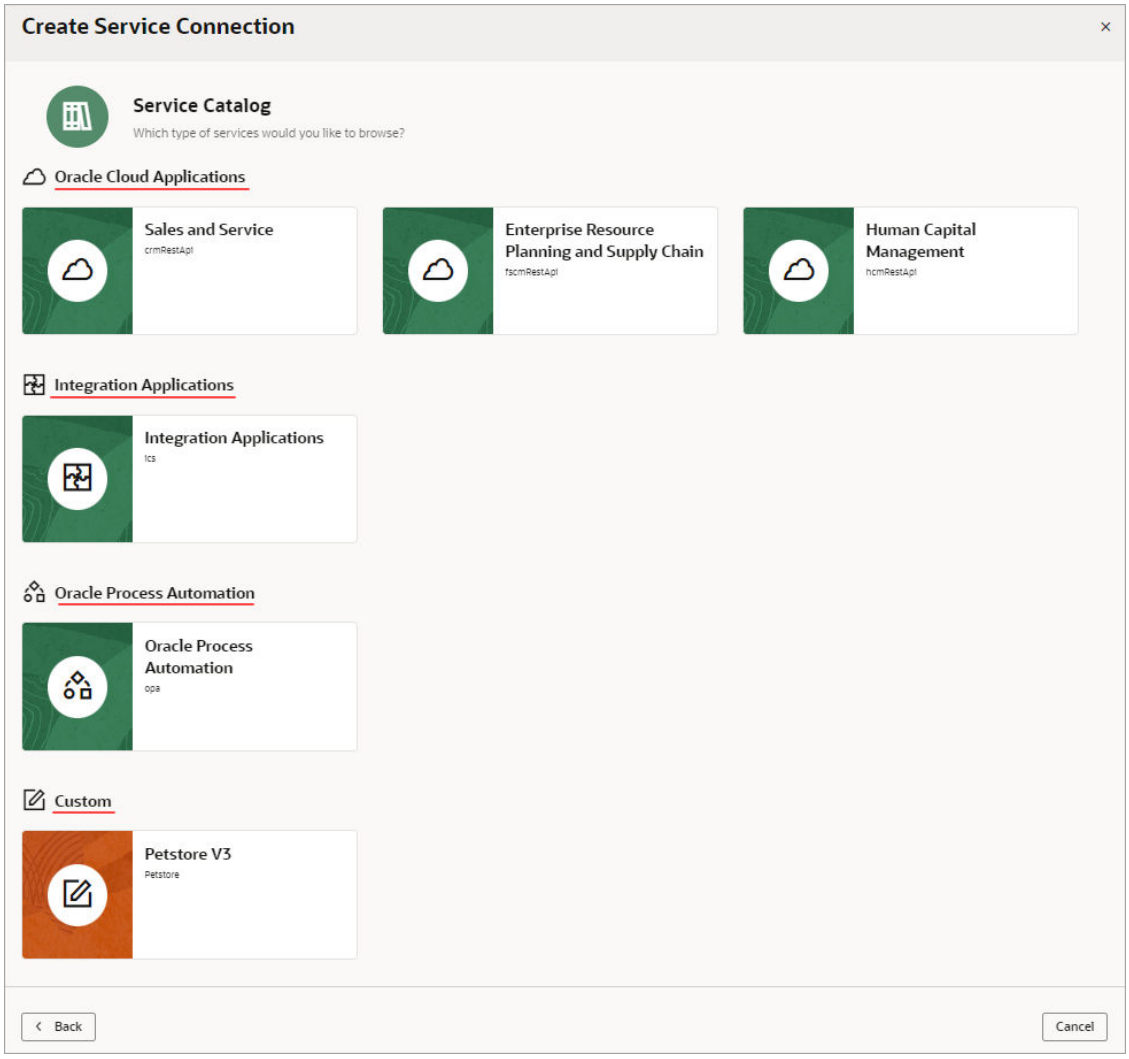
> **Note:**
>
> All URLs in backends and service connections should be HTTPS with a valid certificate.

Create a Service Connection from a Catalog

A catalog provides a list of services and their endpoints from an Oracle cloud service backend or a custom backend. Using a catalog to create a service connection saves you time, as common configuration tasks like authentication, connection type, and headers are predefined for you.



To have a catalog of Oracle cloud services, such as Oracle Cloud Applications, Integration Applications, or Oracle Cloud Infrastructure Process Automation services, you need to have an instance of that service provisioned, and you need to have a backend that represents that service. To have a catalog of custom services, you need to have a custom backend for the services. For details on how to create a backend for an Oracle service, see Create a Backend, and for details on how to create a custom backend, see Create a Custom Backend.



> **Note:**
>
> For App-Driven Orchestration integrations with Visual Builder, use integrations that have a published REST specification, Active status and a /metadata endpoint. You can also de-activate, then activate the integration, and it will be visible in the catalog.

To access a service that isn't listed in your catalog, you can create the service connection by using a service specification (ADF BC REST, OpenAPI/Swagger), or by specifying an endpoint URL.

If you do not see any services in the catalog, confirm the following:

- You have supplied all the details for a given backend service including base URL, required headers, authentication, and connection type. If you are connecting to an Oracle service instance, use the Backends tab in the Services pane to check the URL and authentication method in the backend service.
- You are authorized to access the service with your credentials. Contact the service's administrator to confirm that your credentials are authorized.

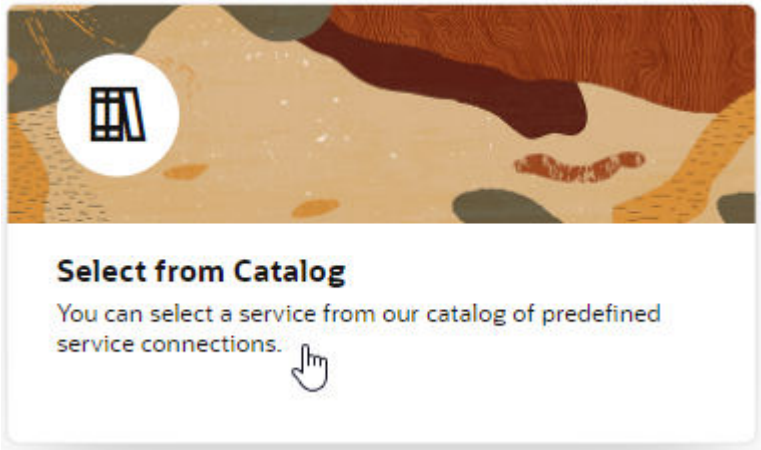
- The service is currently available. Check the connection status of the service manually using a tool such as cURL or Postman.

Create Service Connections from the Oracle Cloud Applications or Integration Applications Catalog

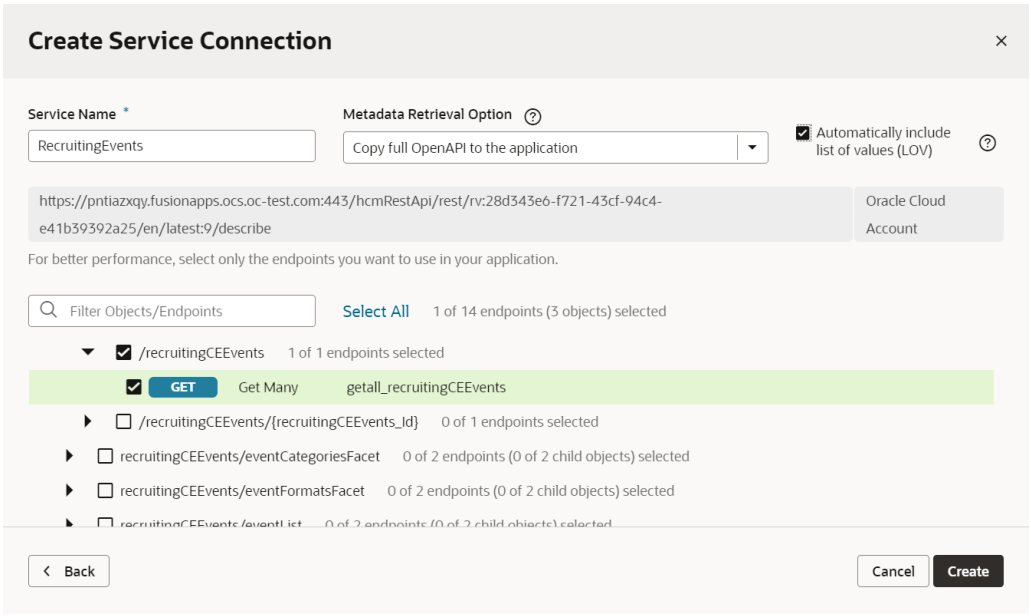
The Oracle Cloud Applications and Integration Applications catalog provides a list of services and their endpoints for you to quickly create needed service connections.

To create service connections from one of these catalogs:

1. Click **Services** in the Navigator.
2. In the Services pane, click the **+** sign and select **Service Connection**.
3. Click **Select from Catalog** in the Select Source step of the Create Service Connection wizard:



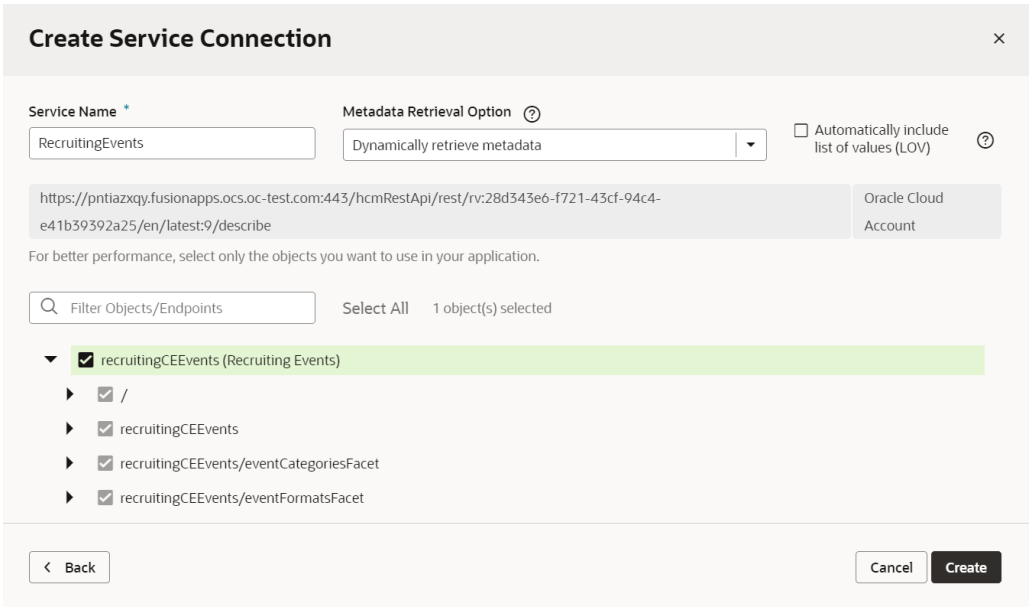
4. Select the Oracle Cloud Applications or Integration Applications catalog you want to browse.
5. Enter a Service Name, and select the REST endpoints you want to add from the list of endpoints available for each resource provided by the service.
    - If you want your endpoint definitions to be stored in your visual application's sources, select **Copy full OpenAPI to the application** in the **Metadata Retrieval Option** drop-down list, then select the endpoints that you require for your application. This way, you create a static service connection:



Tip:

You can select a top-level object to select all endpoints for that object, or select individual endpoints to improve performance.

- If you want your endpoint definitions to always be dynamically retrieved from the service metadata, select **Dynamically retrieve metadata** in the **Metadata Retrieval Option** drop-down list, then select the top-level resource that you require for your application. This option changes the resource selection to include complete objects instead of individual endpoints:



> **Note:**
>
> For ADF Describe-based services, the recommended option for metadata retrieval is **Copy minimal OpenAPI to the application** (default). This option is a happy medium between copying the full OpenAPI endpoint definition to your application's sources and always retrieving the OpenAPI definition from the source URL. It copies a minimal OpenAPI definition for the endpoints you select and dynamically retrieves the request/response schema only when required.

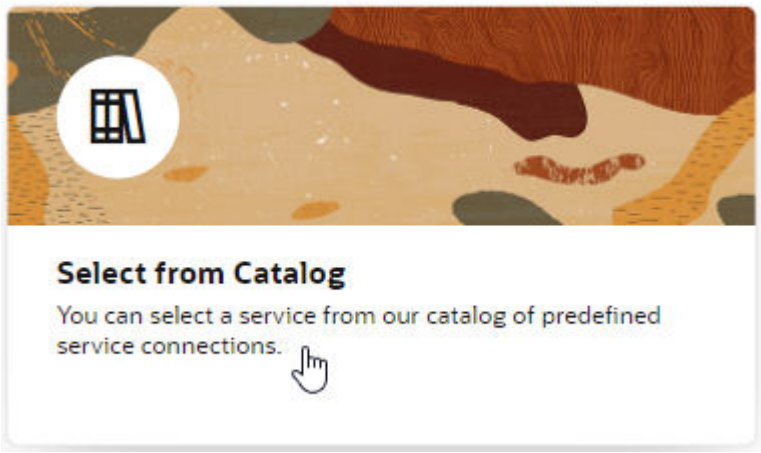
6. Optional: If you want all list of values (LOV) for the selected objects/endpoints in an Oracle Cloud Applications catalog to be automatically included in the service metadata, select **Automatically include list of values (LOV)**. Take note that selecting this option may increase the size of the `openapi3.json` file that holds service metadata and potentially impact performance.
7. Click **Create**.

Create Service Connections from the Oracle Cloud Infrastructure Process Automation Catalog

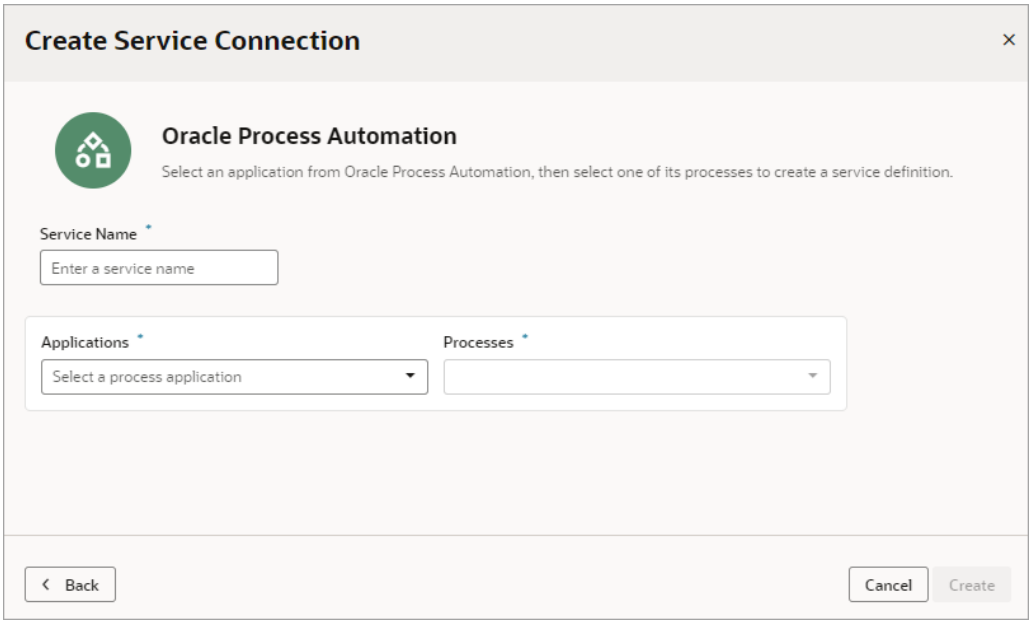
The Oracle Cloud Infrastructure Process Automation (OCI Process Automation) catalog provides a list of services and their endpoints for you to quickly create needed service connections.

To create service connections from the OCI Process Automation catalog:

1. Click **Services** in the Navigator.
2. In the Services pane, click the **+** sign and select **Service Connection**.
3. Click **Select from Catalog** in the Select Source step of the Create Service Connection wizard:

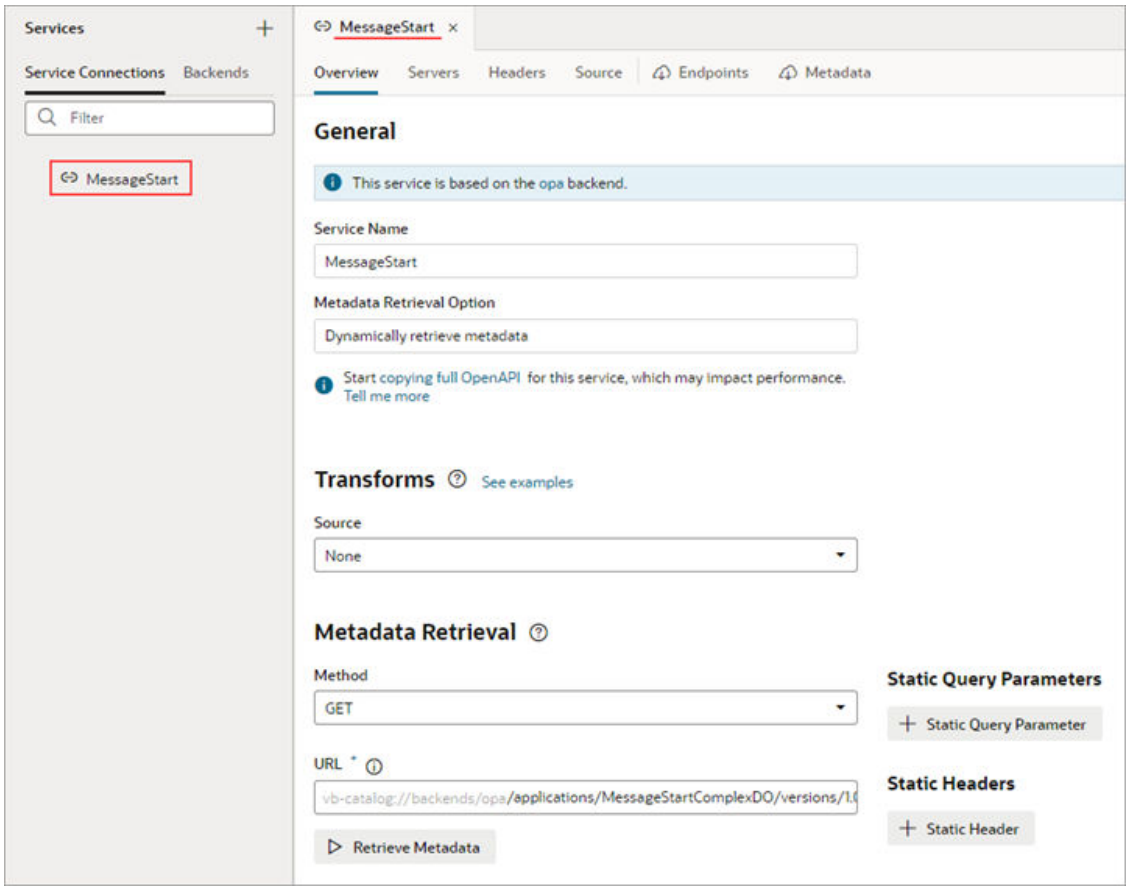


4. Select the OCI Process Automation catalog.
5. Enter these details for the OCI Process Automation service:
   a. **Service Name**: Enter a name for the service.
   b. **Applications**: Select the process application with the OCI Process Automation process you want to use.
   c. **Processes**: Select the OCI Process Automation process, in the process application, to which you want to connect.



6. Click **Create**.

Your new OCI Process Automation service is added to the Service Connections tab of the Services pane. Its details are shown on a tab to the right of the pane, for you to further configure the connection, if necessary:



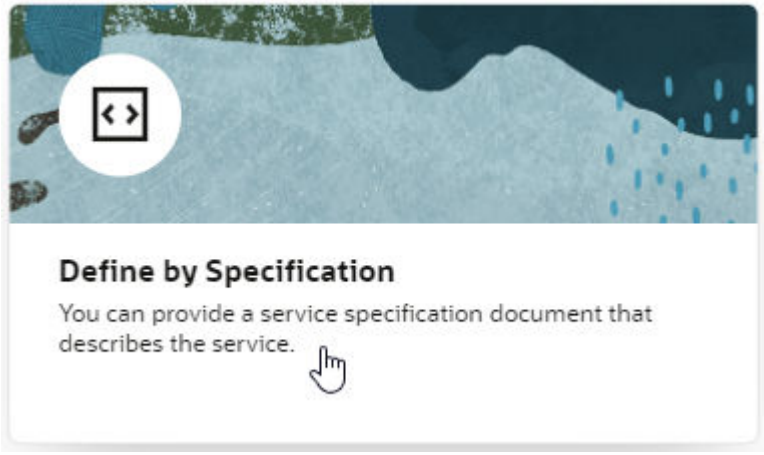
You can now use the new service as required in your application.

Create a Service Connection from a Service Specification

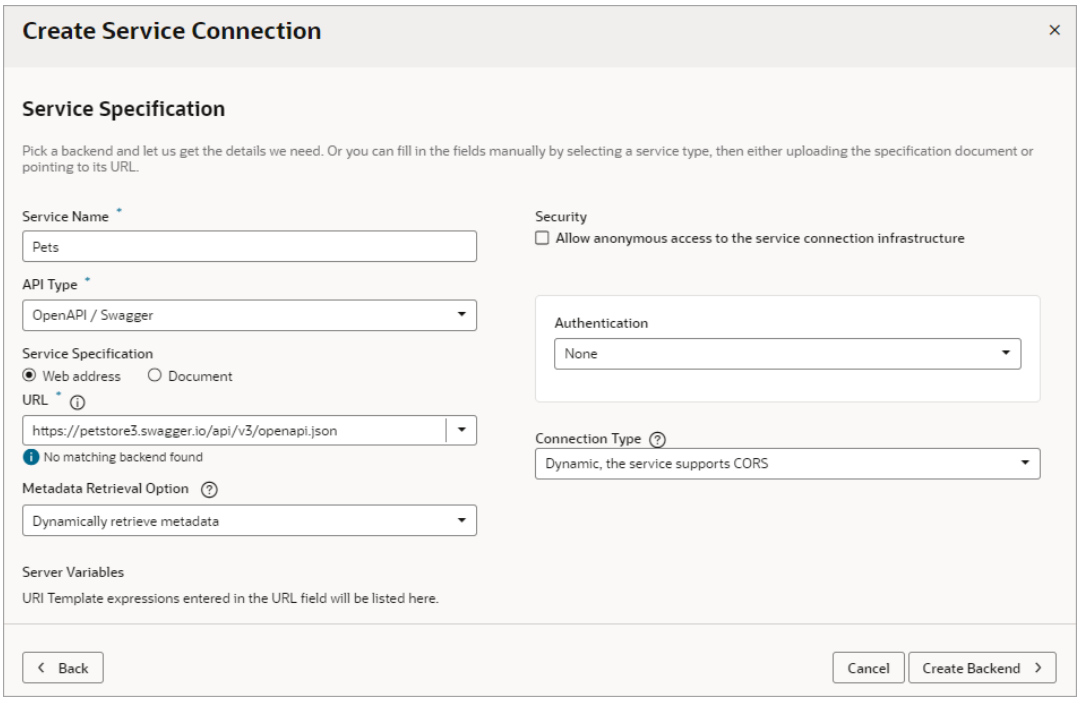
You create a connection from a service specification when you know the URL of the OpenAPI/Swagger or Oracle ADF Describe file that describes the service. You can also upload the specification file from your local system.

To create a connection from a service specification:

1. Click **Services** in the Navigator.
2. In the Services pane, click the **+** sign and select **Service Connection**.
3. Click **Define by Specification** in the Select Source step of the Create Service Connection wizard:

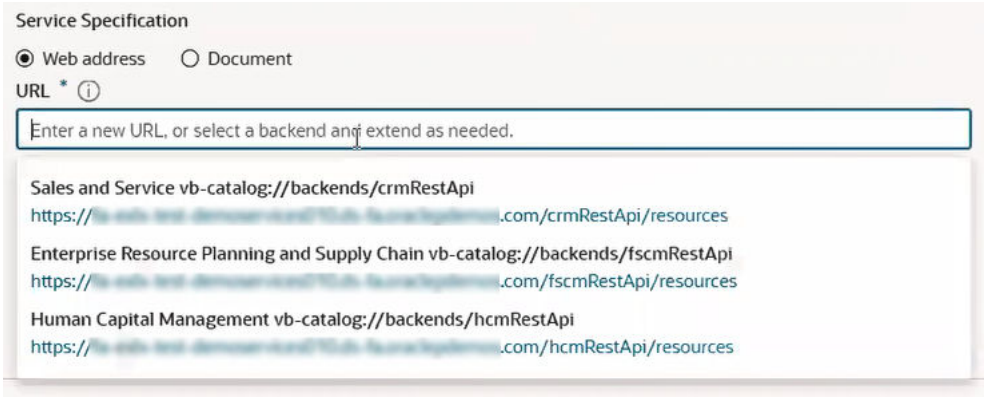


4. Enter a name for the connection in the **Service Name** field.



The name you specify will be the connection's display name in your application.

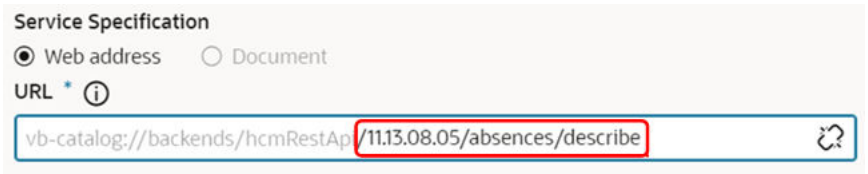
5. Select the **API Type** for the service you want to connect to:
   - Select **OpenAPI / Swagger** if you have an OpenAPI/Swagger specification for your service.
   - Select **ADF Describe** for any REST API that has an ADF Describe file, including Oracle Cloud Applications REST API.
   - Select **ADF Describe (cache-enabled)** if you need to create a service connection to an Oracle Cloud Applications REST API that has a Describe URL.
6. Select the location of the **Service Specification** document.
   - *If you're creating a service connection to an Oracle Cloud Applications REST API* (for example, CRM, FSCM, HCM), select **Web address**, click the **URL** text box and select the appropriate backend to fill in the service description's URL.



Then type in the rest of the Describe URL to the resource using this format:

`<backend URL path>/<REST-API-version>/<resource-name>/describe`

For example, for the HCM absences resource, you'd add `/11.13.08.05/absences/describe` to the selected HCM URL in the text box, where 11.13.18.05 is the REST API version.



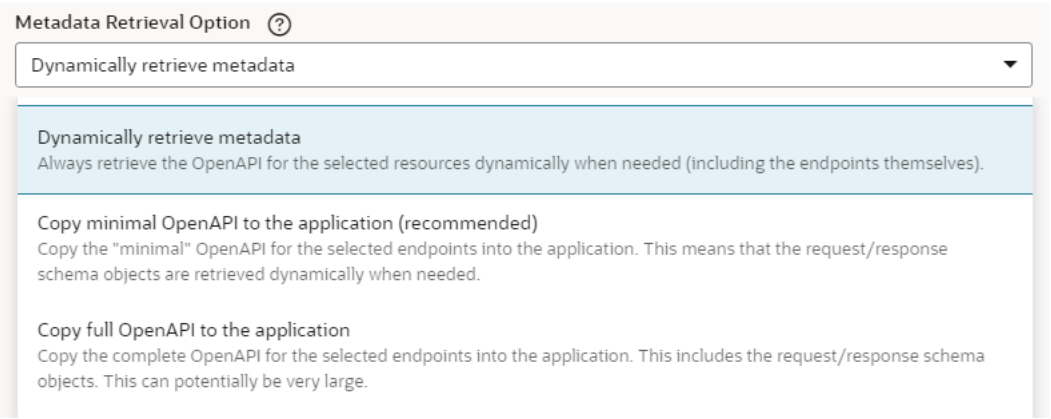
To ensure that the metadata is cached efficiently, specify the actual REST API version (for example, 11.13.18.05) instead of using “latest” for the version. To figure out the latest REST API version, consult the product's Oracle Cloud Applications REST API documentation.

> **Tip:**
>
> To specify multiple resources (for example, both contracts and expenses), you'll need to use a slightly different URL. Use this format if you're specifying the contracts and expenses resources:
>
> `<backend URL path>/<REST-API-version>/describe?resources=contracts,expenses`

- *If you have an external URL representing the service specification* (for example, an OpenAPI3 URL), select **Web address** and enter the service descriptor's URL in the **URL** field.
  If you're using an IP address instead of a proper DNS-based URL in a production environment, you're probably using self-signed certificates. Certificates based on IP addresses are less secure and more difficult to maintain than DNS-based ones. To avoid potential security issues, self-signed certificates should never be used in production environments. An audit warning will be displayed whenever you use an IP address-based service connection.
- *If you have a document that contains the service specification*, select **Document**, then either drag and drop the file that describes the service from your local system to the dialog or use the file browser to locate and select it on your system.

> **Note:**
>
> If the service specification URL does not match an existing backend, you will need to create a new backend. Instructions for creating a new backend from a service specification URL are covered in a later step.

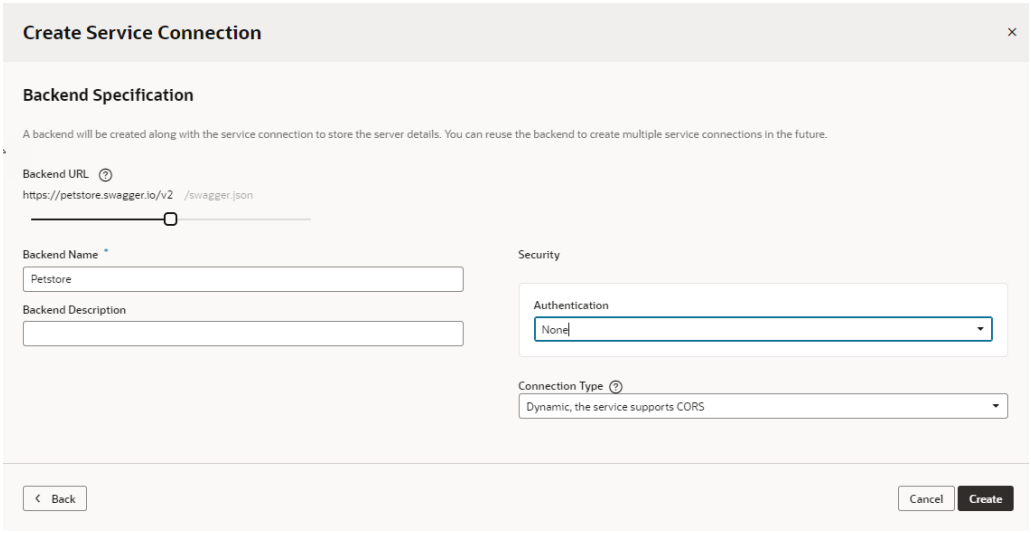
7. Select a **Metadata Retrieval Option**.



If you entered a web address, you can select **Dynamically retrieve metadata** to create a dynamic service connection that always pulls in the most up-to-date service definition for your application or you can select **Copy full OpenAPI to the application** to copy the complete service metadata during development and statically store it in your visual application's sources.

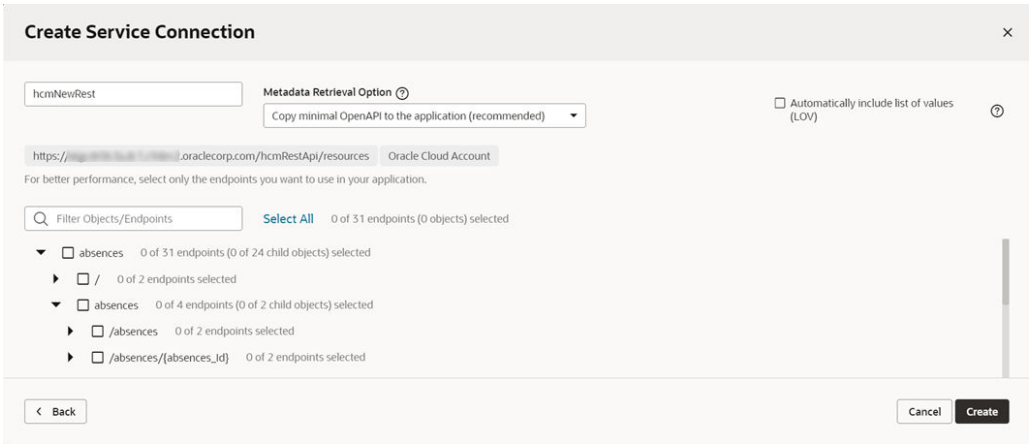
If your API type is ADF Describe, you may want to select **Copy minimal OpenAPI to the application (recommended)** to copy the minimal metadata for the endpoints you'll select in the next step and dynamically retrieve the request/response schema, when required.

8. If you did not have a backend to match your service specification URL, add service connection details as needed, then create a backend:
   a. Update the **Security** and **Connection Type** settings as needed.
   b. Click **Create Backend**.
   c. In **Backend URL**, use the slider to specify which part of the URL you want to use for the backend.



   d. Add a name and optional description for the backend.
   e. Update the **Security** and **Connection Type** settings for the backend as needed.
9. Do one of the following:
   - If you have an **ADF Describe** API type, continue to the next step.
   - If you have an **OpenAPI/Swagger** API type, click **Create**. The procedure is complete.
10. Click **Next** and select the resources and endpoints you want to add.

    The Select Endpoints pane displays a list of the endpoints and child objects available for each resource provided by the service.
    a. 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.



    b. Click **Create**.

After creating a service connection, 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.

You can also test the service connection like this:

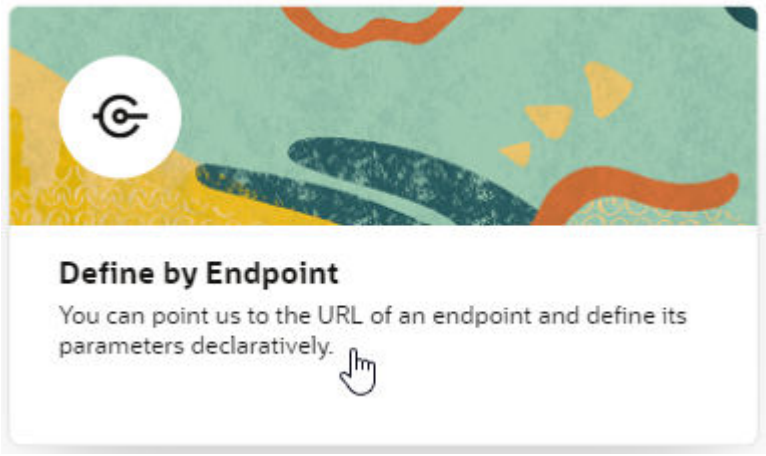
1. From the **Endpoints** tab, select one of the resource's methods to test.
2. Go to the **Test** tab and, under Response, enter an integer for a query parameter.
3. Click **Send Request**.
4. Check the Send Request status. A status of 200 indicates success.

Create a Service Connection from an Endpoint

You can 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 Response body.

To create a service connection from an endpoint's URL:

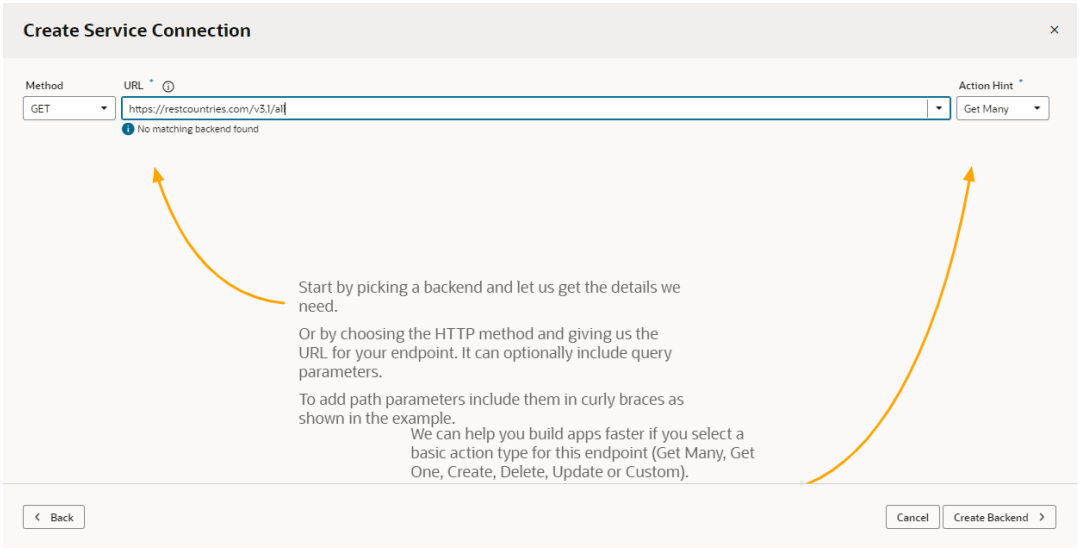
1. Click **Services** in the Navigator.
2. In the Services pane, click the **+** sign and select **Service Connection**.
3. Click **Define by Endpoint** in the Select Source step of the Create Service Connection wizard.



4. Select the HTTP method and enter the endpoint URL.

> **Note:**
>
> If you're using an IP address instead of a proper DNS-based URL in a production environment, you're probably using self-signed certificates. You need to be aware that certificates based on IP addresses are less secure and more difficult to maintain than DNS-based ones and, to avoid potential security issues, self-signed certificates should never be used in production environments.
> An audit warning will be displayed whenever you use an IP address-based service connection.

If you know the result expected from the endpoint URL, select it in the **Action Hint** drop-down list to indicate what the endpoint does. For example, when you select **GET** as the **Method**, select **Get One** as the Action Hint if the endpoint URL will retrieve a single record, or **Get Many** if the endpoint will retrieve multiple records. If the endpoint URL will create a record, you would select **POST** as the method and **Create** as the Action Hint.

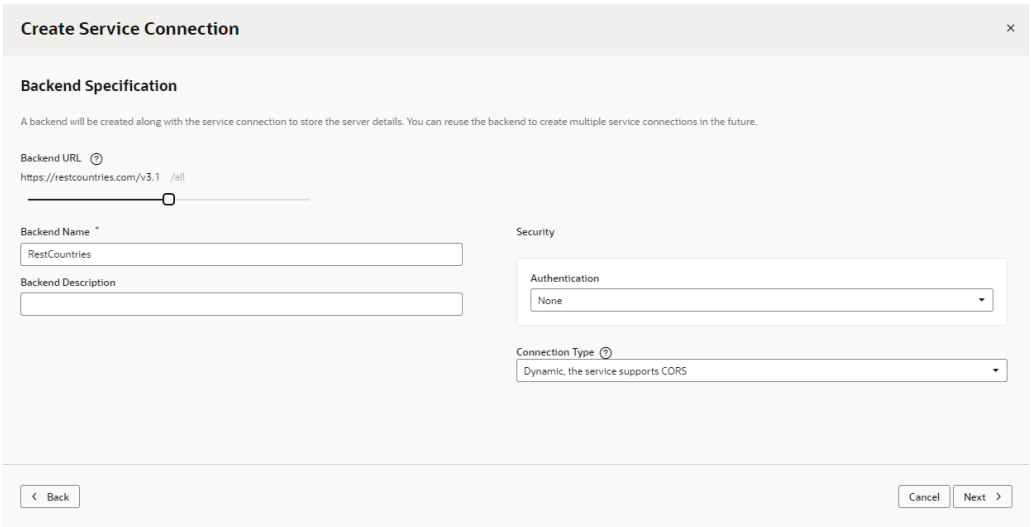


If an existing backend match is found, the URL will fill the field automatically; otherwise, you will need to create a new backend. See the following step for more details.

When you're done, click **Next**.

5. If you did not have a backend to match your service specification URL, click **Create Backend**.

   a. In **Backend URL**, use the slider to specify which part of the URL you want to use for the backend.



   b. Add a name and optional description for the backend.

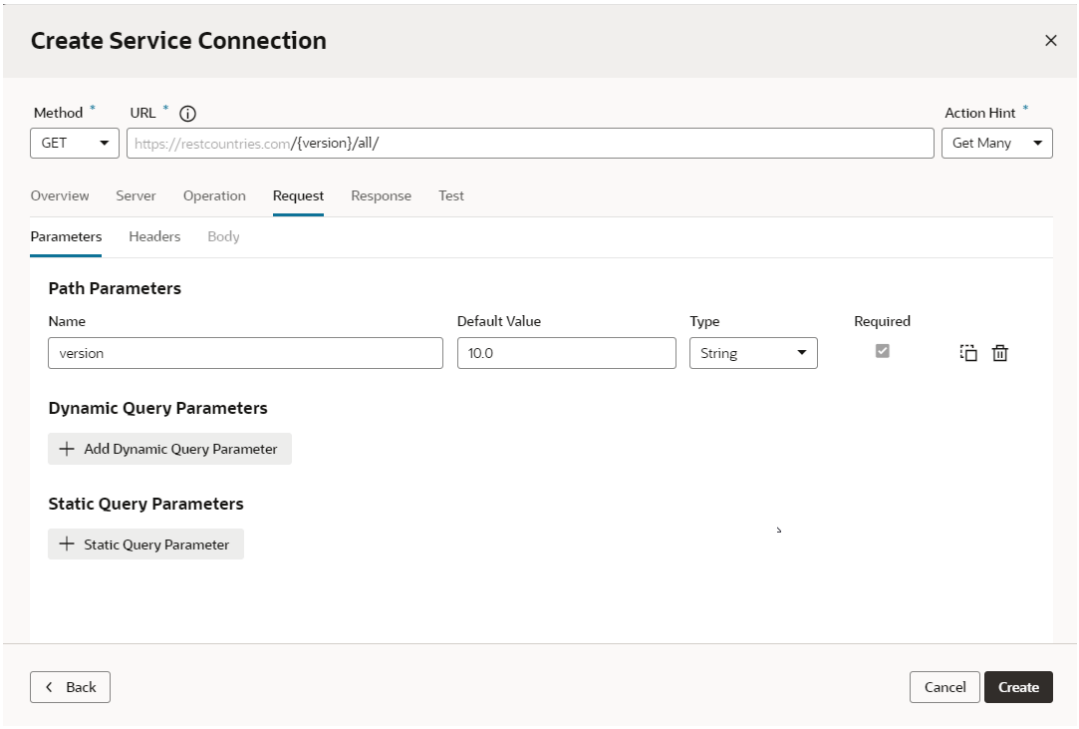
   c. (Optional) Change the **Security** and **Connection Type** settings for the backend.

   d. Click **Next**.

6. In the **Overview** tab, add a Service name, and confirm the title and version. Optionally, enter a description.
7. If you want the service connection to only be used by the server (for example, from a Groovy script), select **Server Only Connection**.
8. Click the **Server** tab. If you selected a pre-existing backend in Step 4, all the details here will be read-only. If you created a new backend, complete the following details:
   - Specify an application profile if you want to use the server that is specified here for a specific phase of an application's development.
   - Add any static headers (custom and secure) to be used when the service connection connects to the REST service using this server.
   - If your service requires authentication, select an authentication option for logged-in users.
     Authentication is **None** by default.
   - Choose a connection type, set by default to **Dynamic, the service supports CORS**. Only choose **Dynamic, the service supports CORS** if you know that the external REST service has enabled CORS for the Visual Builder domain. If the external REST service hasn't enabled CORS, choose a different option, such as **Dynamic, the service does not support CORS**. The service connection and REST API's CORS settings must match, otherwise you'll get an error.

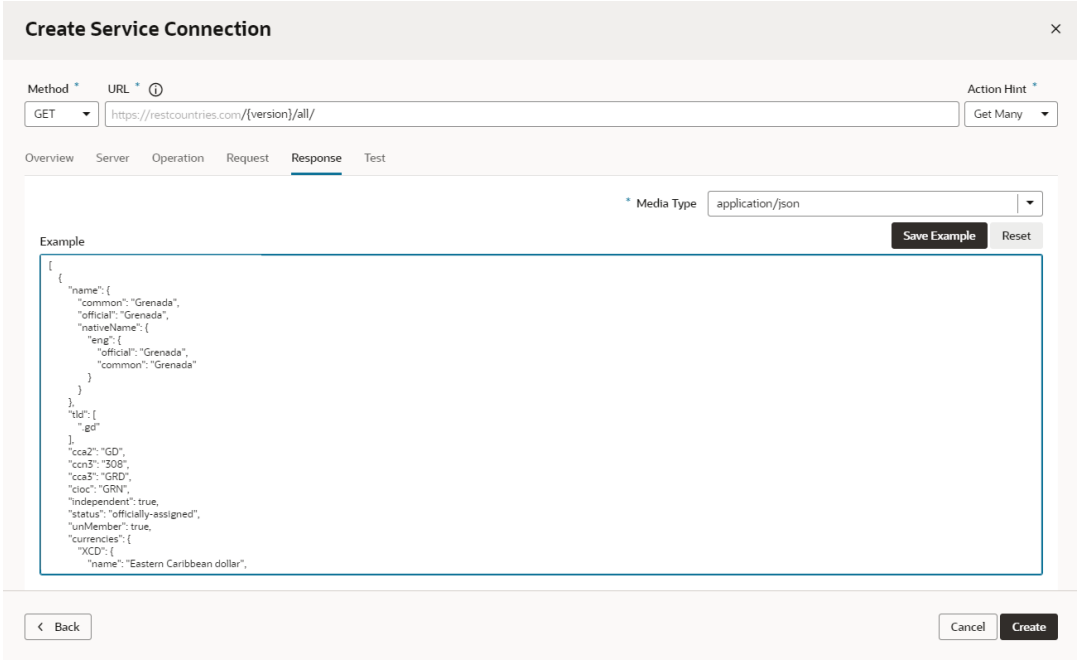
     For more information about setting the connection type, see Use an Appropriate Connection Type to Handle CORS for REST Services.
9. Click the **Operation** tab to view the Endpoint ID that Visual Builder will use to identify the REST API endpoint you specified at the start of this task.
10. Click the **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.

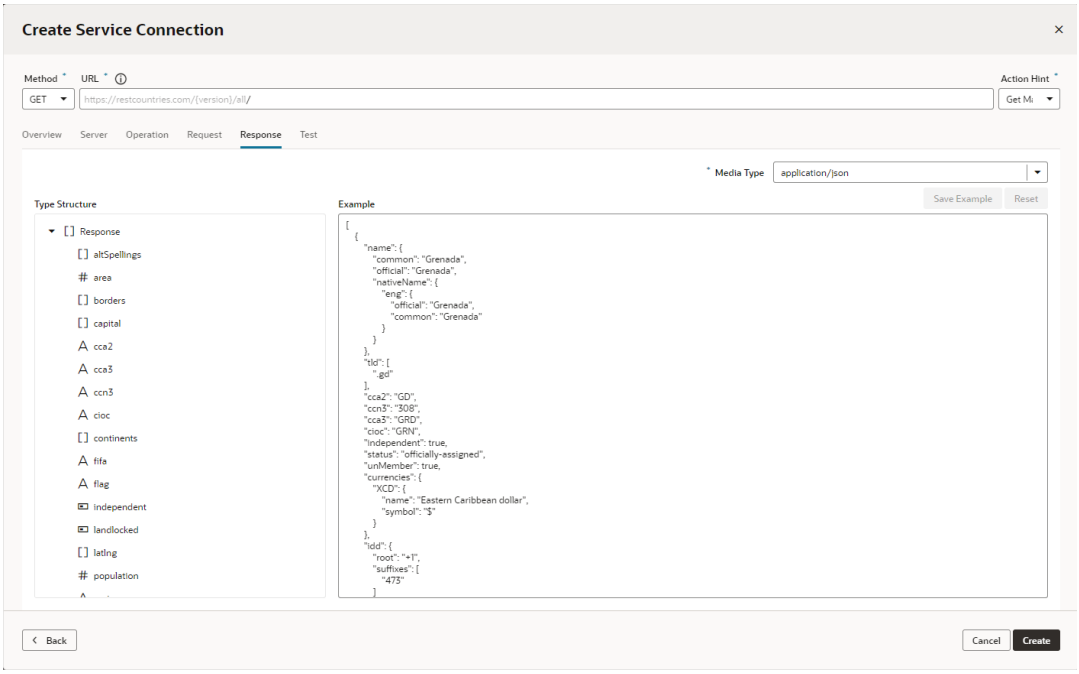


11. Click the **Response** tab and enter the response body for the endpoint.

The **Response** tab displays the media type's OpenAPI3 metadata artifacts that can be represented: the example and the schema. You paste in an example of the body of the response into the text area and then click the **Save Example** button to commit your input or click the Reset button to clear it and start over. These buttons will stay disabled until you add or edit the example text.



After you click the **Save Example** button, your new example content is saved, the schema is generated, and the Type Structure panel displays.

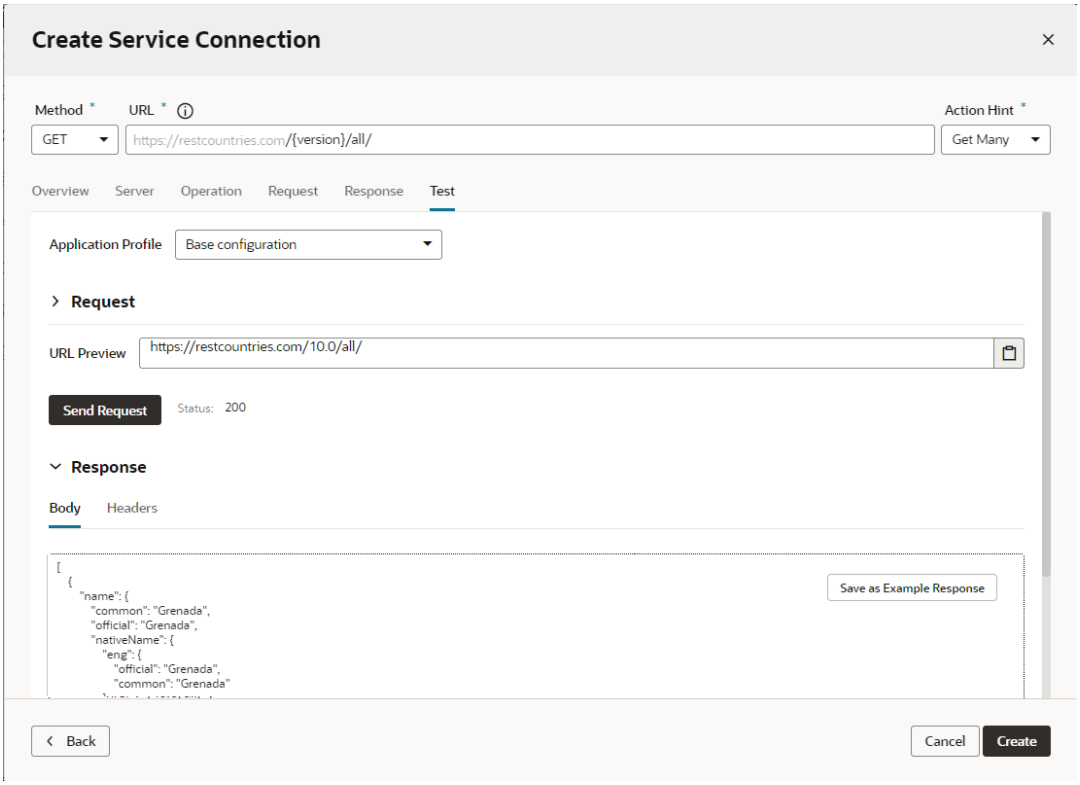


If you don't have an example to add, you can use the **Test** tab to send a request to the service, then save and use the response that is returned as your example (or edit it as needed) in the tab's text area.

> **Note:**
>
> The panels you see in the **Response** tab are determined by the presence or absence of the example and the schema:
>
> - If neither the schema nor the example exist, such as when you're creating a new service by using the Define by Endpoint flow in the wizard, the tab displays just the Example text area.
> - If you arrive at a tab for an existing service whose endpoint already has the schema and example defined, the Type Structure panel and the Example text area are both displayed.
> - You may also encounter cases where existing endpoints have a schema defined, but there is no example. In most of these cases, it's either undesirable or potentially detrimental to include the Example text field on the page, so a placeholder panel with a boilerplate message that says there is no example is displayed instead.

12. Click the **Test** tab to test your request (based on the settings in the other tabs) and view the response from the endpoint.

    Click **Send Request** to view the Response body and headers and confirm that the data you receive is what you're expecting.



    You can experiment with different request parameters until you achieve the response you want. If your response returns an error, check the details of your connection, for example, ensure that you're using the correct credentials or that the service uses a valid SSL certificate.

13. 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. For example, defining a Get Many endpoint is enough if you only want to view records, but you'll need to create more endpoints to create, edit, or delete records.

Edit a Service Connection

After you create a service connection, you can edit it to add and remove endpoints, modify requests, add functions for filtering and sorting responses, and more.

For each of your service connections, you can use the following tabs in the connection editor to view and edit the connection's details. What you see depends on whether your service connection is static or dynamic:

Tab	Description
Overview	Displays the title and version of the REST API that you create a service connection to. You can edit the title of the service as it appears in your visual application editors. You can also use this tab to add transform functions to the service connection, which would override any corresponding transforms applied to its backend.
Server	*If a service connection was created with an association to a backend*, the backend is displayed. Server details can be updated from the backend. See Edit a Backend. **Note:** Starting with release 23.10, all service connections used in extensions must be associated with a backend. If you have older service connections that you still want to use, create a new backend, then migrate your existing service connections to it. You can select the authentication method you want to use when connecting to the service. See Set the Backend's Authentication Method and Connection Type.
Headers	Displays the headers written in the REST call to the service. You can add and edit headers in the tab.
Source	Displays the OpenAPI 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 in your applications. You can edit the entries in the **Source** tab.

Tab	Description
Endpoints	• For static service connections, displays a list of the service endpoints that you selected when you created the connection. Each endpoint in the list has an options menu where you can choose to edit, duplicate, or delete the endpoint. To add another endpoint from the service, click **+ Endpoint**. You can add transform functions to an endpoint, which would override any corresponding transforms applied to its service connection and backend, by editing it and using its Overview tab. See Understand Data Access Through REST for more on the options and parameters that you can use to configure service connections. • For dynamic service connections, the tab provides a read-only view of all endpoints in the service's OpenAPI metadata, as retrieved from the source URL. You can click **Edit Object Selection** to add more objects. You can also select or deselect the option to automatically include all related list of values for the selected objects.
Metadata	For a dynamic service connection, displays a read-only view of service metadata that is contained in the OpenAPI3 document, which was dynamically retrieved from the source URL.

Add a Server to a Service Connection

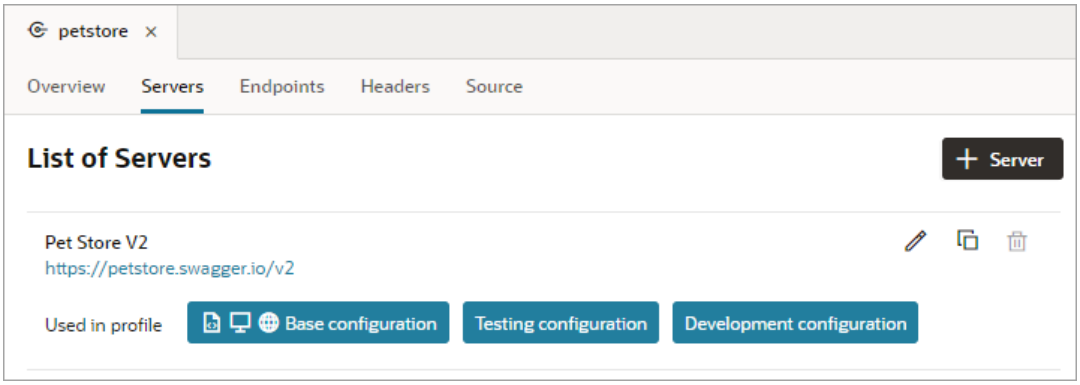
For older service connections created using the Visual Builder 23.10 release or earlier, which were not based on a backend, you can add different server definitions to a service connection, as explained in this topic. For service connections created after the 23.10 release, new server definitions must be added from the backend associated with the service connection. For more information, see Edit a Backend.

For older service connections, you'd add different server definitions and use them in conjunction with application profiles to set which server definitions to use during the different stages of an application's lifecycle.

For example, you might add a development server that hosts an instance of the REST service where non-customer data is used, and an authentication method like basic authentication is acceptable. This development server can be used to develop your application, but once you publish it to production, your apps need to connect to the REST service using a production server with more stringent authentication requirements and access to customer data.

To add a server to a service connection:

1. Open the **Servers** tab of the service connection where you want to add the server:



2. Click the **+ In-Source Server** button to open the New Server dialog. Alternatively, click the Copy icon to make a copy of an existing server and modify some of its entries.

   In both cases, a dialog appears where you can identify the server, specify an application profile that will use the server, add headers, and configure security options. These options are the same as those you configure when you initially created the service connection.

3. Once you complete the definition of the server options, click **Save** or **Copy** depending on the option you chose to add the server.

The first server in the List of Servers is the default server. It will be associated with all application profiles, unless you configure the other servers in the list to use another application profile.

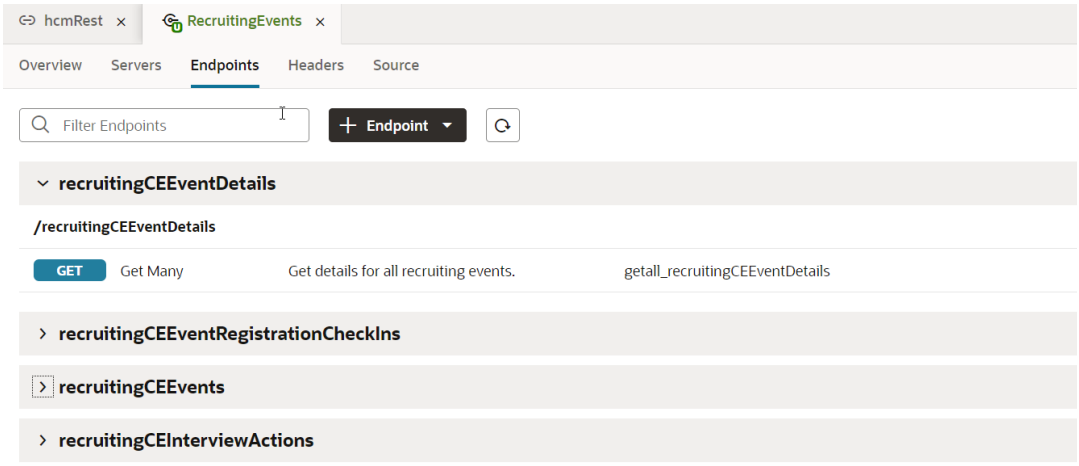
Edit Service Endpoints for a Static Service Connection

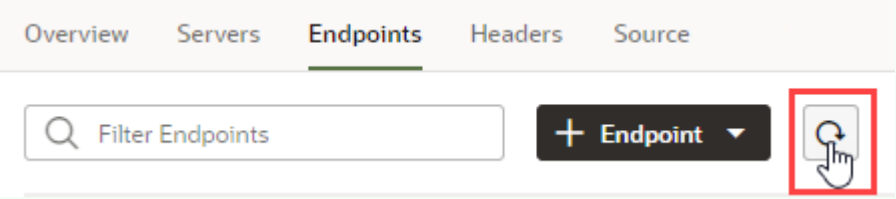
After you create a service connection and select the service endpoints, you can edit the endpoint settings, request parameters, and the response for each endpoint in the Endpoints tab. Endpoints can be edited only for static service connections (identified by the icon that appears in front of the service name).

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:

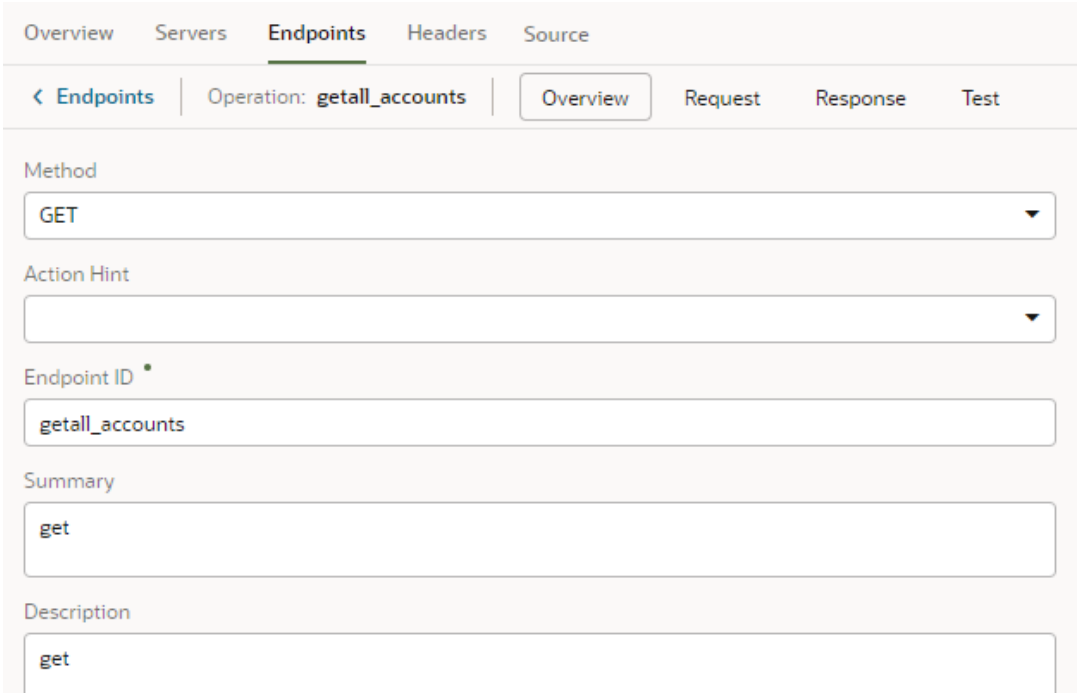
1. Open the Endpoints tab of a service connection.



> **Tip:**
>
> You can use the **Replace** button to update the service definitions of all selected endpoints with the latest definitions from the service. The **Replace** button is available when the service connection is registered via the Catalog or a service specification document. This capability is available for both OpenAPI3 style services as well as ADF Describe services:
>
> 
>
>
> For static Integrations service connections, using the **Replace** button lets you update all endpoint definitions without manually creating a new connection and deleting the old one. When you see the confirmation dialog, click **Replace** to proceed.

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.



   If your service connection was created from an OpenAPI3 specification that includes request and response examples specified in the `example` or `examples` keys, the examples will show up within the Request and Response tabs.

   The `examples` key takes precedence over `example` and if multiple examples exist, only one will be shown. If you created an example where one didn't exist (say, if you used the

Test tab to send a request to the service and saved the response as an example), that's the example you'll see; otherwise, you'll see the first example in the `examples` key list.

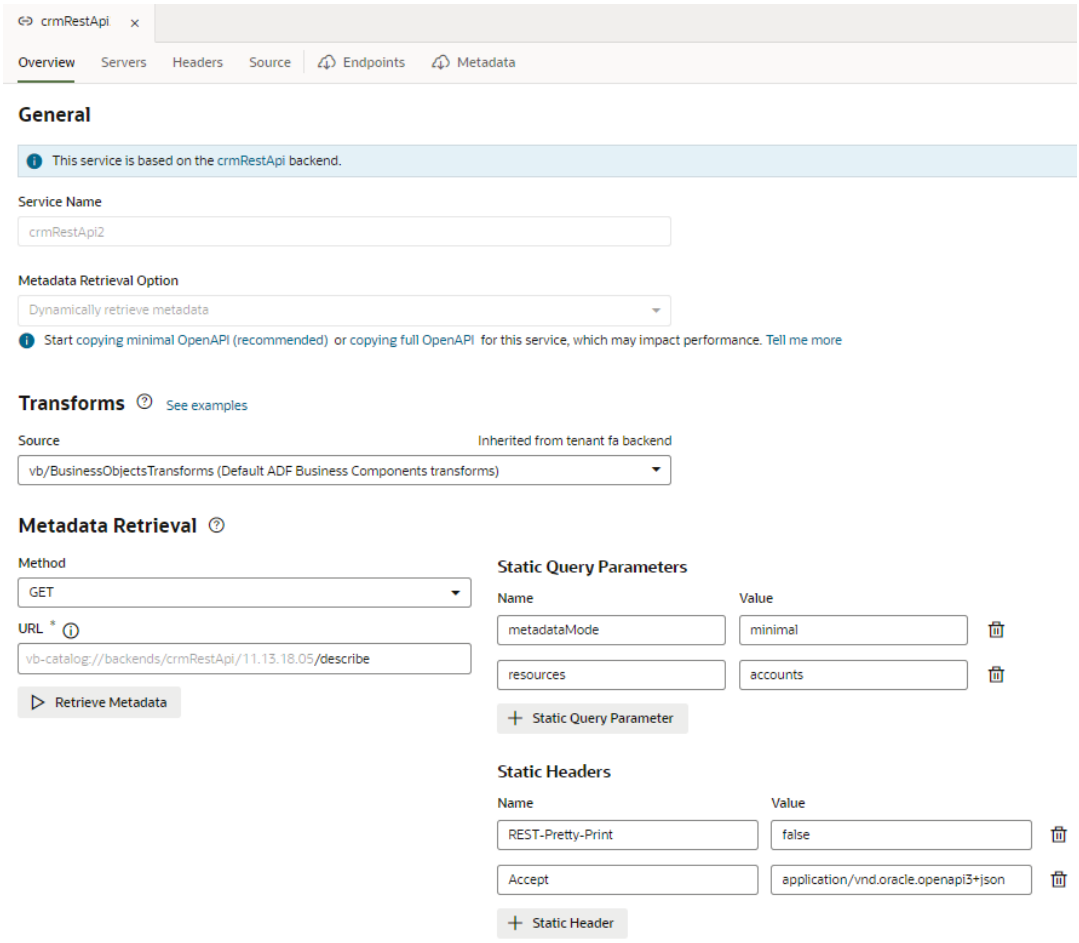
3. Click the service connection link in the breadcrumb to return to the list of service endpoints.

Retrieve Service Metadata for a Dynamic Service Connection

After a dynamic service connection is created, you can retrieve the service's most recent metadata for your application to use the latest service definition.

To retrieve the service metadata for a dynamic service connection:

1. Open the **Overview** tab of a dynamic service connection:



The Metadata Retrieval section defines the method and URL to be used to retrieve the OpenAPI document that describes the REST APIs for the dynamic service connection. In the example shown here, the GET method is used to fetch the most recent ADF Describe available on the `vb-catalog://backends/fa/crmRestApi/resources/latest` server.

2. Click **Retrieve Metadata**.

   The Endpoints tab opens, showing a read-only view of the endpoints for the service connection's objects dynamically retrieved from the service's `openapi3.json` metadata file in the source URL.

3. Click the **Metadata** tab to view the complete contents of this file, including endpoints and type definitions.

Add Server Variables for Service Connections

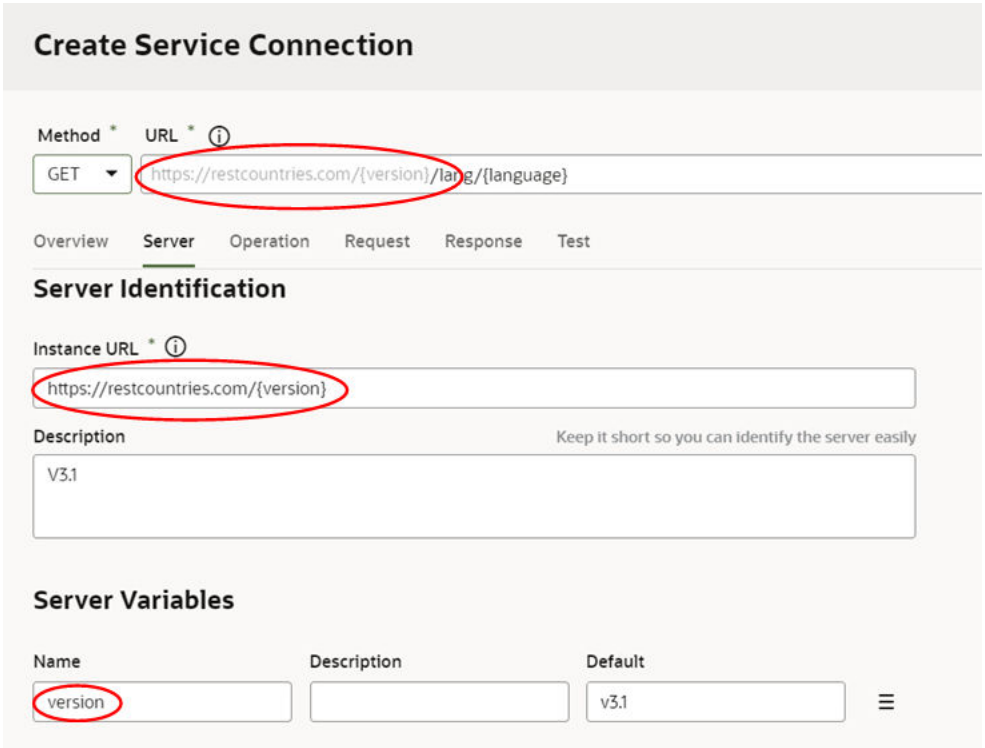
After creating a service connection, you can go to the Servers tab and edit the server's instance URL and add server variables.

> **Note:**
>
> For service connections created after the 23.10 release, server variables must be updated from the backend associated with the service connection. For more information, see Add Server Variables for Backends.

When you enter a valid URI template expression, such as `{version}`, in an instance URL, a server variable will be created automatically and displayed in the Server editor's **Server Variables** section. You can also create, edit, or reorder a list of variables. A default value must be set for each server variable.

Let's take a look at how this works, using `https://restcountries.com/v3.1/lang/german` as an example:

1. The server instance URL can be represented as `https://restcountries.com/{version}/lang`.
   Here, there's just one server variable, `{version}`, which has a default value of `v3.1`, as shown:



2. The endpoint is `/{language}`.
   Here, `language` is a path parameter, because it's a part of the endpoint path. It can't be a server variable, in this example, because it's outside the instance URL. Its default value is `german` and its type is string.

3. The full URL becomes `https://restcountries.com/{version}/lang/{language}`.
   After substitution (`v3.1` for the server variable `{version}` and `german` for the `language` path parameter), this represents the instance URL we started with, `https://restcountries.com/v3.1/lang/german`.

Here's how you can add or change a variable (or multiple variables) in a server URL:

1. Open the **Servers** tab of the service connection.
2. Click **Edit** to the right of the server instance to which you want to add a new server variable (or modify an existing one) to the instance URL.

   The Edit Server dialog is displayed.
3. In the Instance URL field, add a variable where one is needed.

   For example, you may want to replace a version number in the URL, such as "1.2" with the variable {version}. After you do this, you'll see the variable you added to the URL displayed in **Server Variables**, below the instance URL's **Description** field. The variable you added is shown in the **Name** field.
4. Enter a description in the **Description** field and set the default value in the **Default** field.

   Click **Done** if you are finished or click **+ Add Value** to add another value. To create a list with multiple values, click **Menu** and select **Create Value List**. Then, after adding values, use the or controls to set the default value or reorder the list of values. The default value will be the one at the top of the list. You can use **Delete** to discard values you don't need.
5. Add another variable in the URL, if needed.

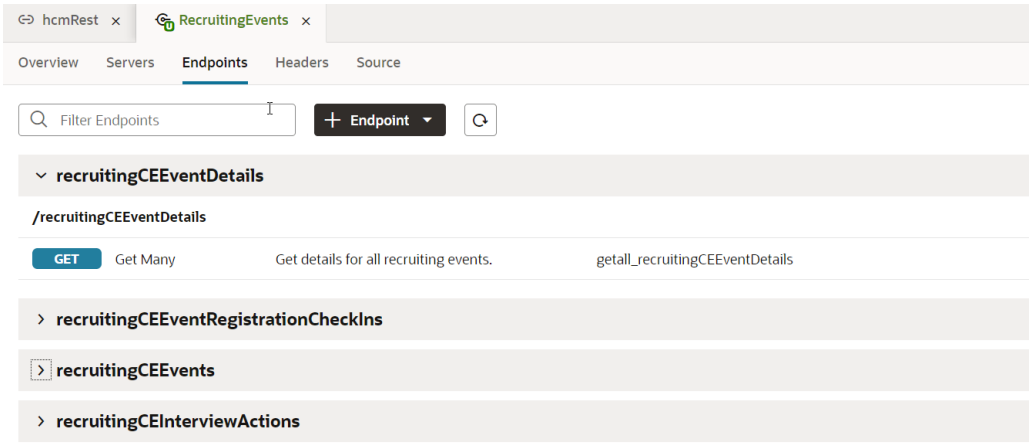
   Note that you must set a value for each variable you define. If you don't, you'll see a message indicating that a value is required. Enter the default value and click **Done**.
6. Click **Save**.

> **Tip:**
>
> There's a new **Server Variables** tab in the Endpoint editor's **Test** tab. You can use this tab to test the value of server variables when you're testing an endpoint. With a defined value list, you can effectively change the URL and test each version. The process for using this functionality is very similar to what was described here.

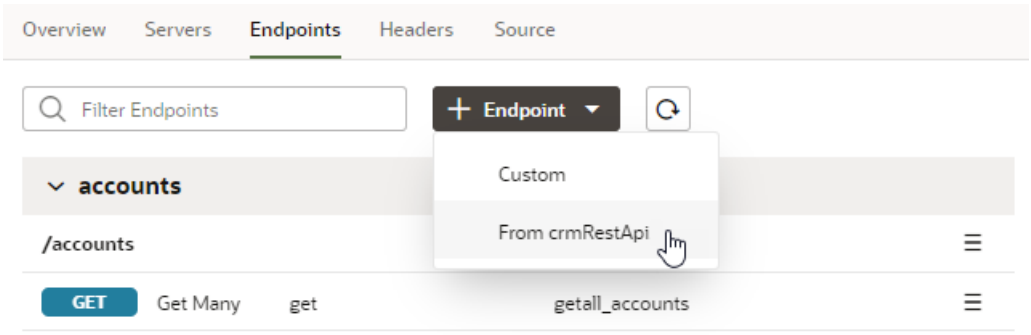
Add More Endpoints to a Service Connection

After you create a service connection, you can add more endpoints. For a static service connection, you can add endpoints from the same source or add custom endpoints.

- To add endpoints to a static service connection:
  1. Open the Endpoints tab of a static service connection:



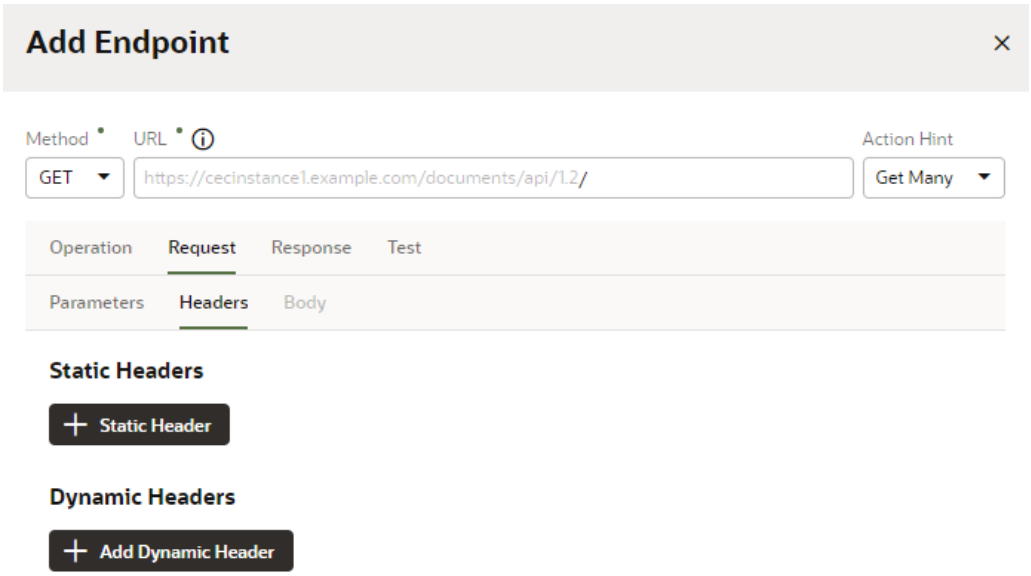
2. Click **+ Endpoint**. If your service connection was created for a service from the catalog, select **Custom** or **From *original_service*** in the drop-down list:



3. Select or define the new endpoint in the Add Endpoint dialog box. Your options depend upon the type of service connection and if you choose Custom or From *original_service*.

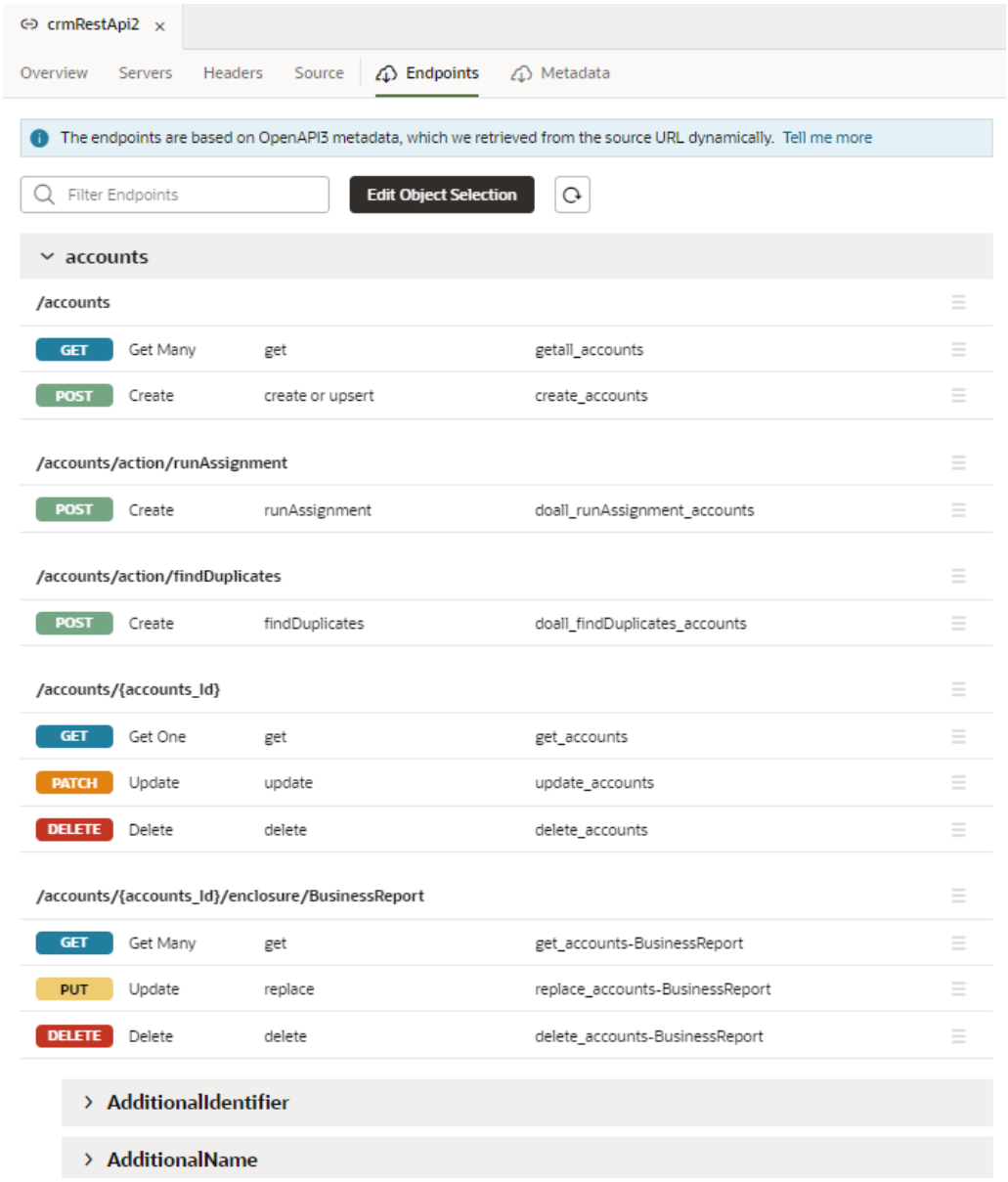
   For example, if your service connection was created for a service from the catalog and you choose to add an endpoint from the original service, you can choose from the list of endpoints available in that service.

   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, as shown here:



4. Optional: If you chose to add an endpoint from the original service, you can change how your connection's service metadata is retrieved in the **Metadata Retrieval Option** drop-down. Select **Dynamically retrieve metadata** to convert your service connection to a dynamic one. For ADF Describe-based services, select **Copy minimal OpenAPI to the application (recommended)** to leverage both static and dynamic capabilities for your connection.
5. Click **Save**.

- To add endpoints to a dynamic service connection:
  1. Open the Endpoints tab of a dynamic service connection:



2. Click **Edit Object Selection**.
3. Select one or more objects in the Add Endpoint dialog box.
4. Optional: Change how your connection's service metadata is retrieved in the **Metadata Retrieval Option** drop-down. Select **Dynamically retrieve metadata** to convert your dynamic service connection to a static one. For ADF Describe-based services, select **Copy minimal OpenAPI to the application (recommended)** to leverage both static and dynamic capabilities for your connection.
5. Click **Add**.

Edit a Server's Authentication Details

After a service connection is created, you can edit the server's authentication details from the **Servers** tab.

> **Note:**
>
> For service connections created after the 23.10 release, server authentication details must be updated from the backend associated with the service connection. For more information, see Set the Backend's Authentication Method and Connection Type

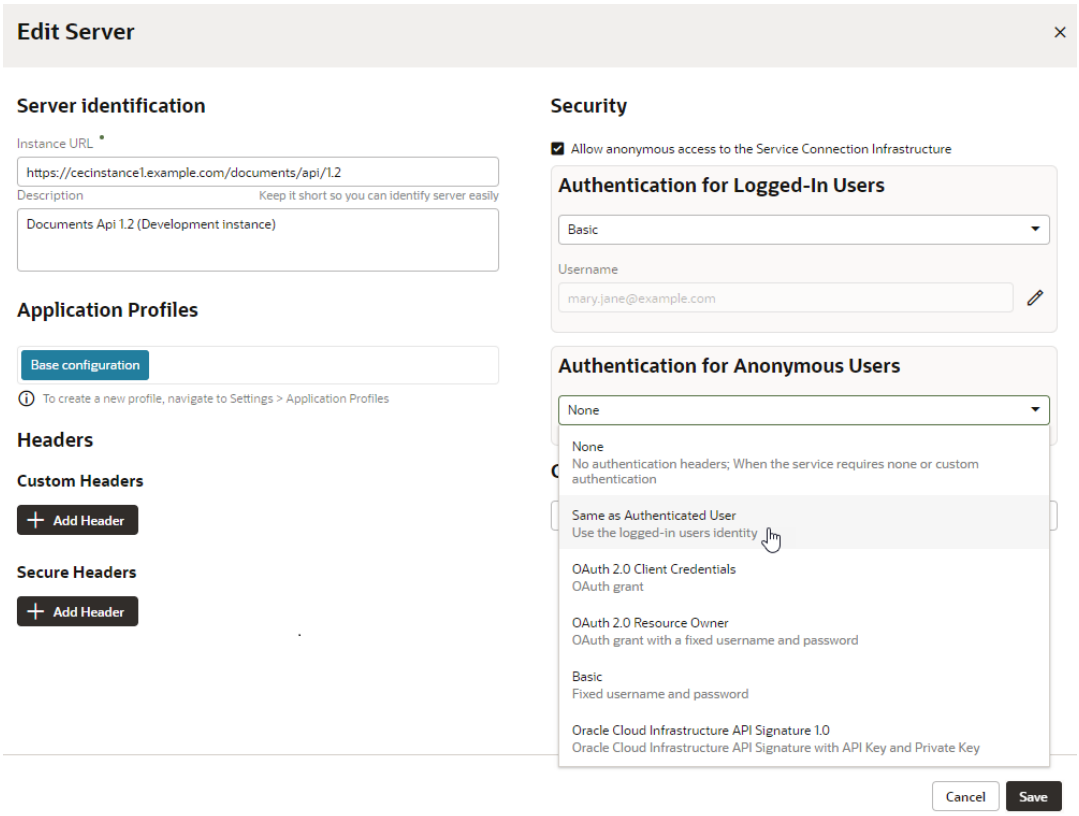
You might want to edit the authentication details when the authorization requirements of your app change, for example, you need to allow anonymous access to the service or you need to override the settings for the backend service. If you have multiple servers added to your service connection, you may need to make changes in more than one server.

To edit a service's authentication settings:

1. Open the service connection's **Servers** tab.
2. Click the **Edit**

   icon beside the server instance you want to edit.
3. In the Edit Server dialog, you can use the **Authentication** drop-down list to change the authentication method.

   If the connection is to a service in your Service Catalog, click **Override security** to display the Authentication drop-down list, where you can override settings inherited from the backend. To change the connection type inherited from the backend, click **Override Connection Type** and make changes.
4. (Optional) Select **Allow anonymous access to the Service Connection Infrastructure** and select the authentication method for anonymous users in the **Authentication for anonymous users** drop-down list.



Add Transforms to a Service Connection or an Endpoint

Transforms are JavaScript functions that you can use to alter:

- REST requests, to customize sorting, filtering, and pagination.
- REST responses, to reformat the data and retrieve paging metadata.

Transforms can be applied to a backend, child backend, service connection, or endpoint. These are known as service level transforms, and they are hierarchical. Typically, these transforms are applied to a backend so that they can be inherited by the backend's service connections and by the service connection's endpoints. Being hierarchical, transforms applied at higher levels override any transforms at lower levels. Endpoints are at the top of the hierarchy, then service connections, child backends, and lastly, backends.

For more about service level transforms, see Service Level Transforms.

Convert a Service Connection (Static to Dynamic or Dynamic to Static)

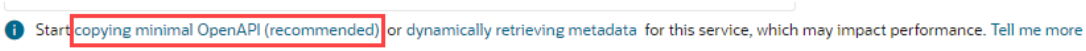
You can convert an existing service connection, either from static to dynamic or dynamic to static, to suit your application's requirements that may change over the course of its lifecycle.

Typically, it's useful to create a dynamic service connection during an application's development stage, when endpoints defined in the service metadata are still evolving. A dynamic connection fetches the service's OpenAPI3 metadata from the source URL whenever the app is opened, enabling your application to get all the updates included in the current version of the metadata. When you deem the service metadata to be stable and your application ready for production, you might want to switch the dynamic service connection to static—because while dynamic connections provide the most recent updates, they may impact

performance. A static connection, on the other hand, has better performance because the service metadata is part of your application's code.

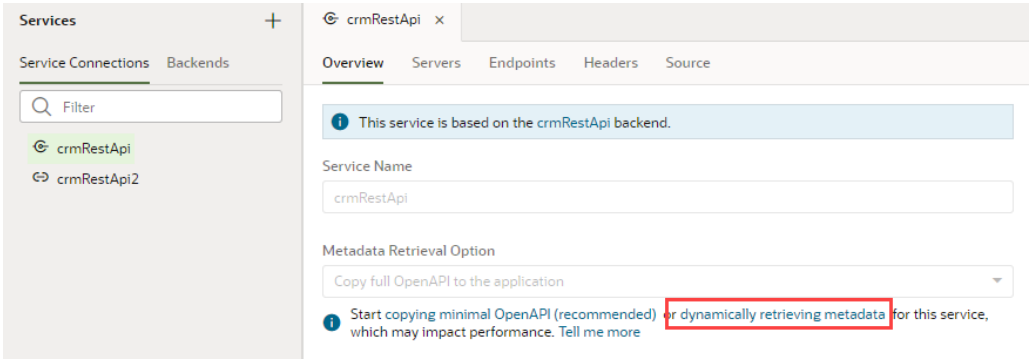
Conversely, if your application's service metadata is changing and you want to include these updates as you work on your app, you can convert your static service connection to a dynamic one. In this case, the service metadata is copied from the source URL and saved to the application's sources.

For ADF Describe services, however, always choose the recommended option for optimal runtime performance. This option copies minimal service metadata for the endpoints you select from the source URL to the application and dynamically retrieves the request/response schema when required. It provides the benefits of a static and a dynamic service connection and is recommended for both static and dynamic ADF Describe-based service connections:



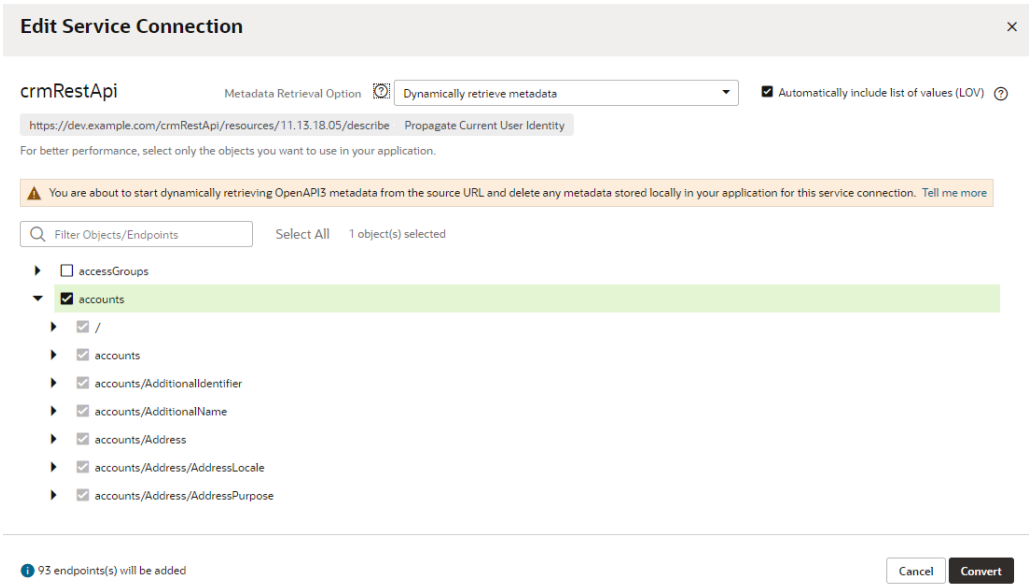
- To convert your service connection to dynamic:

  1. Open the **Overview** tab of a static service connection, then click the **dynamically retrieving metadata** link.



  2. In the Edit Service Connection dialog box, notice the **Dynamically retrieve metadata** option that's selected. You'll also see a message about the number of endpoints that will be added to the connection. Remember that dynamic service connections always include whole resources, rather than individual endpoints.

     If you want all LOVs for the selected objects/endpoints in an Oracle Cloud Applications catalog to be automatically included in the service metadata, select **Automatically include list of values (LOV)**.

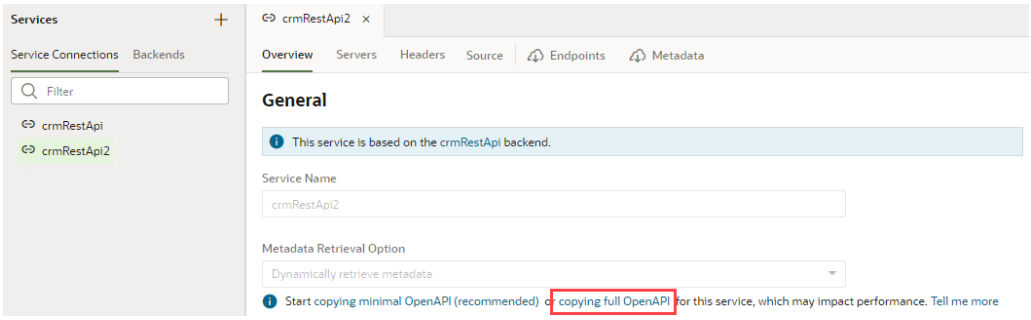


3. Click **Convert**.

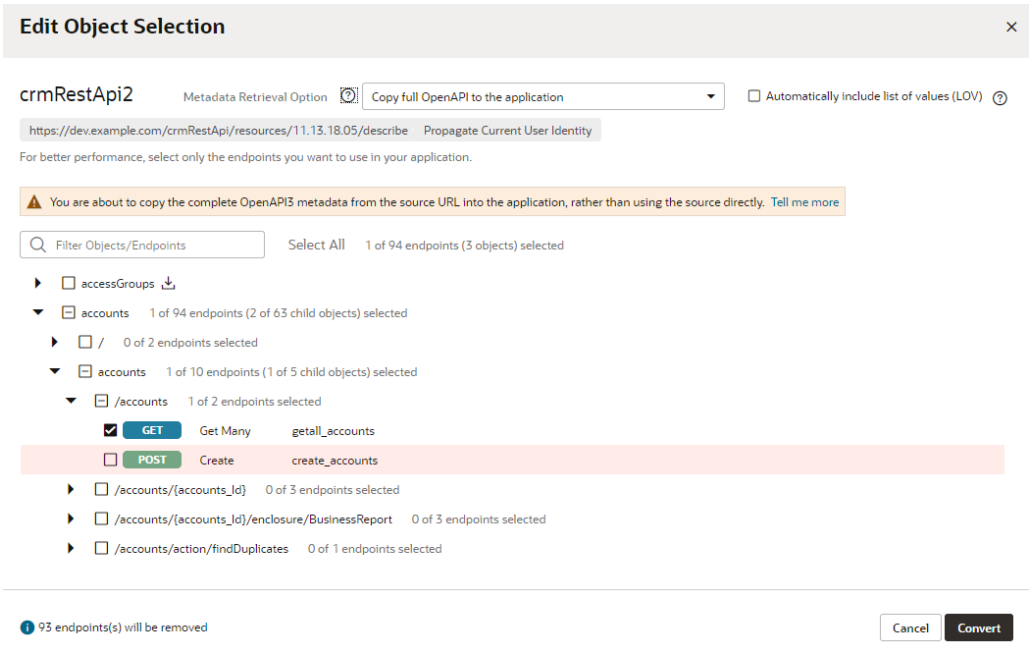
   The static service connection becomes dynamic.

- To convert your service connection to static:

1. Open the **Overview** tab of a dynamic service connection, then click the **copying full OpenAPI** link.



2. In the Edit Object Selection dialog box, select the endpoints you want to use in your application.



Notice the **Copy full OpenAPI to the application** option that's selected and a message about the number of endpoints that will be removed from the connection.

3. Click **Convert**.

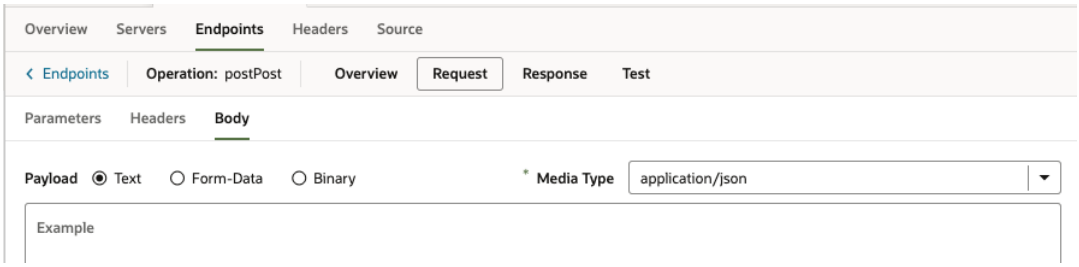
   The dynamic service connection becomes static.

Test Service Connection Responses

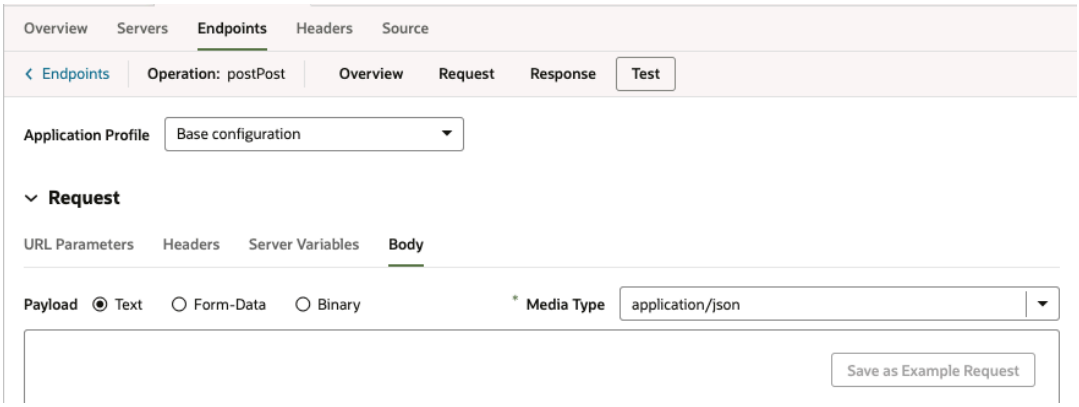
You can configure a POST/PATCH/PUT REST API with application/octet-stream or multipart/form-data format and test it from the service connection's endpoint's Test tab before proceeding with it. You can also use the form-data's schema when you call the REST endpoint.

You can access this functionality from two places, where you can use radio buttons to select the payload type to test:

- From the **Body** tab under the Endpoint editor's **Request** tab:



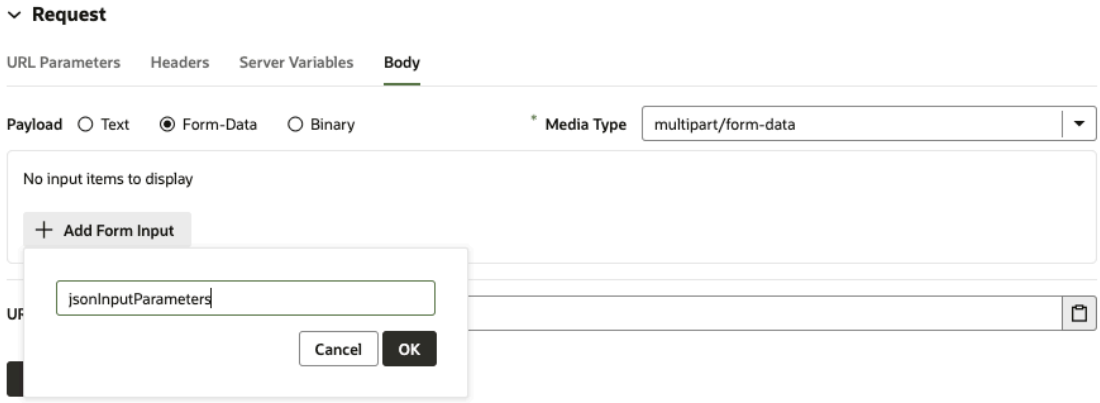
- From the Body panel under the **Test** tab:



In both screenshots, the Text payload option is selected. The body input area beneath the radio buttons looks and operates the same way it has in all prior releases.

Test Responses Using the Form Data Option

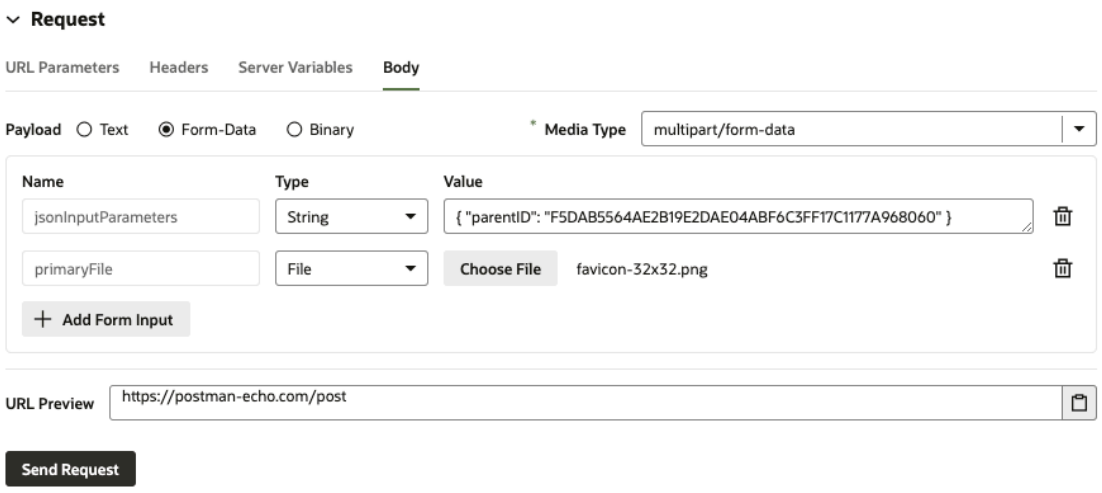
When you select the Form-Data payload option, the Media Type is automatically changed to multipart/form-data and you're presented with a form for defining input elements that provide the data values in a multipart request body:



Two types of input elements can be created:

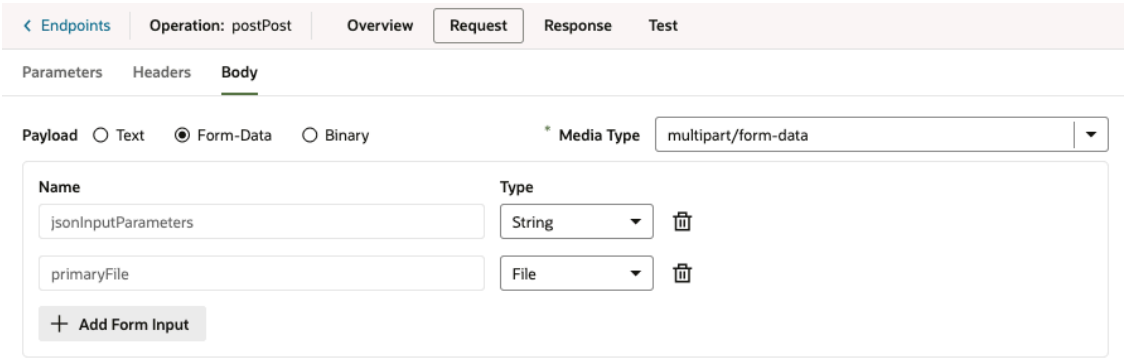
- String, for providing a user-entered text value.
- File, for selectiing a file to provide a binary data value.

As elements are added to the form, corresponding properties will be added to the media type's schema in the OpenAPI metadata:



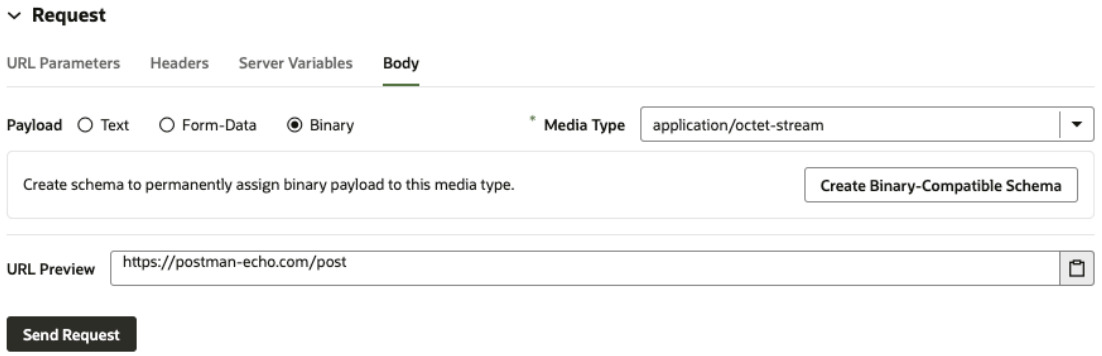
When the request is submitted from the **Test** tab, the names and values of the inputs are used to generate a FormData object that defines the multipart payload.

When form elements are created and displayed in the **Request** tab, the input elements themselves (that is, the text field or file picker) aren't rendered, because testing and submitting the form data can't be performed in this context:



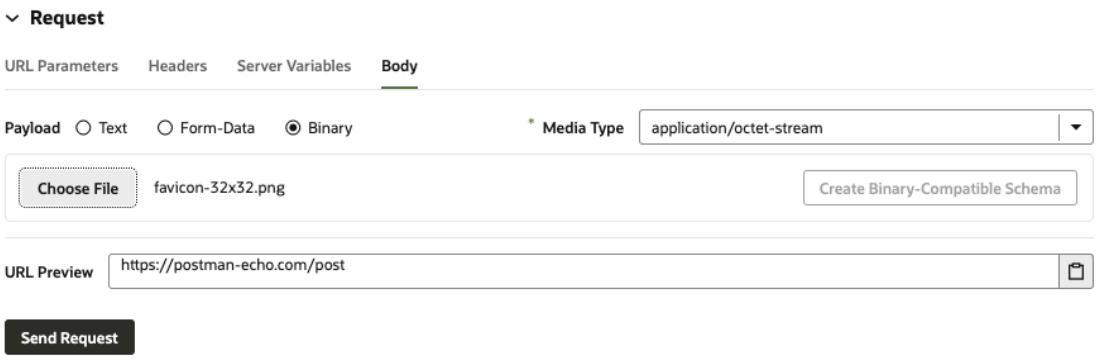
Test Responses Using the Binary Option

When you select the Binary payload option, the media type defaults to application/octet-stream, although you could manually enter any binary content type you want:

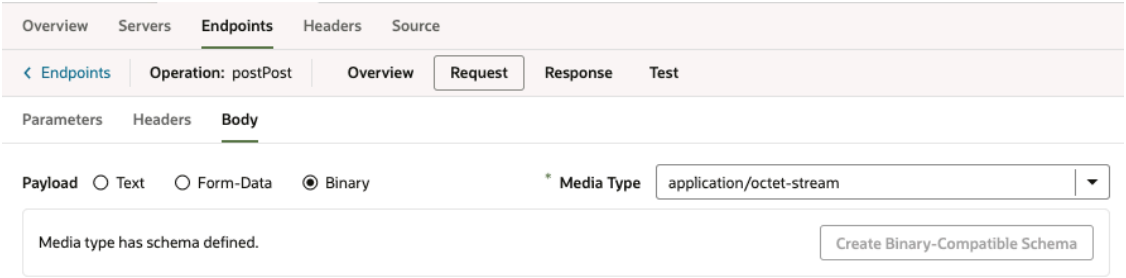


After you click the **Create Binary-Compatible Schema** button, the corresponding schema will be generated.

After the schema has been created, you can use the file picker to select the file that'll be used as binary payload when the request is submitted:

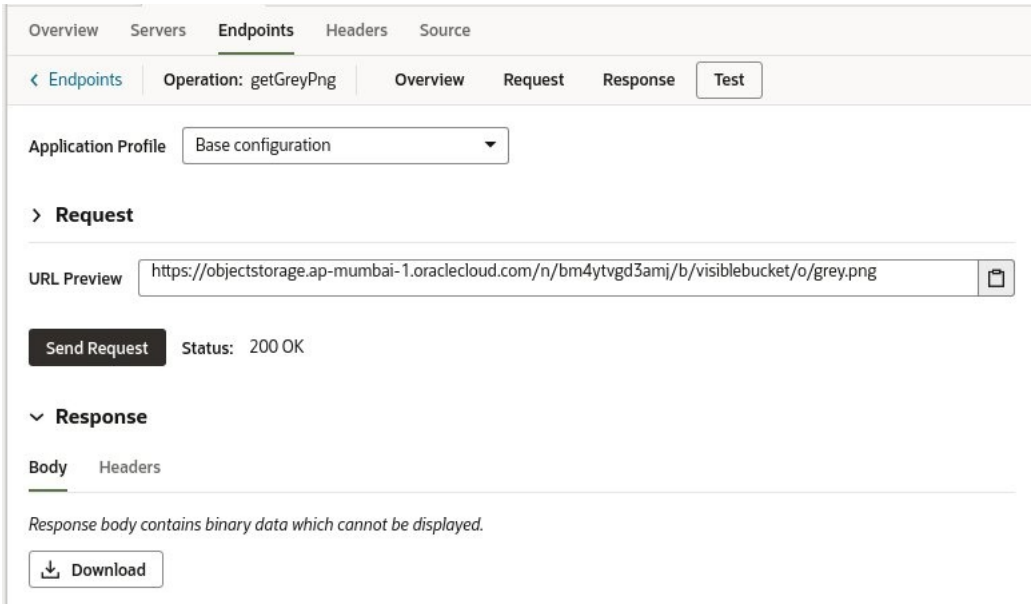


The file picker for the Binary payload type isn't shown in the Request tab:

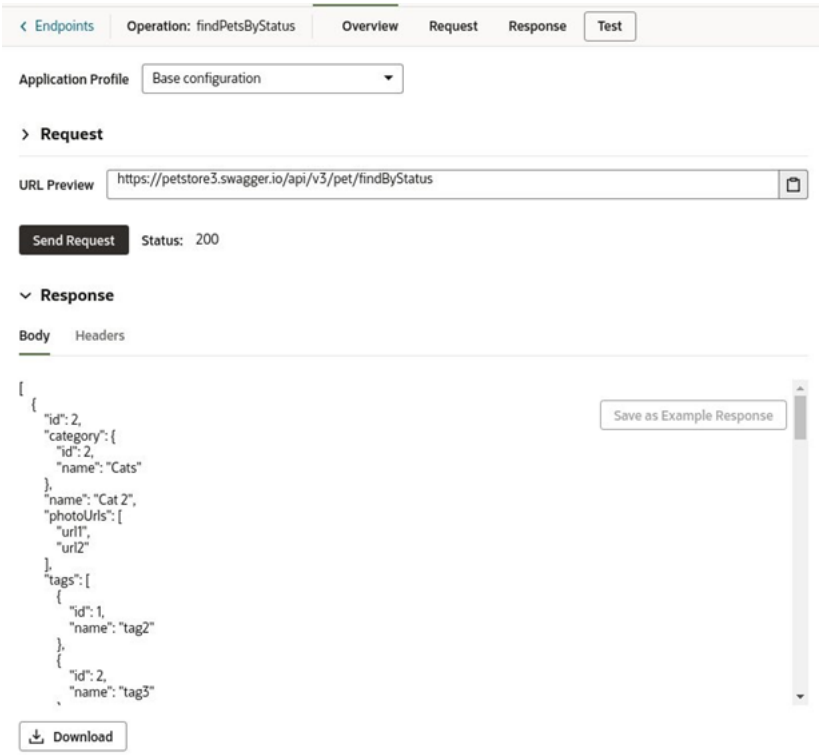


Instead, a message indicates that the schema has been created.

Service endpoints support downloading binary or text responses in the endpoint's Test tab. You can use the **Download** button to save the response body. If the response is non-text, you'll see a message that tells you that the response body contains binary data:



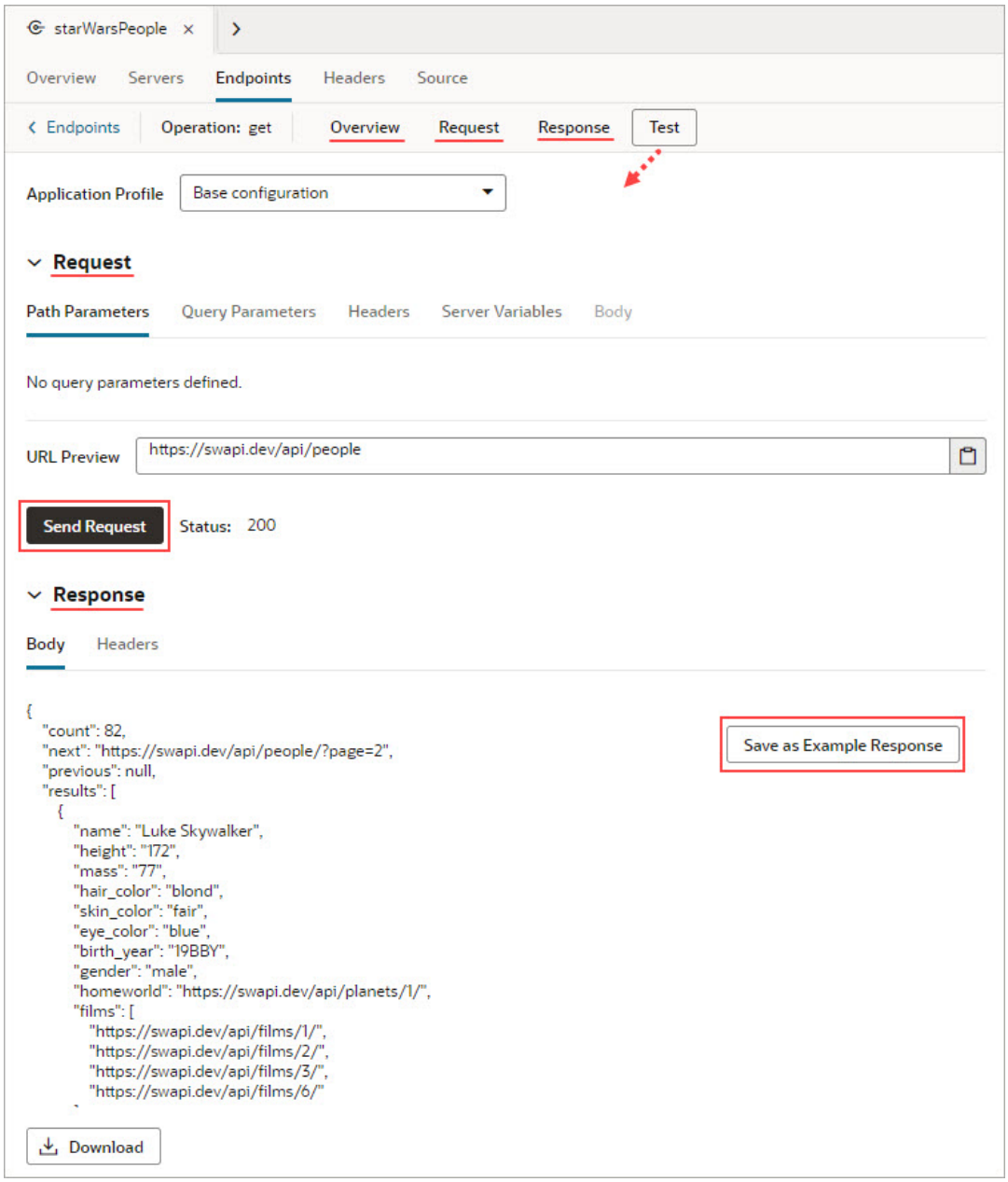
You can also use the **Download** button to download or save text responses:



Update Schema of the Request or Response

If there have been changes to the schema of the request or response, you can follow either of these three procedures to update it:

- **Option 1**: Open the **Request** or **Response** tab and supply a new example body representing the schema. A notification, near the bottom of the window, will notify you of the schema's successful update.
- **Option 2**: Open the **Source** tab, which uses the openapi3 specification and JSON format, and edit the schema (or other details) directly.
- **Option 3**: Open the **Test** tab, and in the **Request** section, ensure the URL for the request is correct. If it isn't, update the request using the **Request** tab. Click the **Send Request** button to get the response. In the **Response** section, on the **Body** tab, click the **Save as Example Response** button to update the response schema. A notification, near the bottom of the window, will notify you of the schema's successful update.



Connect to Oracle Cloud Services

Provided here are topics on how to connect to the Oracle cloud services using the various authentication methods.

Connect to Oracle Cloud Applications APIs

You can connect to Oracle Cloud Applications APIs using either identity propagation or fixed credentials.

Connect to Oracle Cloud Applications APIs With User Propagation for Authenticated Flows

To connect to Oracle Cloud Applications APIs using user propagation, the identity provider used by Oracle Cloud Applications and the IDCS used by Visual Builder must be federated.

Oracle Cloud Applications use their own identity provider (IdP), whereas Visual Builder uses IDCS only as its identity provider. The identity providers used by each service must be federated to establish a trust relationship. The list of users can be maintained in either identity provider, but not in both, but it is recommended that you use the Oracle Cloud Applications IdP, which can use SAML to provide additional integration with other identity providers.

In most cases, your Oracle Cloud Applications instance will already be federated to an IDCS instance, which also applies to the Visual Builder instance. If this isn't the case, an administrator will need to set up federation between the Oracle Cloud Applications and Identity Cloud Service instances by following the steps in this document: Oracle Fusion Applications Release 13.

If the federation has already been done, an administrator needs to go to the **Services** tab in the **Tenant Settings** page in the Visual Builder instance, and create an Oracle Cloud Applications backend service. See Add a Connection to Oracle Cloud Applications. The instance URL used for creating the backend service is the same as the Oracle Cloud Applications Base URL.

When creating the service connection, you use the following authentication method:

Authentication method	Details
Oracle Cloud Account	The default authentication and connection type for the service connection is determined by the backend service settings from which the service was added from the catalog. For example, if the service was chosen from the Oracle Cloud Applications > Sales and Service catalog, then the settings will be inherited from the backend representing Sales and Service. Typically, your service administrator will define the backend service settings in **Tenant Settings**. These settings can be overridden at the application level (with **Application Backend Settings**). You can choose to override these inherited settings by navigating to **Services** in the Navigator, then clicking **Backends**. You can: • Add or override headers • Allow anonymous access to Service Connection infrastructure. You can either enable or disable this. • Authentication for logged-in users. • Authentication for anonymous users. • Connection type: You see an **Inherited from Backend** label, which means the value defined at the backend is inherited. You'll need to choose a different connection type if you decide to override the configuration that was inherited from the backend.

To connect to Oracle Cloud Applications APIs with user propagation:

1. Open Services in the Navigator, click the **+** sign, and select **Service Connection**.
2. Click **Select from Catalog** in the Create Service Connection wizard.
3. Click **Oracle Cloud Applications** and select an endpoint. Click **Create**.
4. Test the service connection.
5. Optional: If you want to make the service connection accessible to the application's anonymous users, click **Override security**, then select the **Allow anonymous access to the Service Connection Infrastructure** checkbox in the Edit Server dialog that you invoke from the **Servers** tab. Use the **Authentication for anonymous users** drop-down list to configure an authentication type for anonymous users.

Connect to Oracle Cloud Applications APIs Not in the Catalog Using Fixed Credentials

To connect to a service in an Oracle Cloud Applications 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 Oracle Cloud Applications instance.

To create the connection, you need to have the credentials of a fixed Oracle Cloud Applications user of the instance, or an administrator will need to create the Oracle Cloud Applications user with the necessary privileges for you. When you use this user to create the connection, all requests to the Oracle Cloud Applications REST APIs will use the fixed user's credentials. The credentials of the logged-in user are not used when communicating with the service.

Oracle Cloud Applications instances are usually associated with your Visual Builder instance in the Tenant Settings. If your instance is already associated with an Oracle Cloud 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 Oracle Cloud 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 Oracle Cloud Applications instance to populate the Service Catalog.

If you do not want to change the default Oracle Cloud 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 method for the service connection:

Authentication method	Details
Basic	To use this option you need to provide the following details: • User name and Password. These can be the valid credentials of any user that has access to the Oracle Cloud Applications REST APIs.

To connect to an Oracle Cloud Applications service that is not in your catalog:

1. Open Services in the Navigator, then click the **Backends** tab.
2. Select **Oracle Cloud Applications**, then click **Override Backend**.
3. In the **Servers** tab, click the server's Edit icon.
4. In the Edit Server dialog box, provide the Instance URL of the Oracle Cloud Applications instance.
5. Select **Basic** as the authentication method, provide the username and password of the fixed user, and click **Save**.
6. In the Services pane, click the **+** sign and select **Service Connection**.
7. Click **Select from Catalog** in the Create Service Connection wizard.

   If you did not change the default Oracle Cloud Applications instance of your application, you can choose Define by Endpoint and provide the URL of the endpoint.
8. Click a service under **Oracle Cloud Applications**, then select an endpoint. Click **Create**.
9. Confirm the connection is working.
10. Optional: If you want to make the service connection accessible to anonymous users of the app, select the **Allow anonymous access to the Service Connection Infrastructure** checkbox in the Edit Server dialog that you invoke from the Servers tab. Use the **Authentication for anonymous users** drop-down list to configure an authentication type for anonymous users.

Connect to Oracle Integration APIs

You can connect to Oracle Integration APIs using identity propagation or fixed credentials.

Connect to Oracle Integration APIs Using Identity Propagation

To connect to Oracle Integration using identity propagation, the Oracle Integration and Visual Builder instances should be in the same domain. If 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 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 method should be **Oracle Cloud Account**. This will appear as the default URL and authentication method in the visual application's Backends editor from the Navigator's **Services** tab. Using

the Oracle Cloud Account authentication method provides identity propagation from Visual Builder to Oracle Integration without any additional configuration.

When creating the service connection, you use the following authentication method:

Authentication method	Details
Oracle Cloud Account	The default authentication for the connection is determined by the authentication set in the Backends editor and in the Tenant Settings. You can view the connection details in the Backends editor that you access from the Navigator's Services tab. You can choose to override these inherited settings: • Add or override headers • Allow anonymous access to Service Connection infrastructure (you can either enable or disable this) • Authentication for logged-in users • Authentication for anonymous users • Connection type: You see an **Inherited from Backend** label, which means the value defined at the backend will be inherited. Choose a different connection type if you decide to override the configuration inherited from the backend.

To connect to Oracle Integration APIs using identity propagation:

1. Open Services in the Navigator, click the **+** sign, and select **Service Connection**.
2. Select **Integration Applications** in the Backends tab and review its configuration to confirm that **Oracle Cloud Account** is selected as the default authentication method.

   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 Visual Builder and Oracle Integration are in the same domain.
3. Open Services in the Navigator, click the **+** sign, and select **Service Connection**.
4. Click **Select from Catalog** in the **Create Service Connection** wizard.

   Alternatively, you can click **Define by endpoint**, provide the URL of the sample Integration endpoint, select **Oracle Cloud Account** as the authentication method, and select your preferred connection type.
5. Select the sample integration endpoint from the list of catalog endpoints. Click **Create**.
6. Test the Service Connection.
7. Optional: If you want to make the service connection accessible to anonymous users of the app, click **Override security**, then select the **Allow anonymous access to the Service Connection Infrastructure** checkbox in the Edit Server dialog that you invoke from the Servers tab. Use the **Authentication for anonymous users** drop-down list to configure an authentication type for anonymous users.

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 method.

To access the Oracle Integration APIs using fixed credentials, the Oracle Integration and 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 Backends tab from the Navigator's Services tab and override the tenant-level settings for Integrations and select the authentication method you want to use instead of the default Oracle Cloud Account method.

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 methods are the same in both cases.

If you want to use OAuth 2.0 Resource Owner Password as the authentication method, 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 methods:

Authentication method	Details
Basic	To use this option you need to provide the following details: • User name 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 • User name 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

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 method.

To connect to Oracle Integration APIs using fixed credentials:

1. Open Services in the Navigator, click the **+** sign, and select **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 Backends 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 methods and provide the authentication details.

If you chose a service from your Service Catalog, you can override the default authentication settings in the Edit Server dialog that you invoke from the Servers tab of the service connection.

5. Test the service connection.
6. Optional: If you want to make the service connection accessible to anonymous users of the app, select the **Allow anonymous access to the Service Connection Infrastructure** checkbox in the Edit Server dialog that you invoke from the Servers tab. Use the **Authentication for anonymous users** drop-down list to configure an authentication type for anonymous users.

Connect to Oracle Cloud Infrastructure Process Automation APIs

You use Oracle Cloud Infrastructure Process Automation (also know as Process Automation and OCI Process Automation) APIs to develop, automate and monitor your business processes. In this topic, it's explained how you can create a Process Automation backend, if one hasn't been made available to you, and how you can create a service connection to a Process Automation process in a Process Application using the Process Automation catalog and the Create Service Connection wizard. For more about Process Automation, check here.

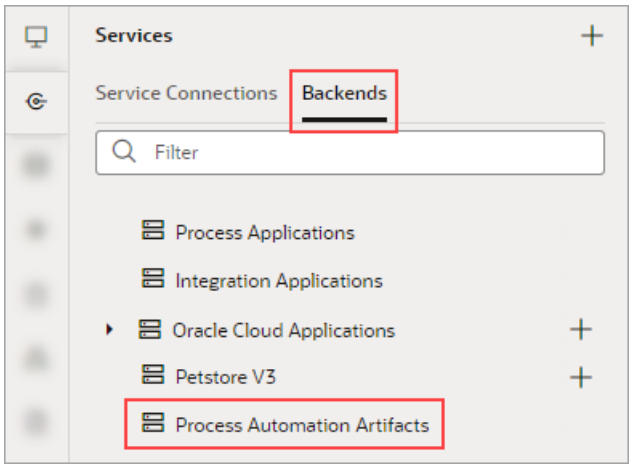
If you don't have an OCI Process Automation instance, an administrator needs to provide one, following the instructions in Provision and Manage Oracle Cloud Infrastructure Process Automation Instances.

You can connect to the Process Automation APIs using both Identity Propagation or Fixed Credentials:

Use Case	Prerequisites	Authentication method	Details
Connect to Process Automation REST APIs using Identity Propagation	• The Process Automation instance is created in the same IDCS stripe or OCI IAM Identity domain as Visual Builder. • Full scope of the Process Automation instance from its IDCS/OCI IAM is available.	OAuth 2.0 User Assertion	To use this option you need to provide the following details: • Client ID and secret kept blank • Token URL kept blank • Scope from IDCS/OCI IAM, typically `https://<opa_instance>/process`
	The Process Automation instance has been co-provisioned with Oracle Integration Cloud and Visual Builder.	Oracle Cloud Account	When the Process Automation instance is co-provisioned with Oracle Integration Cloud and Visual Builder, a backend is automatically created in the Visual Builder Tenant Settings with "Oracle Cloud Account" authentication.

Use Case	Prerequisites	Authentication method	Details
Connect to Process Automation REST APIs using Fixed Credentials	Client ID, secret, and the full scope of the Process Automation instance from its IDCS/OCI IAM is available.	OAuth 2.0 Resource Owner Password	To use this option you need to provide the following details: • Client ID and secret as per IDCS/OCI IAM • Token URL, typically `https://<idcs_or_iam>/oauth2/v1/token` • Username and password of a valid user having access to the Process Automation REST APIs • Scope from IDCS/OCI IAM, typically `https://<opa_instance>/process`

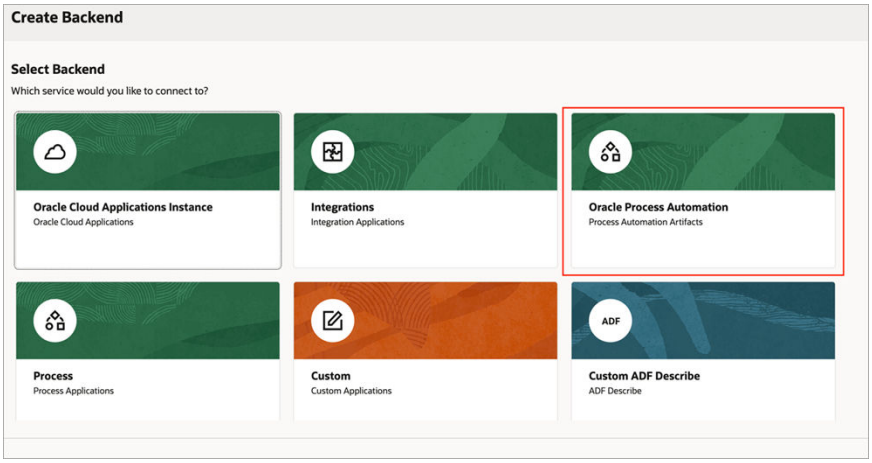
First, to create a service connection to a Process Automation API, you need a Process Automation backend representing a provisioned Process Automation instance. If you have a Process Automation backend, you can see it on the Backends tab of the Services pane (shows as Oracle Process Automation or Process Automation Artifacts):



If you need to create a Process Automation backend, it can be done at the tenant level, making it available to all applications, or at the application level, making it only available to the current application:

- To create a Process Automation backend at the tenant level, you need to have someone with administrator privileges create a tenant-level backend, as explained in Visual Builder - Understanding Backends and Servers. This is recommended, as it'll be available to all of the applications. Also, refer to the table given above for details about using the Identity Propagation or Fixed Credentials authentication method.

- To create a Process Automation backend at the application level:
  1. On the Services pane, click the add (+) button on the pane's top-right and select the **Backends** option to create a new backend using the Create Backend wizard.
  2. On the Select Backend step of the wizard, select the **Oracle Process Automation** tile and proceed to provide the backend's details.



For more about how to create a backend, see Create a Backend, and refer to the table given above for details about using the Identity Propagation or Fixed Credentials authentication method.

Once the Process Automation backend is available, you can use the Process Automation catalog to create a service connection to a Process Automation process in a Process Application, using a wizard. To do so, see Create Service Connections from the Oracle Cloud Infrastructure Process Automation Catalog.

Connect to Process Cloud APIs

You can connect to Process Cloud APIs using identity propagation or fixed credentials.

> **Note:**
>
> Oracle Process Cloud Service, which is included in the Enterprise edition of Oracle Integration Gen 2, is deprecated. See Process Features for details.

Connect to Process APIs Using Identity Propagation

To connect to Process APIs using identity propagation, the Process and 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 Visual Builder to Process without the need for any additional configuration. No CORS configuration is required because the instances are in the same domain.

To connect to Process APIs using identity propagation:

1. Open Services in the Navigator, click the **+** sign, and select **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 methods.

5. Choose an appropriate connection type, such as **Dynamic, the service supports CORS** to take advantage of direct flows.

6. Test the service connection.

7. Optional: If you want to make the service connection accessible to anonymous users of the app, select the **Allow anonymous access to the Service Connection Infrastructure** check box in the Edit Server dialog that you invoke from the Servers tab. Use the **Authentication for anonymous users** drop-down list to configure an authentication type for anonymous users.

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 method.

For access to Process APIs using fixed credentials, the Process and 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 method for the service connection:

Authentication method	Details
Basic	To use this option you need to provide the following details: • User name and Password. These can be the valid credentials of any user that has access to the Process API.
OAuth 2.0 Client Credentials	To use this option you need to provide the following details: • Client ID and Secret. This is from the IDCS of Process • Token URL. The instance token URL from IDCS. • Scope. This is from the IDCS of Process.
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 Process • Username and Password. These can be the valid credentials of any user that has access to the Process REST endpoint. • Token URL. The instance token URL from IDCS. • Scope. This is from the IDCS of Process.

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 methods.

To connect to Process APIs using fixed credentials:

1. Open Services in the Navigator, click the **+** sign, and select **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 methods and provide the authentication details.

5. Test the service connection.
6. Optional: If you want to make the service connection accessible to anonymous users of the app, select the **Allow anonymous access to the Service Connection Infrastructure** check box in the Edit Server dialog that you invoke from the Servers tab. Use the **Authentication for anonymous users** drop-down list to configure an authentication type for anonymous users.

Connect to Oracle Content Management REST APIs

You can connect to Oracle Content Management REST APIs using identity propagation or fixed credentials.

Connect to Oracle Content Management REST APIs Using Identity Propagation

Oracle Content Management and Visual Builder are not provisioned together so the service administrator needs to perform the following steps in IDCS to add Oracle Content Management as a resource of the Visual Builder application.

This adds Oracle Content Management 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 Oracle Content Management 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 Oracle Content Management instance "/documents" endpoint and save the application. The added scope should now be visible in the Application in the Resources list.
   If other Oracle Content Management 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 Oracle Content Management with identity propagation. If you don't have access to IDCS, the administrator will need to provide you with the Oracle Content Management scope that you need to enter in the Authentication tab.

When creating the service connection, you use the following authentication method for the service connection:

Authentication method	Details
OAuth 2.0 User Assertion	To use this option you need to provide the following details: • **Client ID** and **Secret**: Leave blank. • **Token URL**: Leave blank. • **Scope**: The scope added from IDCS that corresponds to the Oracle Content Management instance. This is the full scope, including "/documents".

To connect to Oracle Content Management:

1. Open Services in the Navigator, click the **+** sign, and select **Service Connection**.
2. Click **Define by Endpoint** in the Select Source step of the Create Service Connection wizard.
3. Select the HTTP method and enter the URL of the Oracle Content Management endpoint.

For example, the URL of your endpoint might be similar to the following: `https://<Oracle_Content_Management_instance>/documents/api/<VERSION>/folders/{folderId}`

4. In the Authentication section of the **Server** tab, select **OAuth 2.0 User Assertion** as the authentication method.
5. In the **Scope** field, enter the scope corresponding to the Oracle Content Management instance that was added in IDCS.

   The **Client Id**, **Secret**, and **Token URL** fields are blank.
6. Test the service connection.
7. Optional: If you want to make the service connection accessible to anonymous users of the app, select the **Allow anonymous access to the Service Connection Infrastructure** checkbox in the Edit Server dialog that you invoke from the Servers tab. Use the **Authentication for anonymous users** drop-down list to configure an authentication type for anonymous users.

Connect to Oracle Content Management REST APIs Using Fixed Credentials

To connect to Oracle Content Management using fixed credentials, the Oracle Content Management and 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 Oracle Content Management, you need to provide a user name and password that are valid in IDCS.

If you want to use OAuth flows for authenticating your connection to Oracle Content Management, 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 Oracle Content Management functionality (for example, Social), you'll need the URL that corresponds to its scope. If you don't have access to the IDCS instance used by Oracle Content Management, you'll need to request the details from a user with access to the instance.

To retrieve the details of the Oracle Content Management application from IDCS:

1. In the Configuration tab for the Oracle Content Management application in IDCS that represents the Oracle Content Management instance, expand the General Information panel and note the Client ID and Client Secret.
   The name of the Oracle Content Management 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 Oracle Content Management instance "/documents" endpoint. The URL will be similar to `https://<primary audience url>/documents`.
   If other Oracle Content Management functionality (for example, Social) is required, you'll need to note the URL for the corresponding scope.

When creating the service connection, you can use one of the following authentication methods for the service connection:

Authentication method	Details
Basic	To use this option you need to provide the following details: • **User name** and **Password**: Valid credentials of any user from IDCS.

Authentication method	Details
OAuth 2.0 Resource Owner Password	To use this option you need to provide the following details: • **Client ID** and **Secret**: Get from the Oracle Content Management application in IDCS. • **User name** and **Password**: Valid credentials of any user with access to the Oracle Content Management REST endpoint. • **Token URL**: The URL for the endpoint used to obtain an access token from IDCS, in the form of `<base_URL_corresponding_to_Oracle_Content_Management_in_IDCS>/oauth2/v1/token`. • **Scope**. Get from the Oracle Content Management application in IDCS.

If you don't have access to IDCS, get the Oracle Content Management application details from an administrator.

To connect to Oracle Content Management using fixed credentials:

1. Open Services in the Navigator, click the **+** sign, and select **Service Connection**.
2. Click **Define by Endpoint** in the Select Source step of the Create Service Connection wizard.
3. Select the HTTP method and enter the URL of the Oracle Content Management endpoint.

   For example, the URL of your endpoint might be similar to the following: `https://<Oracle_Content_Management_instance>/documents/api/<VERSION>/folders/{folderId}`
4. In the Authentication section of the Server tab, select **OAuth 2.0 Resource Owner Credentials** as the authentication method.
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 Oracle Content Management application in IDCS.
6. Test the service connection.
7. Optional: If you want to make the service connection accessible to anonymous user of the app, select the **Allow anonymous access to the Service Connection Infrastructure** checkbox in the Edit Server dialog that you invoke from the Servers tab. Use the **Authentication for anonymous users** drop-down list to configure an authentication type for anonymous users.

Access Data in an Existing Oracle Database Using ORDS

If want to expose data from existing Oracle databases, you'll need to expose the database tables and views through a REST layer. To do this, you can use your own database schema to create business objects and service connections in Visual Builder. Alternatively, you can connect to Oracle REST Data Services (ORDS), which makes it easy to expose your tables and other database objects as REST services.

To expose your data through ORDS, you must first enable ORDS access for tables and views in the schema associated with your database connection. If you're using Oracle SQL Developer, you can enable REST access either to the schema or to the individual table or view. See Automatically Enabling REST Access to a Schema, Table, or View (AutoREST) in *Oracle SQL Developer User's Guide* for more information.

> **Note:**
>
> If your database is in a private subnet or behind a firewall, your ORDS endpoint must have a public IP address to be accessible by Visual Builder.

Choose Between Business Objects and ORDS

Business objects are fully integrated into Visual Builder, making it fairly easy to create business rules, relationships, REST services, and so on. In addition, you can apply the same security roles defined in Visual Builder to business objects to protect them. Finally, business objects automatically provide REST API transforms (`vb/BusinessObjectTransforms`) for filtering, sorting, and pagination.

With ORDS, hosting and creating the REST service layer is done *outside* of Visual Builder. By enabling ORDS on your database, you can use the ORDS application—along with any other supported applications needed— to create REST services. (See Getting Started with Oracle REST Data Services for examples of using your version of ORDS with other Oracle applications). Unlike business objects, ORDS uses its own security roles and privileges, and you must create your own filter, sort, and pagination transform functions. The benefit of ORDS, however, is that it lets you take full advantage of Oracle DB SQL and PL/SQL capabilities to create a middle layer with the business logic you need. ORDS also enables you to use more database features, such as BLOB/CLOB fields, functions, and procedures.

Map ORDS Endpoints in Visual Builder

When your database table or view is REST-enabled, you can use either the individual endpoint or the OpenAPI catalog to map the ORDS endpoint as a REST service in your application.

- To map the individual endpoint, you need the endpoint URL, which may be something like `https://serverName/ords/schemaName/tableName/`. Now create a service connection from this endpoint. You may need to define multiple endpoints that map to each REST operation (GET, POST, DELETE, etc.)
- To map the OpenAPI specification, which allows you to create many endpoints all at once, add `open-api-catalog` to the object's URL, which may be something like `https://serverName/ords/schemaName/open-api-catalog/tableName/`. Now create a service connection using the OpenAPI specification. See this blog for a demo.

> **Note:**
>
> Make sure CORS settings for the ORDS endpoint and the service connection match; in other words, both ORDS and Visual Builder must have CORS enabled, or neither must have CORS enabled. For an example of how to configure an ORDS endpoint to support CORS, see About Browser Origins in ORDS documentation.

Sort, Query, and Paginate ORDS Data

To filter, sort, and paginate the data returned by the ORDS backend, you'll need to provide a transform function as part of your service connection definition in Visual Builder. Find sample code for the transform function in the Visual Builder cookbook's ORDS integration recipe.

Connect to ORDS APIs Using Fixed Credentials

To secure access to the ORDS endpoint using fixed credentials, you use OAuth 2.0 Client Credentials for authentication.

When creating the service connection, you can use the following authentication method for the service connection:

Authentication method	Details
OAuth 2.0 Client Credentials	This is the recommended authentication option. To use this option you need to provide the following details: • Client ID and Secret. From ORDS • Token URL. From ORDS, for example, `https://example.com/ords/ordstest/oauth/token` • Scope. This is blank.

1. Before creating a connection to ORDS, a role and privilege to protect your REST service must be created and the OAuth client must be registered in the ORDS service. The following steps briefly describe this process; for detailed information, see Protecting and Accessing Resources in *Developing Oracle REST Data Services Applications*.

   a. 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;
   ```

   b. 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 shown here:

   ```
   curl -i https://example.com/ords/ordstest/examples/employees/
   HTTP/1.1 401 Unauthorized
   Content-Type: text/html
   Transfer-Encoding: chunked

   <!DOCTYPE html>
   <html>
   ...
   </html>
   ```

c. 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;
```

d. Grant this newly created client the required role:

```
begin oauth.grant_client_role(
      p_client_name => 'Client Credentials Example',
      p_role_name => 'HR Administrator' );
 commit;
end;
```

e. Check the registered client ID and secret:

```
select client_id,client_secret from user_ords_clients where name =
'Client Credentials Example';
```

2. Now create a connection to ORDS using fixed credentials:

a. Open Services in the Navigator, click the **+** sign, and select **Service Connection**.

b. In the Create Service Connection wizard, click **Define by Endpoint**, select the HTTP method and enter the URL of the ORDS endpoint.

You can choose **Define by Specification** and use the URL of the ORDS' OpenAPI specification for your object to define the service connection for multiple endpoints.

c. In the Authentication section of the Server tab, select **OAuth 2.0 Client Credentials** as the authentication method.

d. Provide the details for the Client Id, Secret, and Token URL fields based on your ORDS configuration.

e. Test the service connection.

Part III
Develop Applications

Take a closer look at how you design and develop web (or mobile) in Oracle Visual Builder. Your application can include *pages* and *flows*, as well as *fragments*, which are modular pieces of UI that can be reused in multiple pages.

Topics:

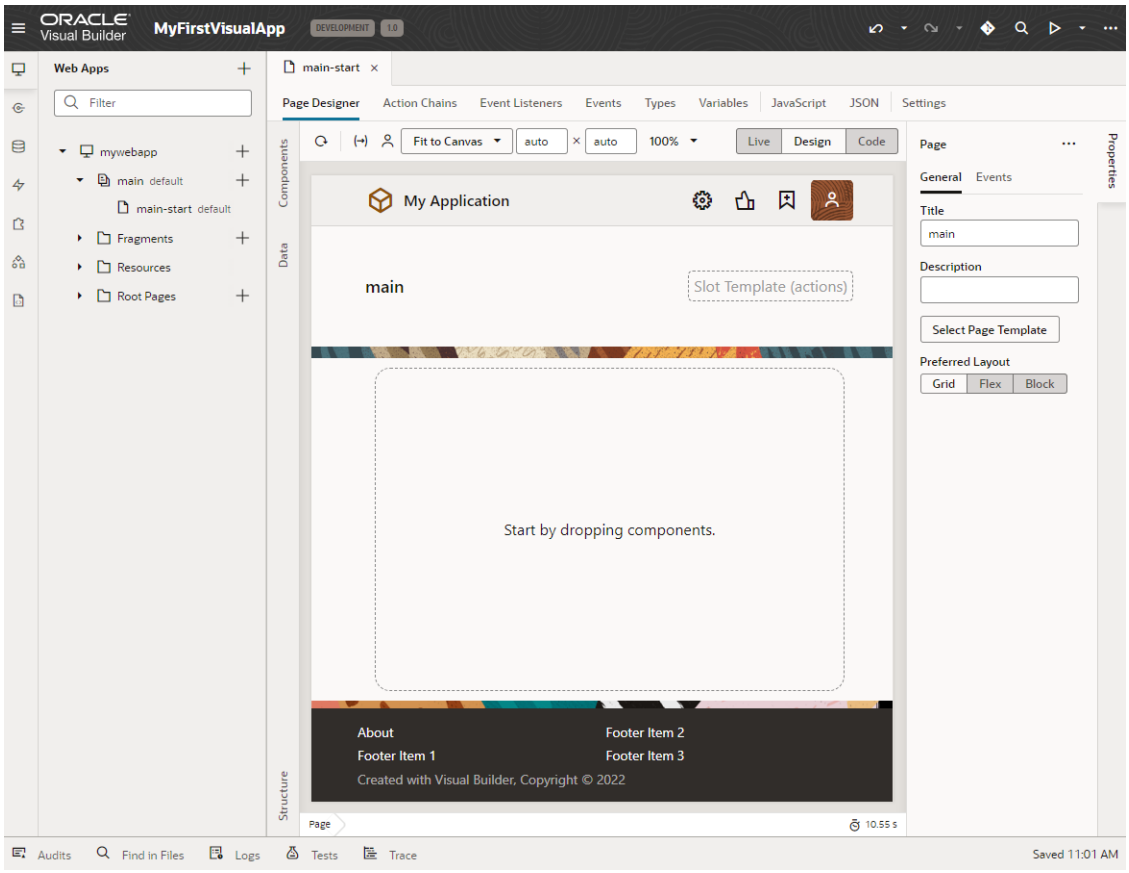
- Develop Your Application
- Work With Pages and Flows
- Work With Components
- Work with Variables and Types
- Work With JavaScript Action Chains
- Work With Events and Event Listeners
- Work With Application Resources
- Work With Fragments

6

Develop Your Application

To develop your application, you must have a visual application. The visual application provides all the resources you need to start building your application.

If you're already in the Designer (with your screen looking something like this), that means you have a visual application and can get to work:



If your screen doesn't look like this, you need to create a visual application and add web (and mobile) apps to it.

To develop your web (or mobile) application, you create and edit the building blocks that determine how your application looks and behaves. The basic building blocks of an application are user interface components, variables, action chains, page flows and page navigation, JavaScript functions, and data access through REST endpoints.

Visual Builder doesn't impose any particular order for developing your application. How you proceed is entirely up to you and determined by the way you planned your application. If you already know the structure of your objects or the data sources you want to use, you might want to start by defining business objects and service connections. Alternatively, you can start with your application's pages and create the artifacts that add functionality to your pages. For example, you might start by defining page variables and creating action chains, in addition to

positioning components on the page. All of this can be done effortlessly in the Designer. Keep reading to familiarize yourself with how apps are structured, what scopes are, and which editors you'd use to work with different artifacts.

Further, as you work on your app and progress through the app dev lifecycle:

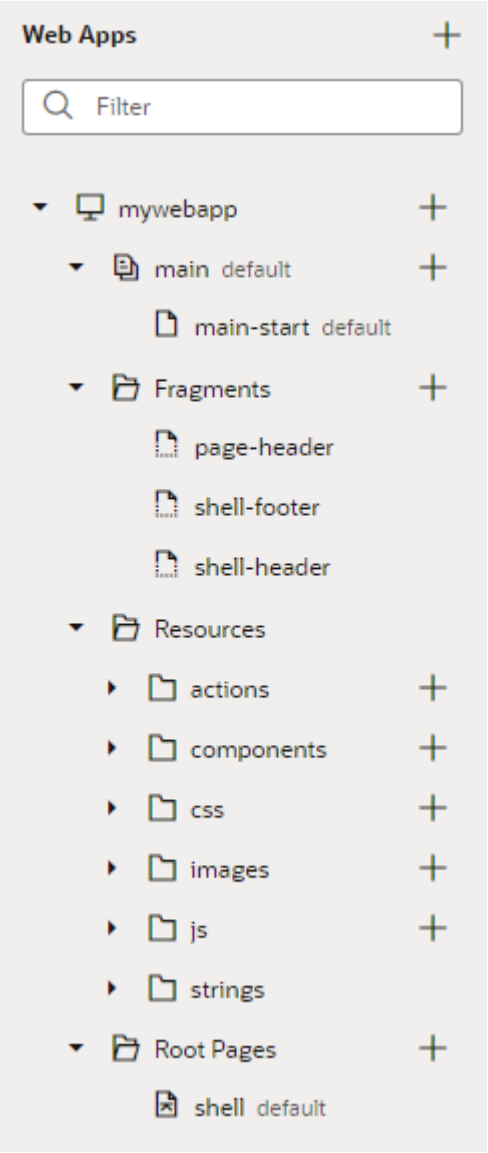
- Learn how to manage a visual application,
- Optionally, integrate your visual application with a Git repository,
- Take steps to test and debug your app,
- When you're done making changes, stage and publish your app to deploy the app's resources to the Visual Builder runtime environment.

How Are Applications Structured?

Your visual application can contain multiple web (and mobile) applications. Each app's structure and the artifacts required to add functionality to it are created by default when you create or import the app.

Open **Web Applications** (or **Mobile Applications**) in the Navigator to see a visual representation of your app's structure and to navigate to its artifacts. The artifacts are represented as one object in the Navigator, 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, its variables and event listeners are described in the flow artifact’s JSON file, while functions are defined in the artifact’s JavaScript file.

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 Designer.



Here's a look at the nodes and artifacts you might see when a web (or mobile) application is open in the Navigator:

Item	Description
application	All the artifacts of an application are grouped under the application node in the Navigator. You will see an application node for each app in your visual application. For example, if your visual application has three web apps, you will see three application nodes in the Web Apps 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. It 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 Which Editor Do I Use?.

Item	Description
flow	All individual flows in your app are grouped under a flow node. Your app can have multiple flows (including sub-flows that are grouped under a `flows` node) 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 web apps, the default flow is named `main` when you use the None navigation style. For imported mobile apps, the default flow is named `item-1`. The default flow is badged `default` for easier identification. A flow consists of a flow artifact and the pages within the flow. The descriptions of the flow are contained in the *`flowname`*`-flow.json` and *`flowname`*`-flow.js` source files. See Create and Manage Flows.
page	All pages in your application are grouped under a flow. 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. The default page in a flow is badged `default` for easier identification. See Work With Pages and Flows.
Fragments	(Only web apps) The fragments node contains artifacts for modular pieces of UI that can be reused in multiple pages of an application. Each fragment is developed just like a page and contains its own HTML file for elements, JavaScript file for functions, and JSON file for metadata. See Work With Fragments.
Resources	The resources node contains resources available to your application, such as images, style sheets (`css`) and translation files (`strings`). See Work With Application Resources.
Root pages	The root pages node contains one or more root page artifacts which describe elements such as a header or title area, a navigation toolbar, and a footer. An application typically contains one root page artifact. By default, a root page artifact named `shell` is created for a web application. For imported mobile apps, the root page artifact is named `app`. See Customize Your App's Root Page.

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

Which Editor Do I Use?

The Designer has dedicated editors for each of the building blocks used to develop your application. Each editor provides an easy-to-use interface for editing the files that describe your 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 Designer provides editors to help you perform these tasks, so you don't need to edit the source code of the HTML, JavaScript, and JSON files used to describe the layout and behavior. But if you want to edit the JSON files directly, you have access to a JSON editor.

The Designer's editors 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 Designer. You switch between editors for the artifact by selecting the appropriate tab in the artifact. The tabs that appear for the artifact depend on the artifact. A **Diagram** tab appears for application and flow artifacts while a **Page Designer** tab appears when you open a page. A number in the tab indicates the number of incidences; for example, the `hrapp` application in this image includes two action chains and one event listener (in addition to types and variables), while the `main-start` page includes four action chains, four event listeners, and a single type:

Here's an overview of the editors available in the Designer:

Tab	Description
Diagram	The Diagram view, shown only for application and flow artifacts, provides a visual representation of how an application is structured in terms of flows and pages. It visualizes an application's root pages, flows, and pages within a flow to provide a quick look at default pages, navigation, and more. For an application artifact, this view displays the application's root page as well as a hierarchical view of the artifact's flows and subflows. For a flow artifact, this view displays the pages contained in the flow and navigation between those pages. See Work With the Diagram View.
Page Designer	The Page Designer, shown only for page artifacts, lets you compose the layout of a page. It contains a canvas that represents the page layout and a palette with page components that you drag onto the canvas to add to the page. It also includes a Properties pane that you use to specify the properties of the page's components and to open Quick Starts. See Use the Page Designer and How Do Quick Starts Work?. If you're looking to design and develop pages based on your data sources, the Data palette is a handy option. It lets you work with REST endpoints that expose data in your application, letting you drag and drop them onto your canvas and display their data in suitable UI components. See The Data Palette.
Action Chains	The Action Chains editor displays a list of the action chains that are defined within the scope of the artifact. You can use this editor to create new action chains and to open action chains in the editor. See About Action Chains.
Event Listeners	The Event Listeners editor displays a list of lifecycle events that are defined for the artifact, the type of event, and the action chain that the event starts. You can use the editor to create new events. See Start an Action Chain From a Lifecycle Event.
Events	The Events editor displays a list of custom events that are defined for the artifact, the type of event, and the action chain that the event starts. You can use the editor to create new events and action chains. See Start an Action Chain From a Lifecycle Event.
Types	The Types editor displays the data types that are defined within the scope of the artifact. You can use the Types editor to create and edit types. See What are Variables and Types?.
Variables	The Variables editor displays the variables and data types that are defined within the scope of the artifact. You can use the Variables editor to create and edit variables. See What are Variables and Types?.

Tab	Description
HTML	The HTML editor displays the code for an application's `index.html` file when the application artifact is open in the Designer. Use Code view in the Page Designer to view and edit the HTML of pages in apps. See Work With Code.
JavaScript	The JavaScript editor contains a code editor for editing the artifact's JavaScript functions. By default, application artifacts use a file named `app-flow.js`, flow artifacts use a file named `<FLOWNAME>-flow.js`, and page artifacts use a file named `<PAGENAME>-page.js` to define their JavaScript functions. See Work With JavaScript.
JSON	The JSON editor contains an editor for editing the JSON file that contains the artifact's metadata, including descriptions of variables and action chains. By default, application artifacts use a file named `app-flow.json`, flow artifacts use a file named `<FLOWNAME>-flow.json`, and page artifacts use a file named `<PAGENAME>-page.json`. See Work With JSON.
Settings	The Settings editor contains tabs for editing an artifact's settings. The options available depend upon the artifact. The Settings editor for flow and application artifacts includes a Security tab, which you can use to set the artifact's security. You can use the Imports tab to import components, custom modules, and CSS files for application artifacts, flows, and pages, and the Translation tab to create additional translation bundles for application artifacts, flows, and pages. See Manage Custom Component, CSS, and Module Imports and Create Translation Bundles.

What Are Scopes?

Scope refers to how and where certain artifacts—like variables, action chains, root pages, and more—can be used within Visual Builder.

In a nutshell, where you define something determines where it can be referenced; in other words, artifacts can be scoped at different levels:

- Visual application scope: Artifacts defined through the far left ribbon in the Navigator—Layouts, service connections, business objects, components, and so on—are available to all apps within the visual application.
- App scope: Artifacts are available to all the page flows and pages in a standalone app. Examples: Resources at the app level, as well as action chains, events, event listeners, types, and variables defined at the app or fragment level.
- Flow scope: Artifacts are available only to the flow (and pages) in which they are defined.
- Page: Artifacts are defined at the page level, and thus are available to only that page.

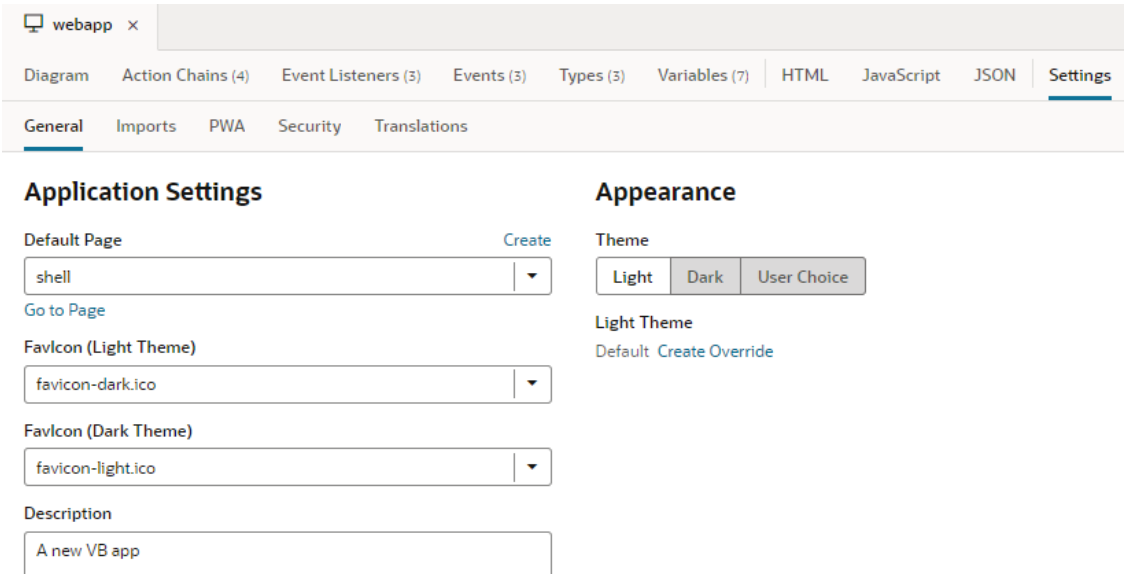
You can also define settings that are scoped to various entities, which control how the entity looks and behaves:

- Visual Application Settings
- App Settings
- Page Settings
- Flow Settings

Manage App Settings

Because an app can include multiple pages and flows, settings at the app level apply to all the pages and flows within it.

To configure an app's settings, open the app, then click **Settings** to open the Settings editor:



Here's how you can use the different app-level settings:

Tab/Setting	Description
General tab	Manage general app settings as well as its appearance:
Default Page	Default root page which provides the entry point for your app. It typically contains a shell with common elements like a header, footer, and a container that embeds your app's flow. The default root page is named `shell` for web apps and is automatically generated when you create an app. It embeds the default flow that is invoked when your app is first launched. This means that when the app is run, the default page in the default flow (for example, `main-start` in `main`) is rendered. If you want any other page to be rendered, change the **Default Page** setting in the flow's Settings editor. If you want to render the default page in a completely different flow, change the **Default Flow** in the root page's Settings editor. You can use your own custom root page, instead of the default `shell` page. To create the page directly from here and set it as the app's default page, click **Create**, then use the **Go to Page** link to design the page in the Page Designer. See Customize Your App's Root Page.
Favicon (Light Theme)	Default favicon used with the default light theme to uniquely identify your application in browser tabs, bookmarks, shortcuts, and more. To change the default favicon used in the Light theme, add your icon to the image gallery, then switch the icon used in this setting.
Favicon (Dark Theme)	Default favicon used with the dark theme to uniquely identify your application in browser tabs, bookmarks, shortcuts, and more. To change the default favicon used in the Dark theme, add your icon to the image gallery, then switch the icon used in this setting.
Description	Optional description of the app.
Theme	Settings to control the app's color display, which is by default Light, as well as to override the default look and feel. See Customize a Web App's Appearance.
Imports tab	Manage resources such as custom CSS files, modules, and components imported at the app level, allowing you to create declarative references in your app's pages to those resources. See Manage Custom Component, CSS, and Module Imports.

Tab/Setting	Description
PWA tab	Enable Progressive Web Support (PWA) support, allowing your app to be installed on user devices for a more native experience. See Enable Progressive Web App Support.
Security tab	Manage authentication to prevent unauthorized access to the app. By default, users must sign in using their Oracle Cloud credentials to access the app. You can further restrict access to the app (as well as its flows and pages) based on user roles defined at the visual application level. See Restrict User Access to an Application, Flow, or Page. Alternatively, you can allow anonymous access if you don't require users to sign in. See Allow Anonymous Access. While user roles can be added in the Settings editor at the app, flow, or page level, permissions are inherited from the parent. So a page inherits permissions from the parent flow, and a flow inherits permissions from the app. You can also use settings here to embed your web app in another app as well as delegate authentication for service connections.
Translations	Create translation bundles for the app, in addition to the default one, to use with a third-party translation tool. See Create Translation Bundles.

If you're working with imported mobile apps, your settings might be slightly different. See Configure Mobile Application Settings.

7

Work With Pages and Flows

Your web (or mobile) application can have multiple flows, each of which can contain multiple pages. Every application has a default flow, and every flow has a default page.

A flow is a group of one or more pages that you treat as an independent unit. It provides a way for you to split your application's logic into modules; for example, an expenses app might have one flow to group the pages where users create and edit expense reports, and another to group the pages where managers approve or reject expense reports

Your application's pages are what your users see and interact with. You can build just about any type of page in Visual Builder—simply drag and drop UI components onto the page, customize their behavior, and arrange them however you want. To display your data, you'd connect these components to REST services through your own business objects or service connections.

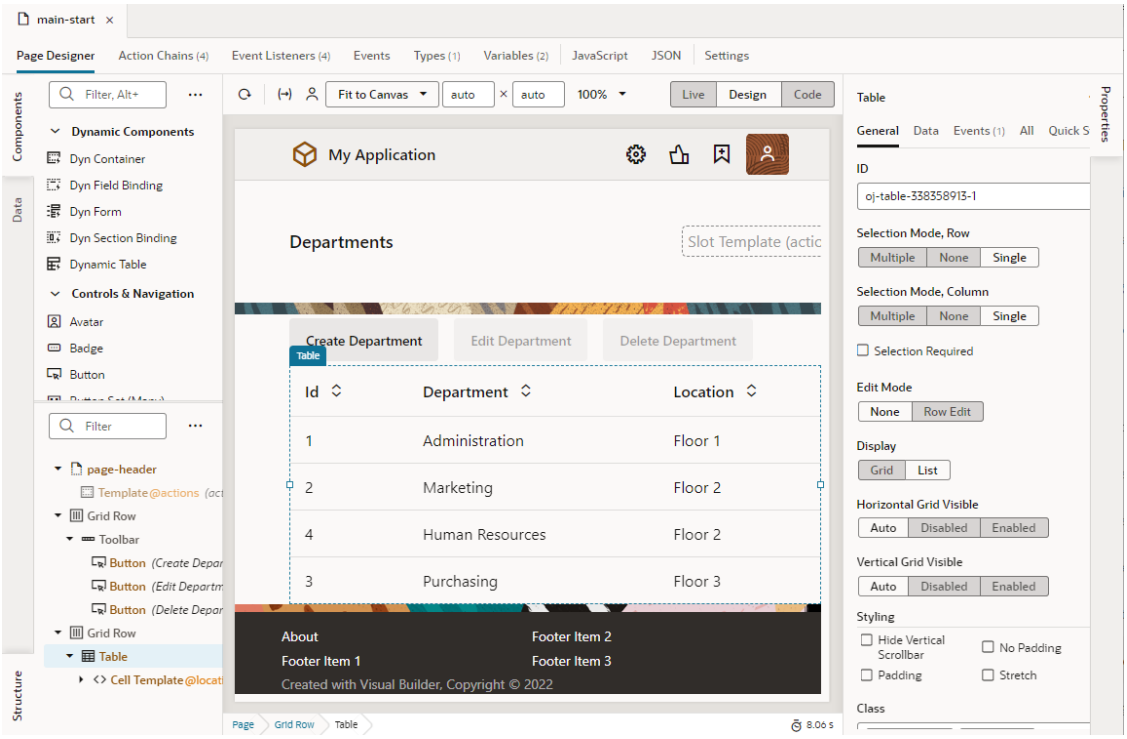
Visual Builder gives you access to a rich set of UI components to build your page, from static ones like heading and avatar to charts and gauges that visualize data, even dynamic components that display content based on rules you define. You also have access to Redwood layouts, styles, and templates based on the Oracle standard for user experience. You can use these components—all based on Oracle JET—to create rich, attractive pages.

Typically, you'll design an overview page (using collection components like a table or list) to display your data, then add other pages to let users interact with that data. Once you have your overview page, you can use quick starts to add pages that provide common functionality, for example, a page to create a new record or one that displays the details of a row selected in a table or list.

All of this is done within a declarative and visual development environment known as the Page Designer, where what you see is really what you get. For advanced use cases, you can always write custom code using standard HTML5, JavaScript, and CSS techniques.

Use the Page Designer

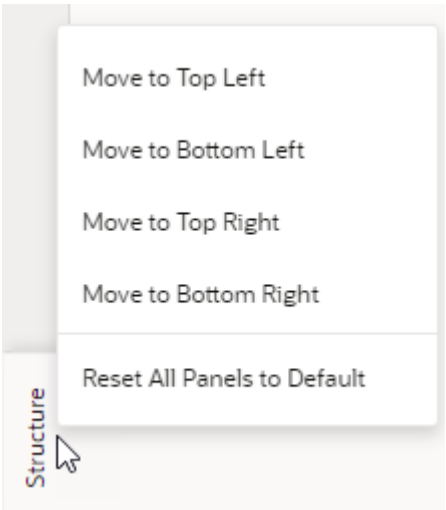
While there are several editors available to you in the Designer, the one you'll likely use most frequently is the Page Designer. To open the Page Designer, click a page in the Navigator's Web Apps (or Mobile Apps) pane:



The Page Designer offers several different ways for you to interact with your page. Specifically, you can:

- Drag and drop components on to the page using the **Components** palette;
- Work with components from a hierarchical perspective, in **Structure** view;
- Start with your data and decide how to represent the data in the user interface, using the **Data** palette;
- View and update the properties that control the appearance and behavior of whatever is currently selected in the canvas, using the **Properties** pane.

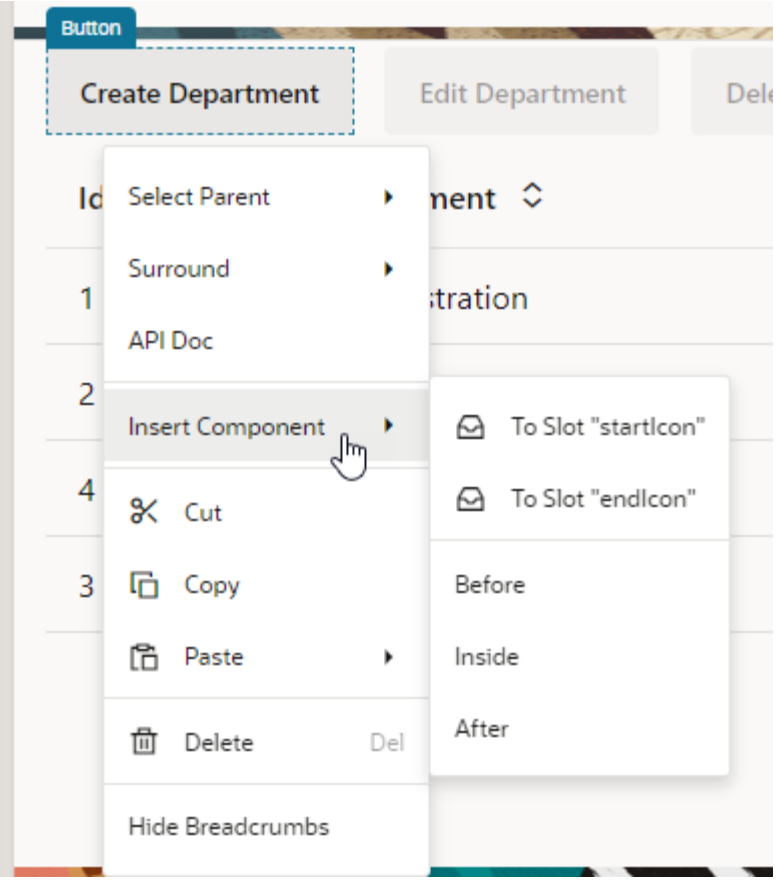
Right-click any of these panels to open a menu that lets you reposition the Components, Data, Structure, and Properties panels for your convenience. Each panel can be moved to the top left, bottom left, top right, or bottom right of the canvas (as shown here for the Structure view):



> **Tip:**
>
> You can hide or show any of these panes by clicking the **Components**, **Data**, **Structure**, or **Properties** tab.

To design your pages, you'll drag components from the Components palette to the canvas. Once you add components, the Structure view provides a hierarchical view of the components on the canvas. Use the Properties pane to view or edit a component's properties as well as open any Quick Starts that can be used with a selected component.

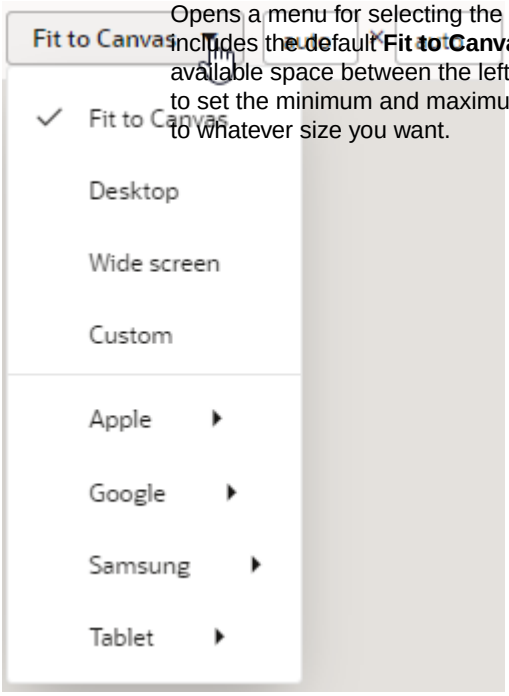
Right-clicking a component opens a pop-up menu that lets you perform several actions on the component, such as selecting its parent component, surrounding it with another component or element, or deleting it. You can also insert a component before, inside, or after the selected component.



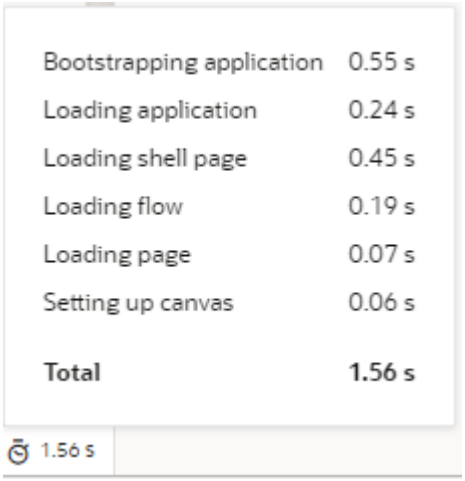
You can also bring up the right-click (or context) menu without selecting any component, both on the canvas or in Structure view. When a component isn't selected, the options in the menu will vary.

Along the top of the Page Designer is a toolbar, which lets you configure the Page Designer itself:

Toolbar Item	Description
	Reloads the page

Toolbar Item	Description
(→)	Opens a dialog box for entering input parameters for the page.
	Opens a dialog for selecting the user roles that are used when previewing pages in Live mode.
	Opens a menu for selecting the screen size represented by the canvas. This menu includes the default **Fit to Canvas** option that resizes the canvas to always take up all available space between the left and right panes. You can also use the **Custom** option to set the minimum and maximum values for viewport resolution and resize the canvas to whatever size you want.
	For an imported mobile app, shows or hides a mobile device's bezel (the border between a device's screen and its frame).
100%	Opens a dialog box for changing the magnification of the canvas.

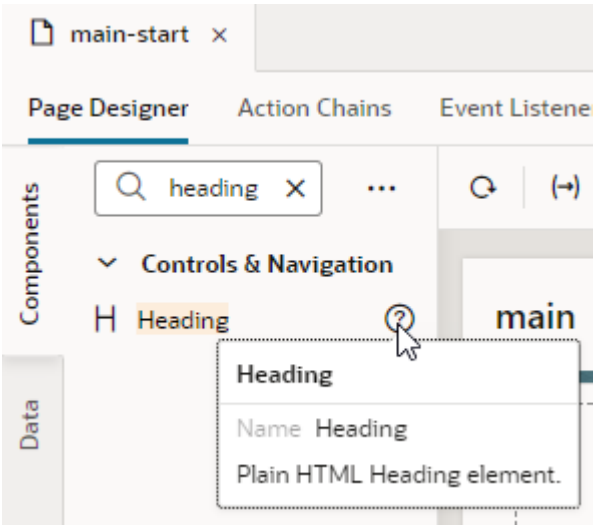
Toolbar Item	Description
	Toggles between the Live, Design, and Code modes: Live mode: Displays the page as it will look and behave when it is published. You use Live mode to interact with the pages in your application to navigate to different pages, create and modify business objects, and confirm that your application is behaving as you expect. • Design mode: Contains a canvas area that you use to place and position components in the page. This is the mode you use most frequently. • Code mode: 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 Structure view to edit and reposition elements, the changes are automatically reflected in the code. The Components palette, Structure view, and the Properties pane 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. When you switch from Design to Code mode (for example), the source code of a component selected on the canvas will be highlighted in the code editor. All the modes are synchronized, so changes you make in the Properties pane or Structure view are visible when you switch to a different mode. **Tip:** Hold the **Ctrl** key (**Cmd** on Mac) to momentarily switch between **Live** and **Design** modes. Make sure the cursor is on the canvas, then hold the **Ctrl** key. For example, you can check the values in a drop-down menu by simply holding the Ctrl key and clicking the menu on the canvas.
	At the bottom of the canvas (not in the toolbar), a breadcrumb path that displays a hierarchical list of links for a selected component to indicate its placement in the page's structure. Clicking a link in the breadcrumb path selects that component on the canvas and in Structure view and lets you view its details in the Properties pane. To hide breadcrumbs (shown by default), select **Hide Breadcrumbs** in the context menu.

Toolbar Item	Description
...	Displays time taken to render and display the page in the canvas. Clicking the icon will show a breakdown of how long different tasks (such as bootstrapping and loading a shell page) take in order to display the page, as shown in this example:  Rendering is done in runtime execution mode, with the Page Designer serving the resources needed to display the page. Because each resource is requested in a different phase of runtime initialization, the time between these resource requests is measured and summarized. This information can thus help you isolate runtime issues that may cause your app to load slowly. For example, if a page takes time because of long REST calls, you might decide to defer the calls or run them in parallel. If the page contains runtime errors, an error message will show instead. Click the message to get details about the errors and resolve them for the page to render correctly.

The Components Palette

The Components palette contains UI components and organizational elements that leverage the Oracle JavaScript Extension Toolkit (JET) to help you build pages. To add a component to a page, simply drag it from the palette onto the canvas.

Components by default show in list view and are organized by categories. You can scroll to locate a component, although it's simpler to use the Filter field. Hover your cursor over a component's Info icon to get hints about the component.



Note:

JET Core Pack components, written entirely using the VComponent API and the JET Virtual DOM architecture, are available under the **Early Access** category. While you can choose when you want to move to the new components in your development cycle, take note that no updates are planned for legacy JET components; all updates and new functionality will be available only through the Core Pack. For more information, see Core Pack overview in JET documentation.

For ease of use, *all* JET components show in the Components palette, including those without design-time properties. These components show under the **Advanced** category and typically require you to manually code different aspects of their functionality. Use tooltips to access JET documentation on how to use these advanced components.

Click **Components** to show or hide the Components palette.

If you want to customize the palette's default settings, click **Components Menu** (***):

- To view components laid out as a grid, select **Grid** (default is **List**).
- To always show components in every category, select **Expand All**; to hide them, select **Collapse All** (default is **Expanded By Default**).
  If you want to change the default view with components collapsed, deselect **Expanded By Default**. Changing this setting when working in a page editor won't change your current view, but it will take effect when you open a new editor. To change this setting in your current view, use **Expand All** and **Collapse All**.
- To hide categories and view all components in a flat list, deselect **Show Categories**.

You can also:

- Click **Get Components** to access your instance's Component Exchange, from which you can add JET web components to your page.
- Click **Show Deprecated** to view components that have been deprecated (badged deprecated for easier identification). Deprecated components are flagged in your application's audit and are retained only to allow existing applications to continue to run. It's recommended

that you move away from these components as soon as possible. Use the component's tooltip to view details of the version it was deprecated in, as well as a suitable alternative.

- Click **Show Maintenance** to view components in maintenance mode (badged maintenance for easier identification). As with deprecated components, you should transition away from components in maintenance mode as they will eventually be deprecated. Use the component's tooltip to view a suitable alternative.

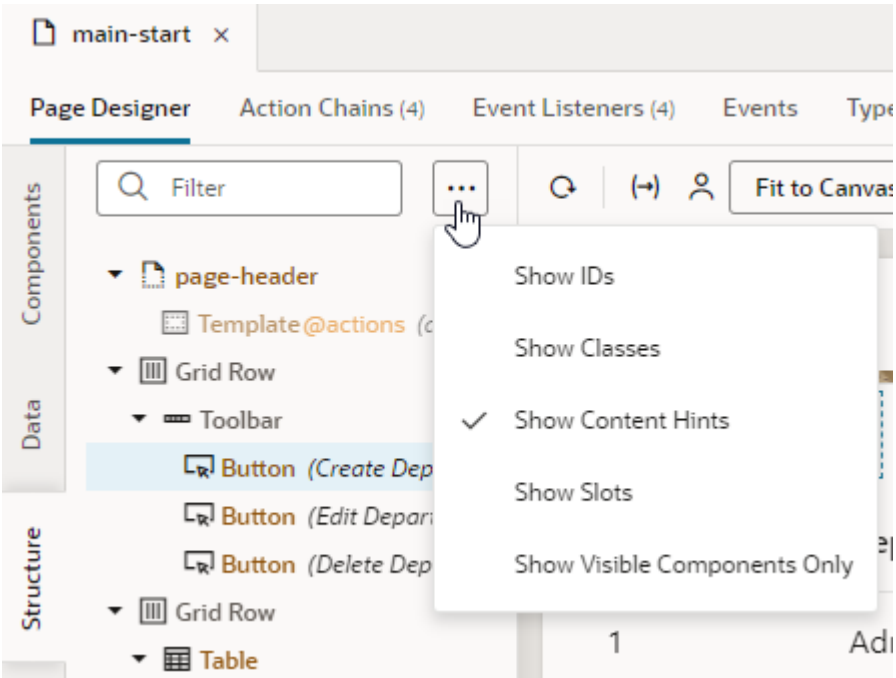
The Structure View

The Structure view provides a structural view of components on the canvas. You can use it to see the hierarchy of a page's components and to reposition components within the page's structure. You can also add components in the Structure view. Click **Structure** in the Page Designer to show or hide the Structure view.

Selecting a component in the Structure view also selects it on the canvas and displays its properties in the Properties pane. You can organize and reposition components on the page either by dragging them into position in the Structure view or by dragging them from the Structure view directly to the canvas. You can also select multiple components in the Structure view to simultaneously reposition them (for example, to move them into a new container).

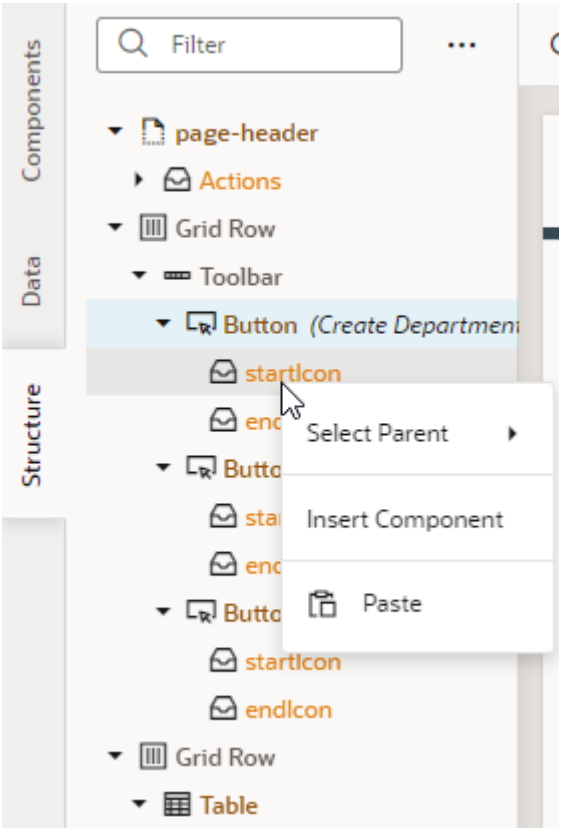
It's also possible to add components to a page by dragging components from the Components palette or the canvas into the Structure view; you can also right-click and select **Insert Component** in the context menu. This option can help you position a component exactly where you want to add it, especially if you're working with complex layouts. When you select **Insert Component** on a selected component, you'll be able to add a component before, inside, or after the selected one. If the selected component has multiple slots, you will have the option to drop the component into a particular slot; if it's a single slot, the component is dropped directly into the slot.

Use the Page Structure Menu (***) to set your Structure view preferences.

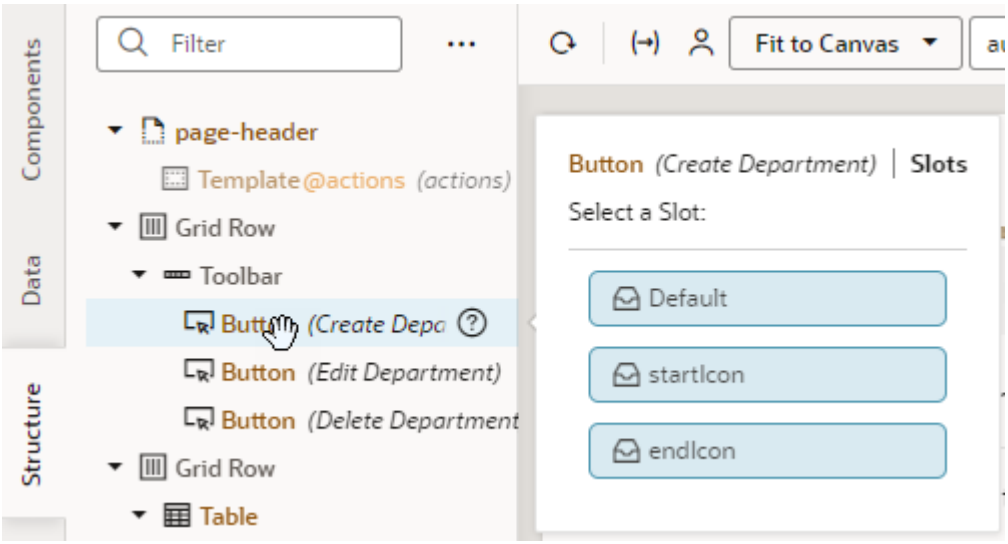


You can choose to display a component's details, for example, its id, classes, or hints about its content (enabled by default). You can use **Show Visible Components Only** to view only the nodes of the components visible on the page. When combined with the default view that fades into the background components that are not currently displayed on the page, this option can trim background information and allow you to focus on parts of the page, a section at a time.

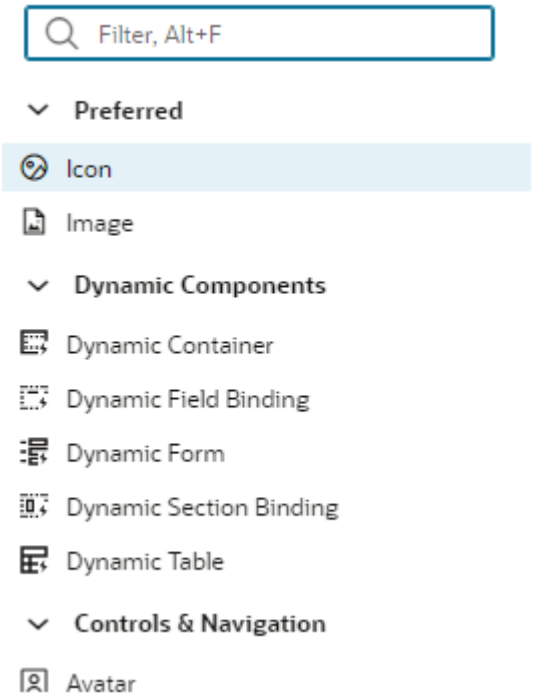
You can also enable **Show Slots** to display the location of empty as well as occupied slots. When slots are visible, you can easily locate the slot where you want to drop a component, as shown here:



Even if you don't enable the option to show slots, it's possible to locate available slots by pausing over a component node when dragging a component into Structure view. If the component node has slots, a pop-up menu that lists the available slots opens; you can then drop your component into the desired slot in the pop-up menu.



You can also right-click a slot and select **Insert Component** to drop a component directly into a slot. Doing this brings up the list of components, including a set of recommended ones. Recommended components show under a **Preferred** category and are based on the type of component that can be used in the slot. For example, components recommended for a button's `startImage` and `endImage` slots are icons and images:



The Data Palette

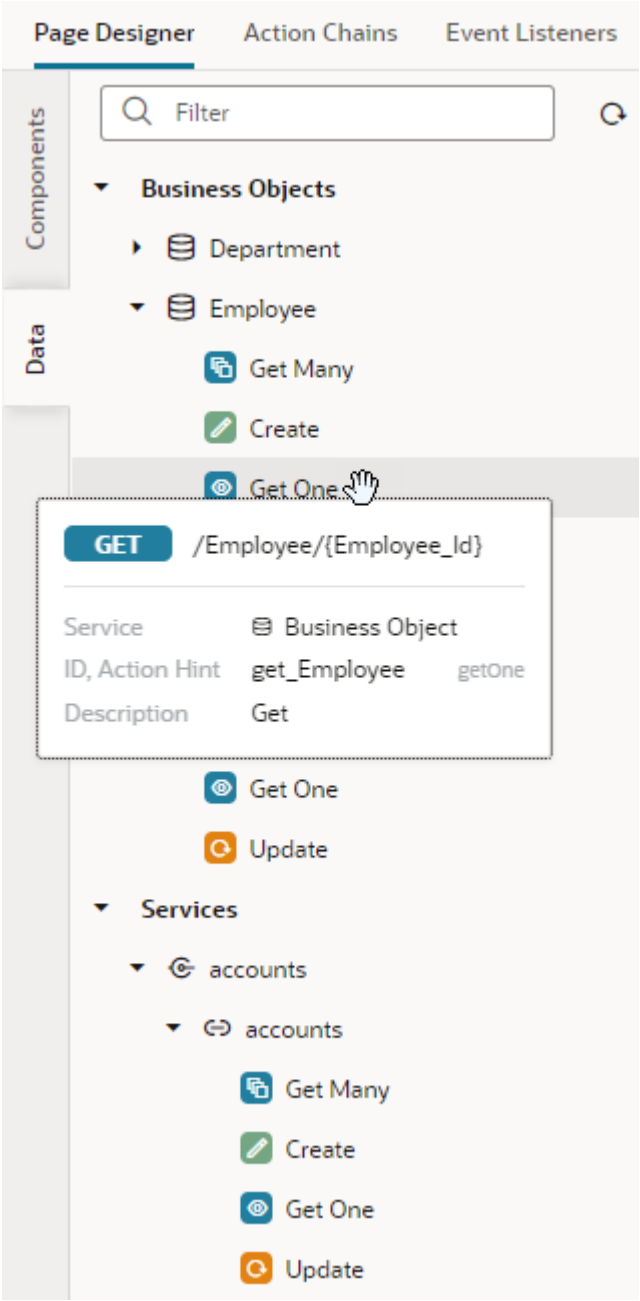
The Data palette provides a data-centric approach to application design. Instead of you choosing UI components and binding each component to a data source, this design approach

starts with your data sources and suggests UI options that optimally display your application's data.

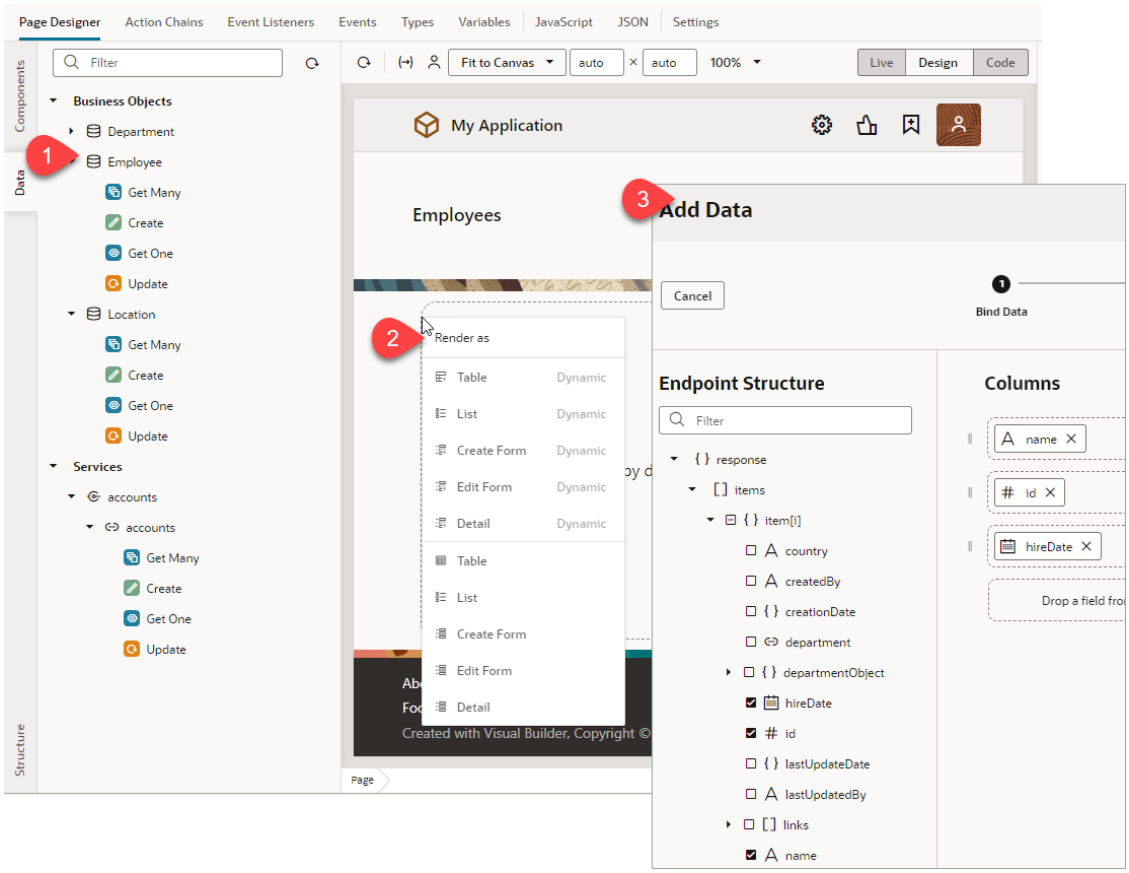
Data is the basis of any application, and when creating an application in Visual Builder you can work with multiple sources of data, all of which is based on REST. You can create business objects and use the REST endpoints automatically generated for the object to fetch your data, define service connections that link your application with external REST APIs to retrieve data, or consume services from multiple sources in combination with your own business objects.

The Data palette brings all these data elements together, allowing you to drag and drop them onto the canvas and render their data in components tailored to display that data. You also have access to quick starts that simplify the task of binding that component to fields in your data sources.

Here's a Data palette example, showing an Employee object's endpoints as well as the `accounts` endpoints (available through a connection to the Sales and Service (`crmRestApi`) service in the Oracle Cloud Applications catalog):

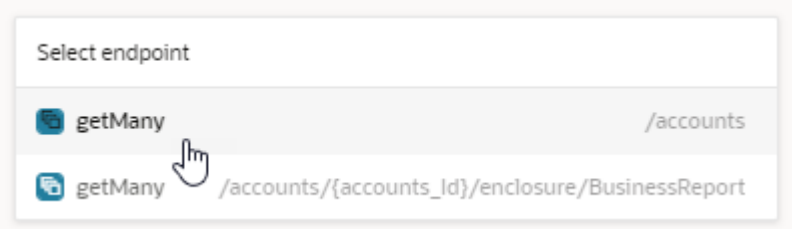


Now let's say you want to show employee data on a page: you would simply drag the Employee business object from the Data palette and drop it on the canvas. (You can drop a data element on the page canvas, in the Structure view, or in Code view.) When the **Render as** pop-up appears, you can choose from the list of components that Visual Builder suggests based on the available REST endpoints:



Notice how both dynamic and standard options are shown. Once you select an option, the corresponding quick start opens, to walk you through the required configuration tasks.

You use the Data palette in much the same way to fetch data from a service connection's endpoints, except when a service contains multiple endpoints of the same type—for example, two `Get Many` endpoints—you'll be additionally prompted to select the correct endpoint:



You don't always have to work with whole objects or services; if you know what kind of data interaction you want, you can drag and drop a particular endpoint. For example, to let your users create new employees, you only need the `create employee` endpoint, not the whole Employee object.

Because endpoints enable CRUD operations, your component suggestions reflect endpoint functionality. For example, data from a Get Many endpoint lends itself to a table or list view. Similarly, a Create endpoint will render a form. Here's a summary of the components suggested (standard and dynamic) for a particular type of endpoint:

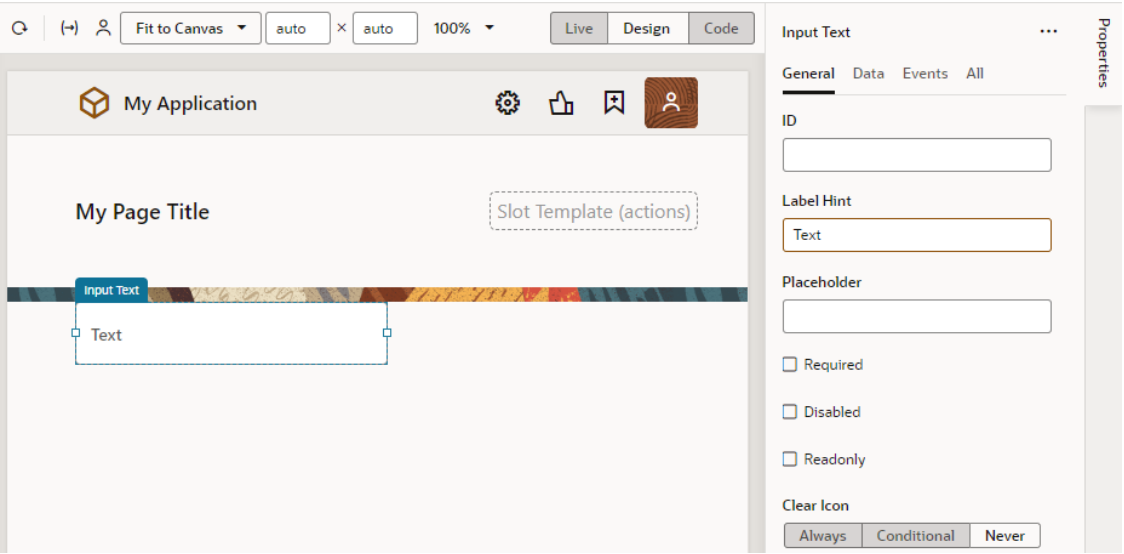
Endpoint Type	Component	Associated Quick Start
Get Many	Table, Table Dynamic, List, List Dynamic	Standard: Add Data Dynamic: Configure Layout
Create	Create Form, Create Form Dynamic	Standard: Configure as Create Form Dynamic: Configure as Create Form
Get One	Detail Form, Detail Form Dynamic	Standard: Configure as Detail Form Dynamic: Configure as Detail Form
Update	Edit Form, Edit Form Dynamic	Standard: Configure as Edit Form Dynamic: Configure as Edit Form

When working with quick starts in the Data palette, keep in mind that both standard and dynamic component quick starts *add a form or a table to the existing page and configure it*; they don't add pages to your application like they do for standard components or configure an existing form or table, as in the case of dynamic components. Except for this key difference, the quick starts are similar to those used for standard and dynamic components.

The Properties Pane

The Page Designer's Properties pane displays the properties of a component that is currently selected on the canvas or in Code view and can be used to set metadata attributes such as ID, display name, and description. It also provides various properties to customize a component's layout, style, and behavior.

Component properties are organized in tabs in the Properties pane and can vary depending on the component. For example, here's the Properties pane of an input text component, showing the **General**, **Data**, **Events**, and **All** tabs; some components such as a table or list view collection component have an additional **Quick Starts** tab:



Here's an overview of what you can do in the different tabs:

Component Properties	Description
General	Contains the most important properties of the selected component, such as layout and style. The slot value of a component inside a parent component's slot also shows here. Select the sub-component added to a slot (for example, a button's icon) and change its slot value to move it from the `startIcon` slot to the `endIcon` slot. This way, you can modify the slots of dropped components without accessing the HTML code. The slot used by the sub-component is also visible in the sub-component's All tab and can be modified there (for example, to bind it to a variable).
Data	Contains properties that are expected to be bound to data. The General tab and All tab also contain properties that can be bound to variables and expressions. See Bind a Component to Data.
Events	Used to bind a component's events to trigger action chains that define its behavior (for example, to open a URL when a button is clicked). See Start an Action Chain From a Component.
All	Contains more advanced component properties and shows all properties, including custom properties. Custom properties are those not defined in component metadata, for example, `data-*` attributes, and can be added by clicking **+** next to General Attributes.
Quick Starts	Contains a list of Quick Start wizards available for the component. When you add a collection component such as a table or list, this tab contains a list of wizards to help you add some actions that are typically associated with the component, such as mapping the collection to data and adding Create and Detail pages.

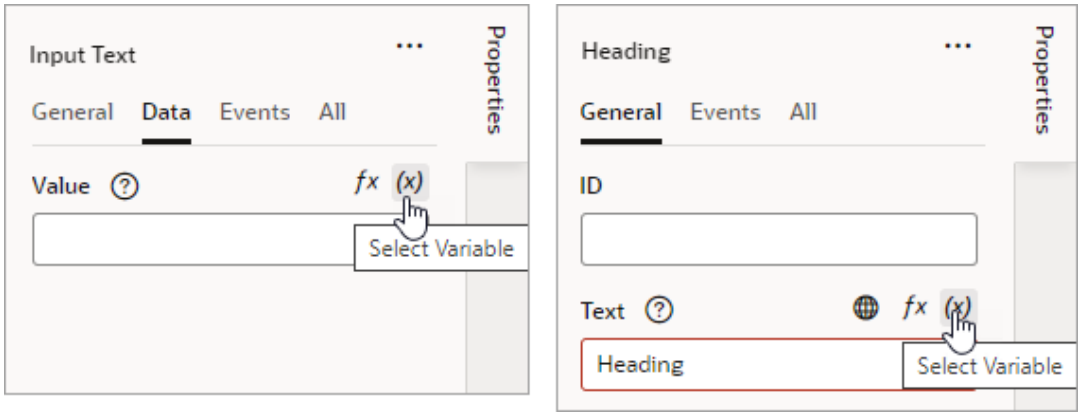
You can toggle **Properties** to hide or show the Properties pane.

Variables Picker

When working with a component's properties, you can bind a variable to the component, populating it with the data stored in the variable. For example, you might bind an Input Text component to a particular field from a REST endpoint, or bind a Heading component to a variable. To do this, you use the Variables picker, which you access by clicking the icon that appears when you hover over a property.

Here's an example of how you can bind a component to a variable:

1. Locate the component's property to bind the variable to, then click to open the Variables picker. For example, you can find the Variables picker on an Input Text's **Value** property on the **Data** tab (shown on the left) or on a Heading's **Text** property (shown on the right):



2. Select the variable available in the current scope, or click **Create** to create one.

   You should see an expression added to the property, something like `[[ $flow.variables.userName ]]`. You can extend this expression if needed within the property itself. For example, you can add `Information` and enclose the `userName` expression within single quotation marks to something like this:

   ```
   [[ " '"+$flow.variables.userName + "' Information"]]
   ```

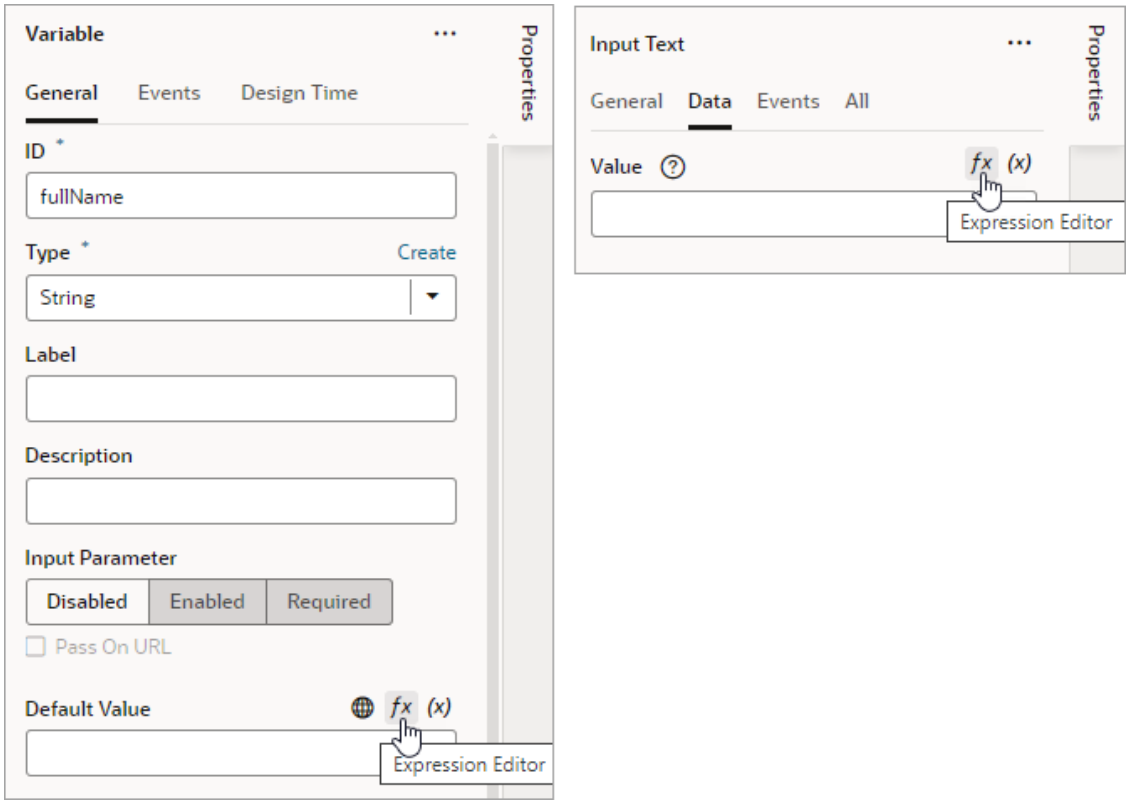
   For complex expressions, you might want to use the Expression Editor.

Expression Editor

Some properties accept *expressions* that resolve to a single value at runtime. An expression is a string enclosed in `[[...]]`, indicating that it should be evaluated. It can include variables, operators, functions, system properties, and the like.

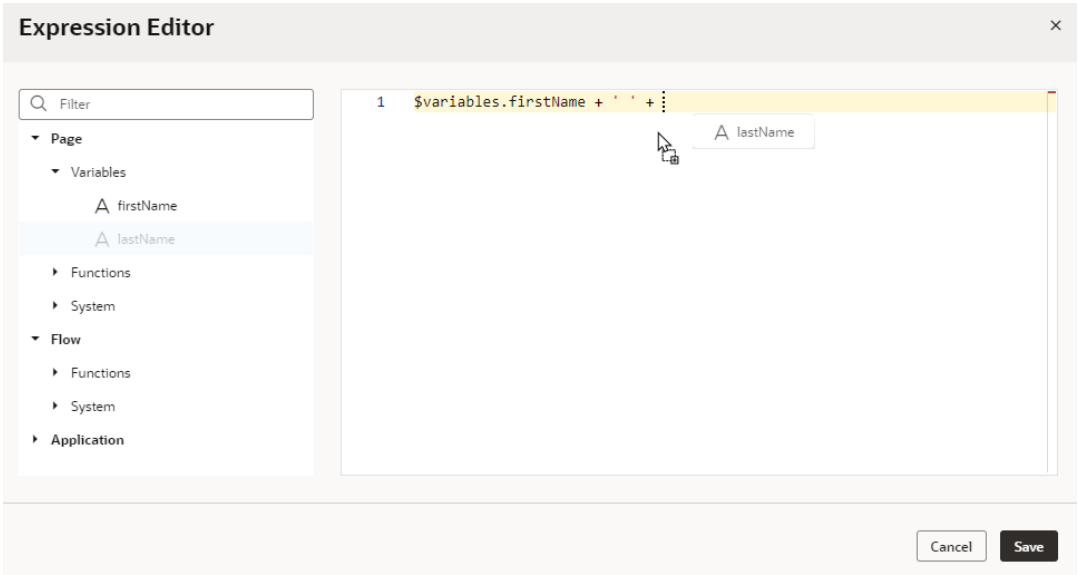
To easily define expressions, use the Expression editor, which you access by clicking the *fx* icon that appears when you hover over a property that supports expressions.

While expressions can be defined for a variety of tasks, they are commonly used to set the default value on a variable (shown here on the left) or to bind a UI control to its data (shown on the right):



For example, here's how you can define an expression to display a user's full name in an Input Text component, based on the `firstName` and `lastName` variables:

1. Switch to the component's **Data** tab and click fx .
2. In the Expression editor, drag and drop the `firstName` and `lastName` variables onto the canvas to define your expression. Predefined variables, functions, and other properties are available to you within their defined scope on the left pane.



Make sure you remove `null` on the canvas. Your final expression may look something like this:

```
$variables.firstName + ' ' + $variables.lastName
```

Note that `$variables` is a shortcut for variables in the current scope. So in a page, `$variables` is the same as `$page.variables`.

3. Check your expression syntax to make sure it's valid and click **Save**.

The Expression editor supports standard JavaScript expression syntax, with some customization to make it work it with UI components in Visual Builder. Suppose you have a variable `members` with a field `count` and you need to display a message when there are more than 10 members. In JavaScript, you might code it like this:

```
members.count > 10 ? 'too many members (current value=' + members.count + '
while max value is 10)' : 'within reason'
```

When applied to a Text component in Visual Builder, the same expression would look like this:

```
<oj-bind-text value="[[ $variables.members.count > 10 ? 'too many members
(current value=' + $variables.members.count + ' while max value is 10)' :
'within reason' ]]"></oj-bind-text>
```

Here are a few simple examples:

Use Case	JavaScript Expression	Expression in Visual Builder
To concatenate the `firstName` and `lastName` variables:	`firstName + ' ' + lastName`	`$page.variables.firstName + ' ' + $page.variables.lastName`
To check whether `employee` and `job` are not null or undefined, before trying to access `jobTitle`:	`employee?.job?.jobTitle`	`$variables.employee?.job?.jobTitle`
To calculate the total price of a row based on its `price` and `quantity`:	`row.price * row.quantity`	`$current.row.price * $current.row.quantity`

For help with standard expression syntax, see Expressions and Operators. For details on how to write efficient expressions when a referenced field might not be available or the field's value could be null, see How Do I Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null?

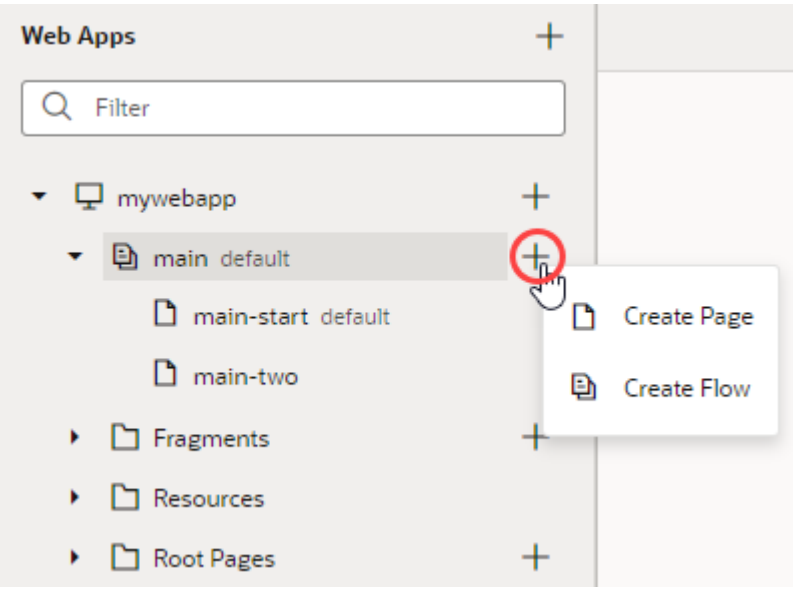
Create and Manage Pages

When you first create an application, one or more starter pages called `flow-start` are created for you within the flow. You can create more pages under the flow, or create a new flow and add pages to that flow. Every flow you create, by default, has its own starter page.

> **Tip:**
>
> New pages are by default empty. Instead of empty pages that you must design and develop, you can create pages with some initial content. These pages can be based on prebuilt Redwood template patterns or fragments in your application.

To create an empty page in a flow:

1. Open a web (or mobile) application in the Navigator and expand the app node.
2. Expand the flow where you want to add a page.



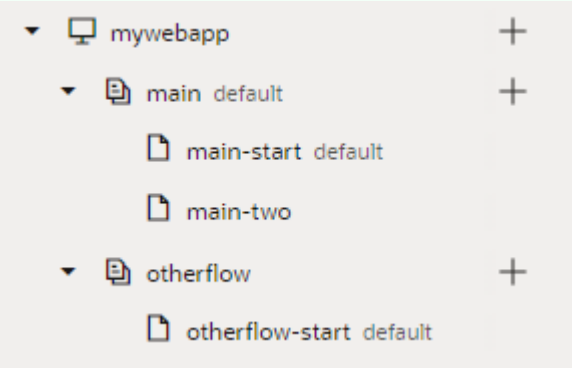
3. Click the **Create Page** icon (+) next to the flow where you want to create the page, then select **Create Page**.
4. Enter the name of the new page in the **Page ID** field of the Create Page dialog box. Click **Create**.

After a page is created, you can duplicate, rename, even delete it if needed. Right-click the newly created page and select an action.

By default, the starter page in a particular flow is set as the flow's default page and that flow is set as the app's default flow. For example, when the `main-start` page is the starter page in the `main` flow, `main-start` is set as the default page in the `main` flow, and the `main` flow is set as the default flow for your app. The default flow is embedded within the app's root page, which is invoked when your app is first launched. This means that when the app is run, the default page in the default flow is rendered, which is `main-start` in this case. If you want any other page to be rendered, change the **Default Page** setting in the flow's Settings editor.

Tip:

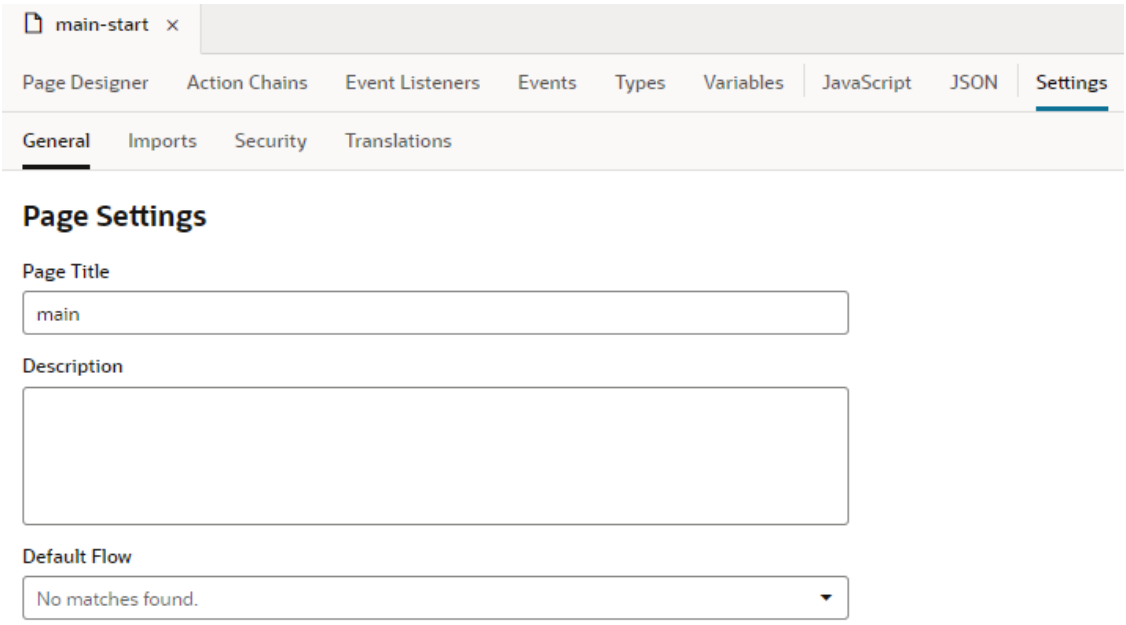
The default page and flow in your app are badged as `default` to help you quickly identify the default page in the default flow. Default pages in all flows are also badged for easier identification.



Manage Page Settings

Each page in your application includes a Settings editor, which you use to primarily manage imported resources such as custom components, CSS files, and modules. You can also manage security and create page-scoped translation bundles.

To configure settings for an application's page, open the page, then click **Settings** to open the Settings editor:



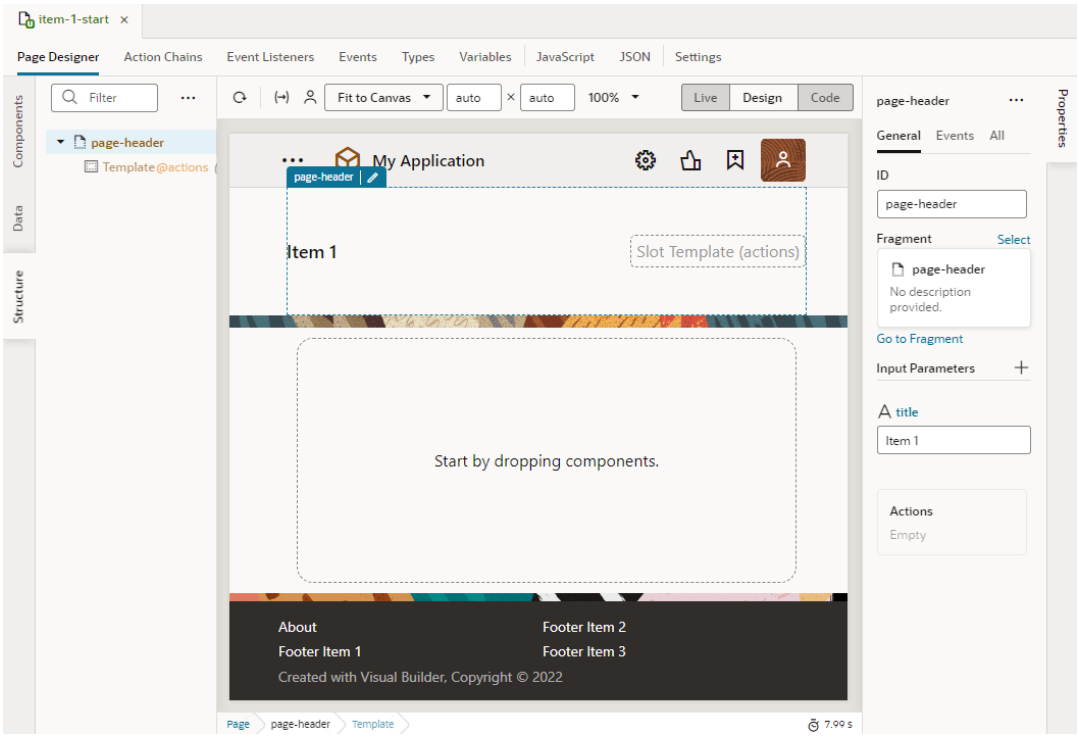
Here's how you can use the different page-level settings:

Setting	Description
General tab	Manage general page settings:
Page Title	Title of the page to be used in the browser and in browser history. If you use a quick start to create pages, this is populated for you.
Description	Description of the page. If you use a quick start to create pages, this is populated for you.
Default Flow	Default flow when you create sub-flows to embed in the page the content of another page or flow. Sub-flows (or nested flows) allow you to change the content displayed in a page without leaving the page. See Embed a Flow Within a Page. For root pages, this setting defines the flow that opens when your app is first run. See Customize Your App's Root Page.
Imports tab	Manage resources such as custom CSS files, modules, and components imported at the page level, allowing you to create declarative references in the page to those resources. See Manage Custom Component, CSS, and Module Imports.
Security tab	Add user roles (defined at the visual application level) to control access to the page. Only users granted one of the assigned roles can navigate to the page. Note that permissions are inherited from the parent, so the page inherits permissions from the parent flow. See Restrict User Access to an Application, Flow, or Page.
Translations tab	Create translation bundles for the page, in addition to the app-level translation bundles, for use with a third-party translation tool. If you create a translation bundle for a page, 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. See Create Translation Bundles.

Customize Page Headers

All pages in your app include a page header, separated from page content by a visual stripe, that provides users with page-specific information. This default header is defined by the `page-header` fragment and can be customized to suit your requirements.

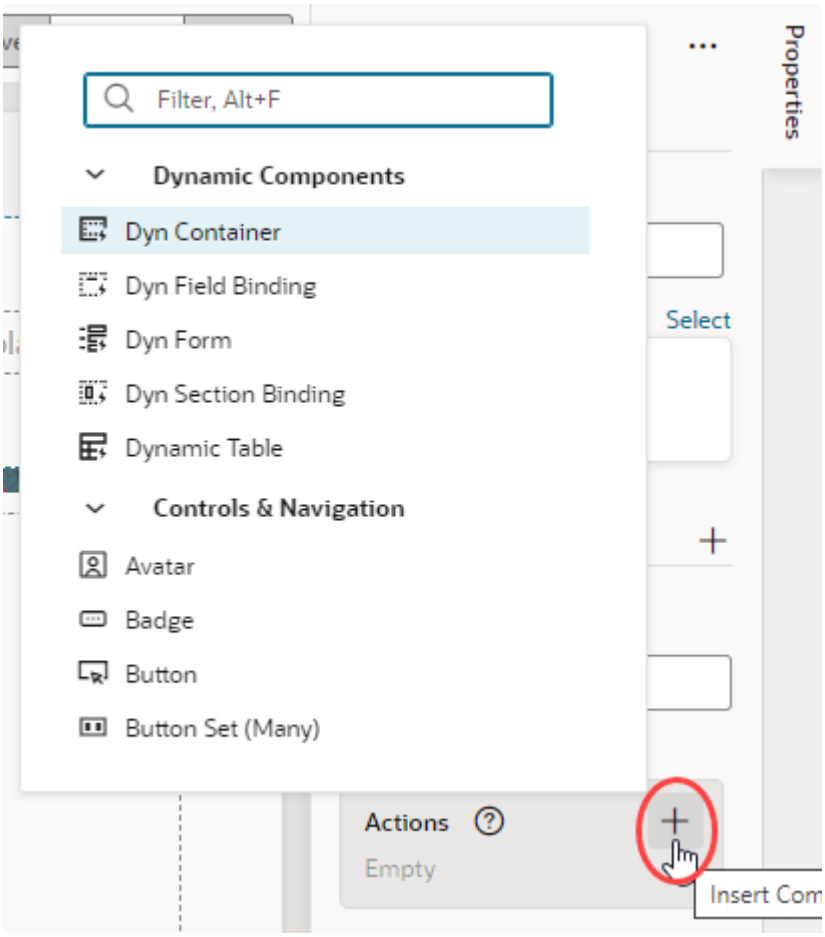
1. Open your page, then select the area above the horizontal stripe on the page canvas. Alternatively, select `page-header` in Structure view.



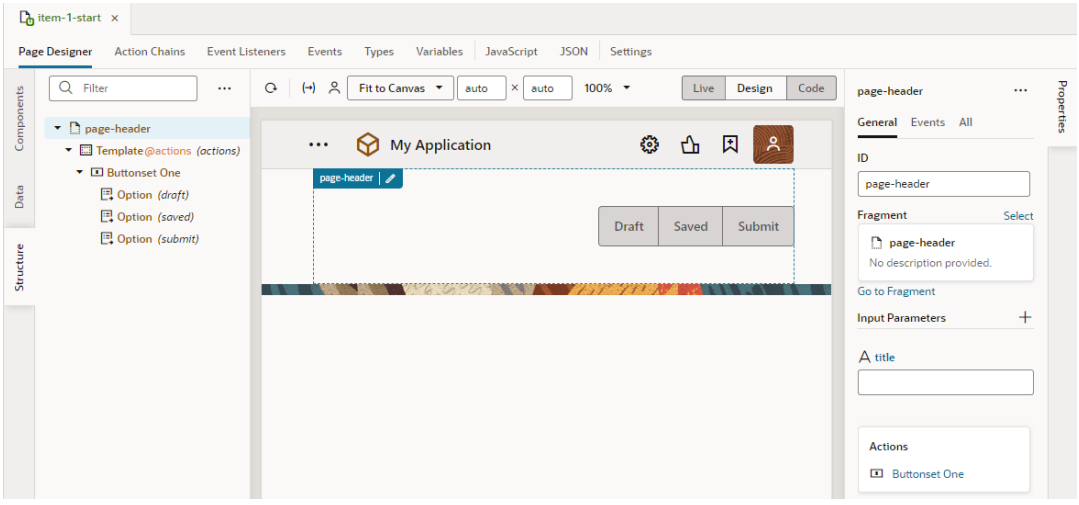
2. In the `page-header` fragment's Properties pane, use the **title** property under Input Parameters to set a page title. You can also remove the title text. By default, the page's flow name is used as the title.
3. Optional: To add page-specific actions, add components of your choice to the Actions slot. For example, you might add a Button Set to the fragment slot and configure it to display some options.

   To do this, simply drag the Button Set (One) component from the Components palette onto the **page-header** fragment on the canvas, onto **Template (actions)** in Structure view, or onto **Actions** in the fragment's Properties pane.

   Alternatively, hover over **Actions** in the fragment's Properties pane, click the **Insert Component** icon (+), and select a component.



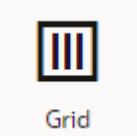
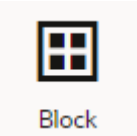
Here's an example of a page with some page-specific actions in the page header:



Set a Page's Layout

All pages have a preferred layout, and you can add additional layout containers and components within this layout to design your pages.

When you select a page (in other words, when no component on the canvas is selected), you use the Preferred Layout options in the Properties pane to set a layout for your page: *Grid* (default), *Flex*, or *Block*. Here's an overview of each page layout:

Layout Type	Image	Description
Flex		The Flex layout lets you add components in rows of any size. In a flex layout, you can lay out the children of a flex container in any direction, and the children will grow to fill unused space or shrink to avoid overflowing the parent. You can also nest boxes (for example, horizontal inside vertical or vertical inside horizontal) to build layouts in two dimensions. The Flex layout provides the most flexibility and you can adjust several properties for alignment, justification, and so on, in the Properties pane.
Grid (default)		The Grid layout builds on the Flex layout, but adds a 12-column grid and rows that make it easier to align elements when you position them. The pages in your application incorporate responsive design to resize gracefully based on the size of the display area of the device.
Block		The Block layout displays components that you drop on a page as blocks; each component starts on a new line and takes up as much horizontal space as it can. This layout is useful when your app already includes hand-coded pages, or when you want to drop a few components on a new page and manually adjust the layout.

Every component you add to the page is placed in a row in the page's layout—or in a *layout component* that you've placed on the page's layout. Layout components are predefined Oracle JET components and patterns that let you control the initial data display and allow the user to access additional content by expanding sections, selecting tabs, or displaying dialogs and pop-ups. Available under several Layout categories in the Components palette, they are various containers and components that you can drag and drop on to the canvas or in Structure view. Some are specifically designed to help you with design styles; for example, the accordion to display a set of collapsible child elements, a navigation list to navigate between different content sections, or a masonry layout that lays out its children in a grid of tiles. Here's a list of some commonly used layout containers and components:

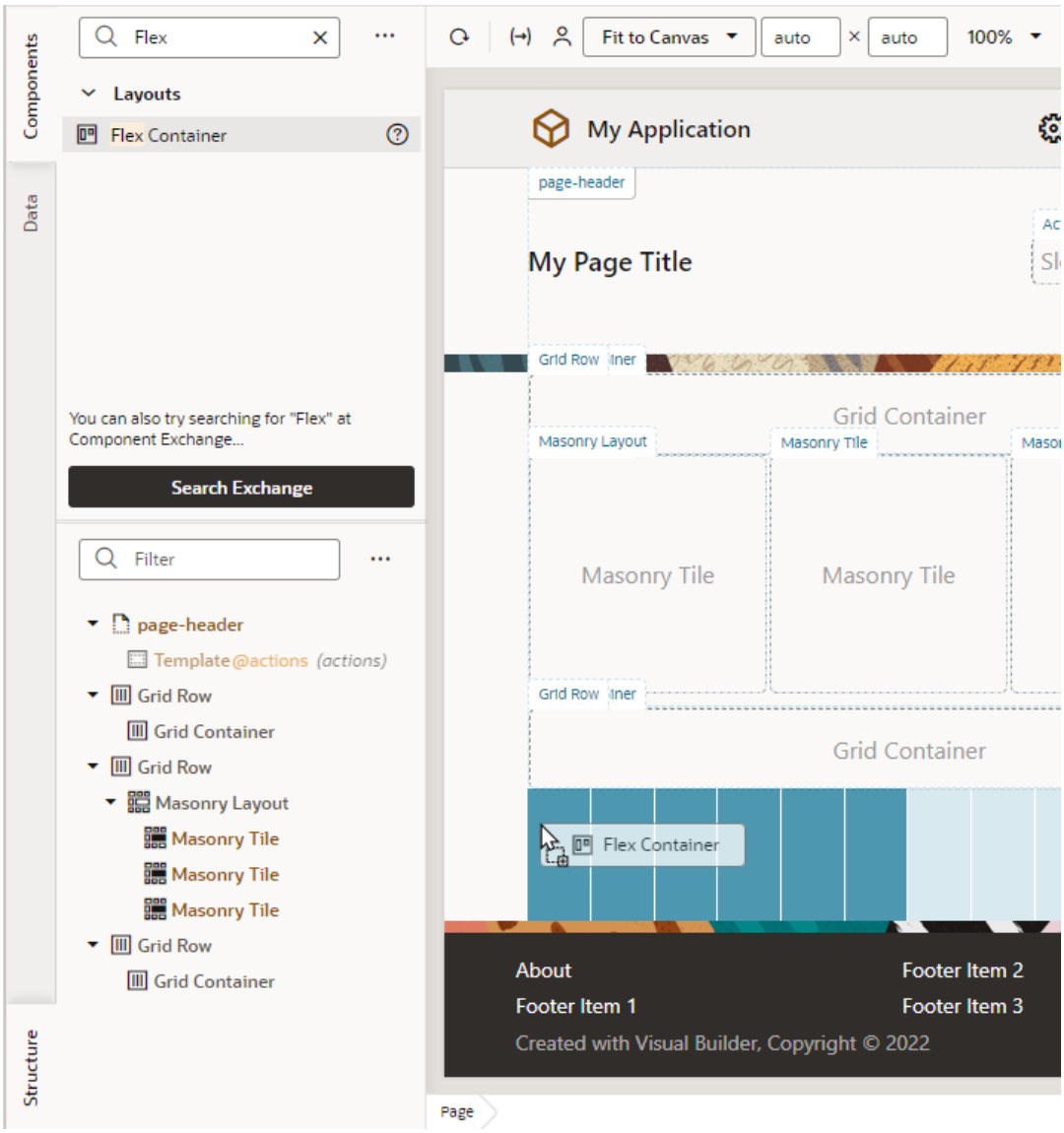
Container Components	Description
Flex Container	The flex container is a flexible container which is useful for responsive designs that optimize the use of the available space.
Grid Container	The grid container is a 12-column grid that is useful when you want to align components precisely according to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

See the **Layout & Nav** section in the Oracle JET Developer Cookbook for examples of how you can use various layout components.

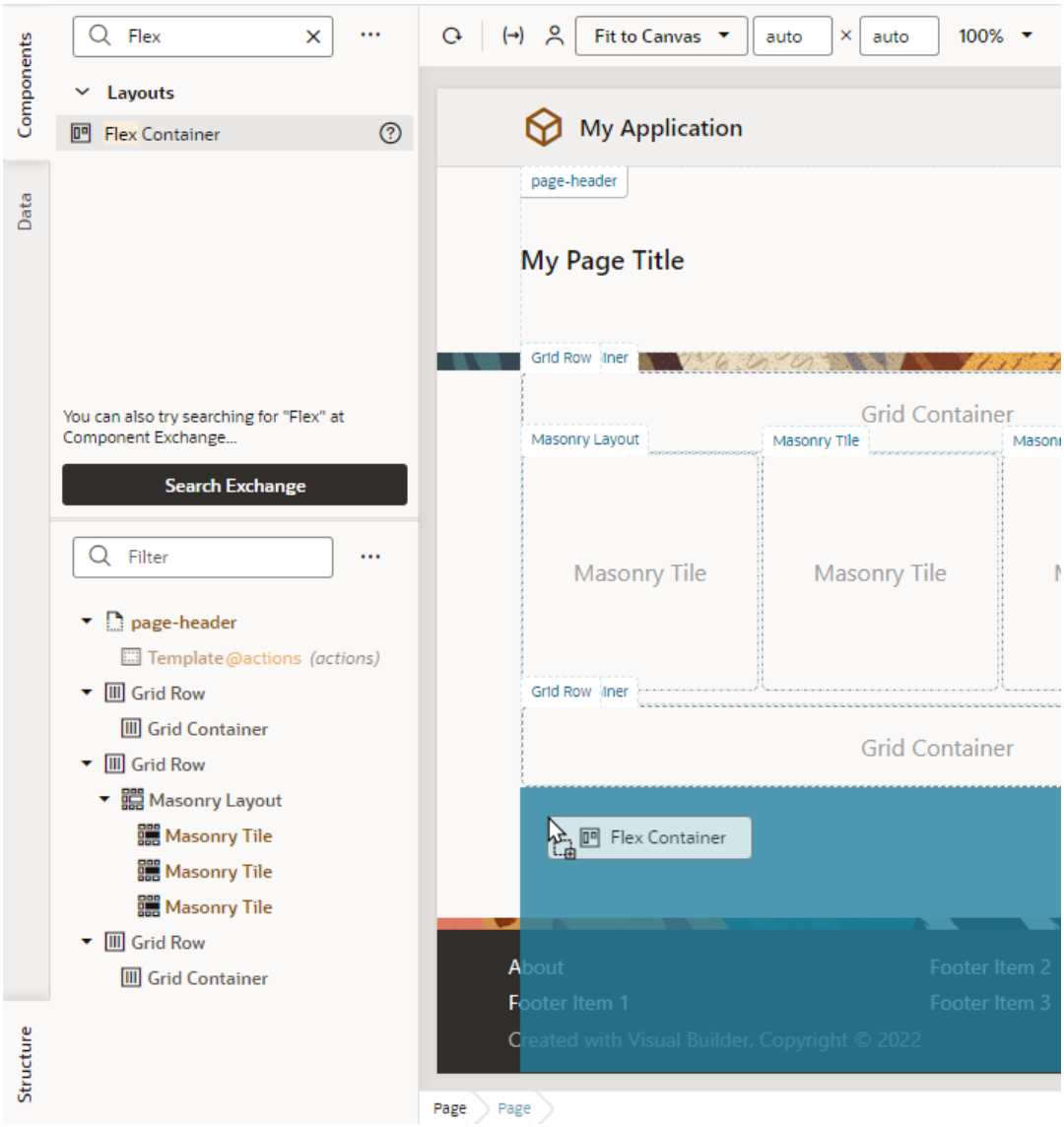
To add a layout container or component to a page:

1. Drag the layout container or 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. For example, here's what your canvas might look when you're dragging a Flex container on a Grid layout:

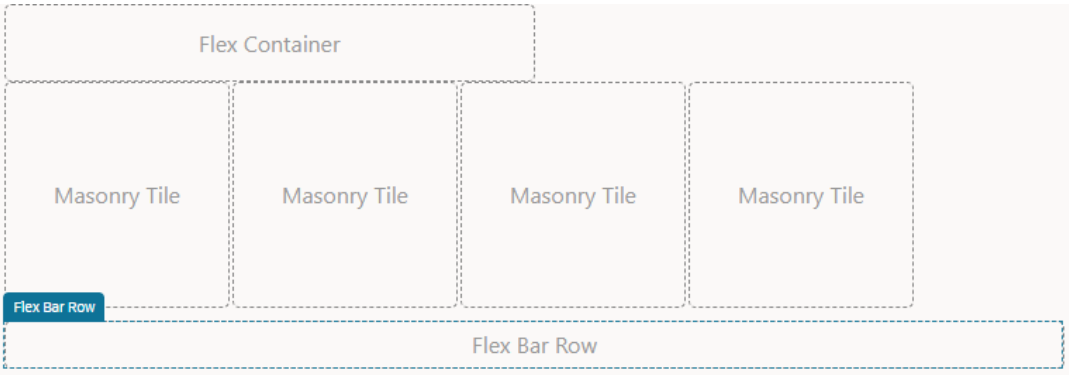


   If you were dragging the same components on a page that uses the Block layout, your view might be something like this:

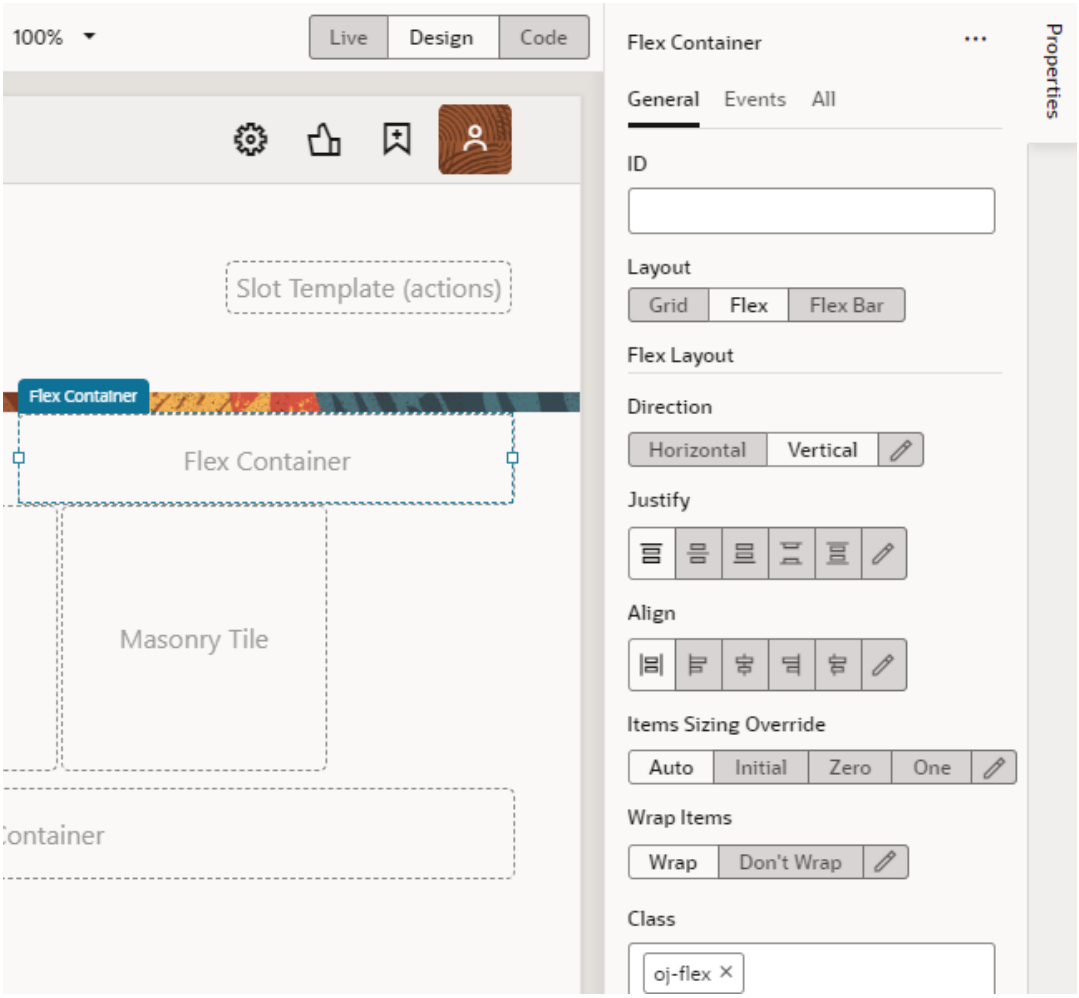


Notice how the Grid Container and Masonry Layout components in the Structure view aren't automatically enclosed in rows. That's because unlike the Grid or Flex layout, components dropped on a page that uses the Block layout aren't automatically wrapped in Grid or Flex rows.

It's also possible to 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 container. 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.



2. Select the component and modify its properties in the Properties pane. You can modify the display settings of each row in a layout to control the layout of the components within the row.



You can drag additional components into the container, or place them above or below an existing row to create new rows.

Create Pages From Templates

Instead of starting with empty pages for your application, it's possible to create pages prepopulated with the contents of a *pattern* or *fragment*.

- Redwood patterns are designed for high-fidelity interactions and responsive performance and can be useful in creating pages that provide a consistent user experience across your app. It's also easy to customize these pages to suit your business requirements.
- A fragment is a reuseable piece of UI that you can include in your application's pages. You can add fragments as sections to a page as well as multiple pages, even use them as page templates to create entirely new pages. Fragments created for one page can also be reused in other pages.

Create Pages From Patterns

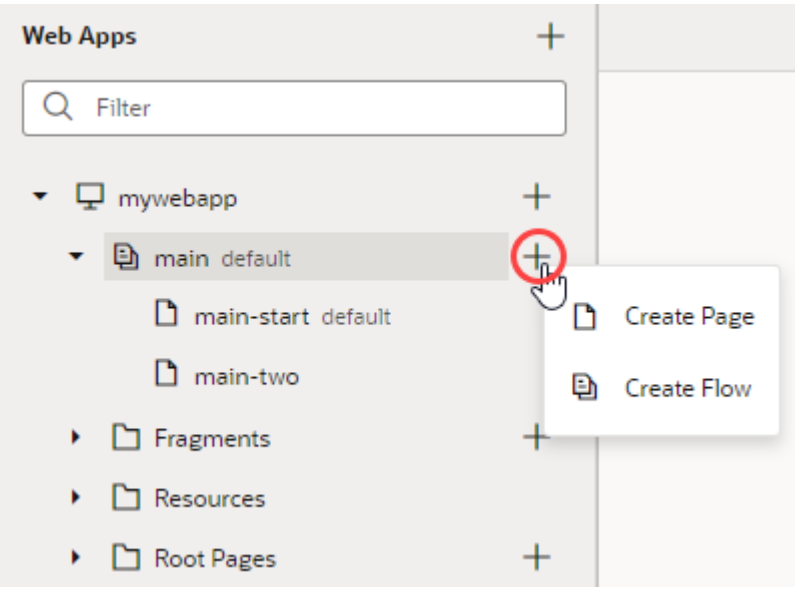
You can leverage Redwood resources in your application to create pages based on Redwood patterns.

Redwood provides several page templates, some of which are also available as patterns. A pattern defines the basic constructs of a page for common use cases and adds variables, actions, and event listeners to your page. Such patterns make it easier for you to create pages by eliminating time-consuming and error-prone work, where you might need to add individual components to a page and manually wire up the necessary actions.

> **Note:**
>
> Creating a page based on a pattern is supported only if your app is leveraging version 2301 or later of the Redwood components.

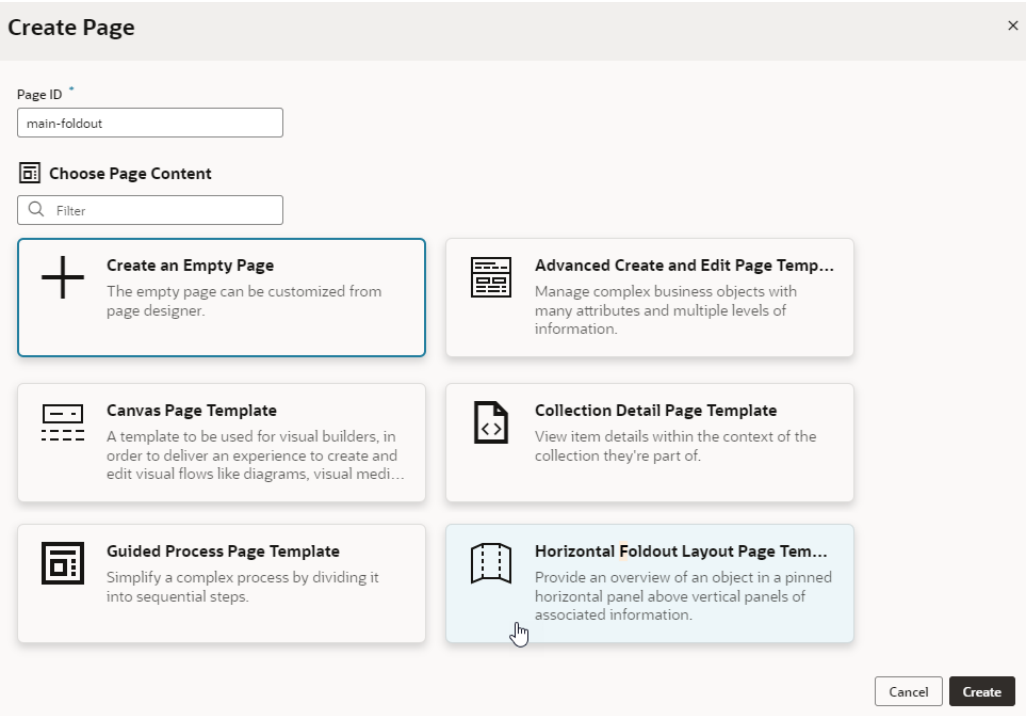
1. To add a pattern to your application:
   a. Open the Components tab ( ) in the Navigator.
   b. Select the **Browse** tab and search for "redwood pattern". Click a pattern to learn more about it.
   c. When you know which pattern you want to use, click **Install** (or **Install Component** in the canvas area). If prompted, make sure you install or update other dependent components.

      Once you install a pattern, it becomes available to you when creating a page.
2. To create a page based on a pattern:
   a. Open your application and expand the flow where you want to add a page.
   b. Click the **Create Page** icon ( ) next to the flow, then select **Create Page**.



c. Enter the name of the new page in the **Page ID** field of the Create Page dialog box. By convention, a page name has its flow name as a prefix.

d. Under **Choose Page Content**, select the pattern that you want to use with the page. If there are too many patterns, filter to find the one you want.



If your application includes fragments for use in pages, those will also be available to you in the Create Page dialog.

e. Click **Create**.

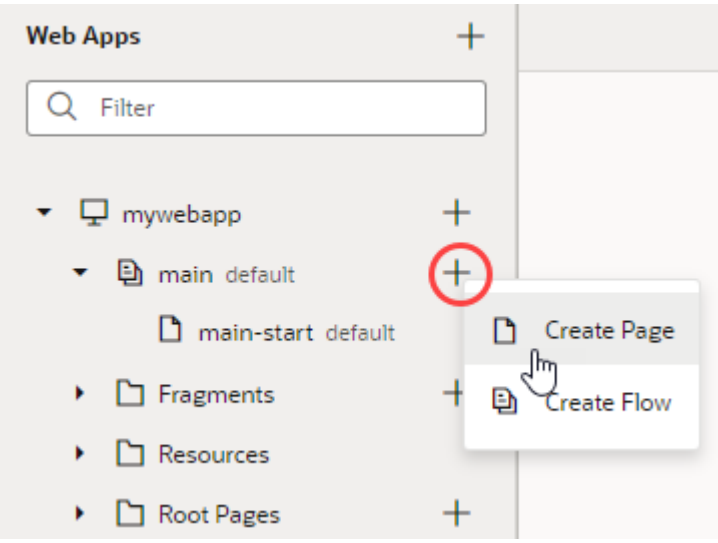
A new page, with the contents of the pattern added to it, opens in the Page Designer. The page contains all the variables, events, listeners, and action chains required by the pattern to work and can now be customized as needed.
If the page doesn't open in the Page Designer and instead you see only a spinning icon, the page template might be outdated. To fix this, you can:

- Update your Redwood components to the latest version.
- Upgrade Your App.
- Clear your browser cache.

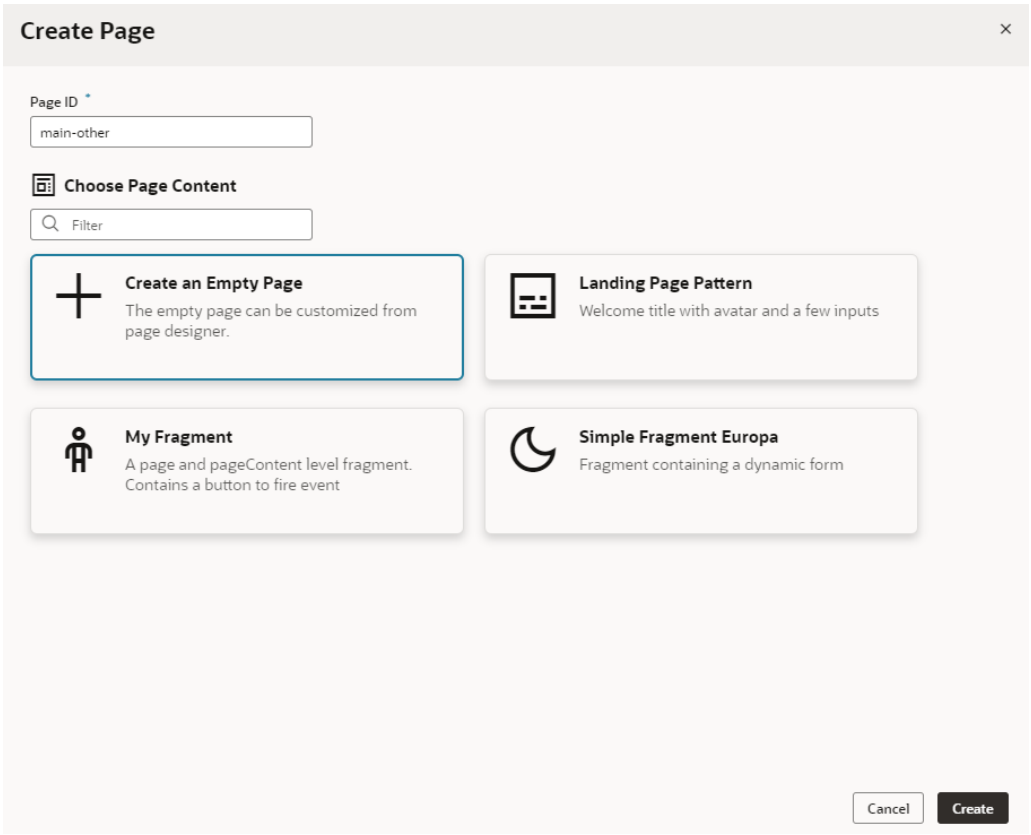
Create Pages From Fragments

You can create pages starting with the contents of a fragment, essentially using the fragment as a page template. You can do this so long as the fragment is available for use in pages.

1. To make your fragment available as page-level content:
   a. Open the fragment and click **Settings** to open the Settings editor.
   b. In the **Used For** field, select **page**.
   c. Set a custom icon for the fragment. The icon can then be used to identify the fragment in the Create Page dialog as well as the Flow Diagram.
2. To create a page using a fragment as the page's template:
   a. Open your application and expand the flow where you want to add a page.
   b. Click the **Create Page** icon (+) next to the flow, then select **Create Page**.

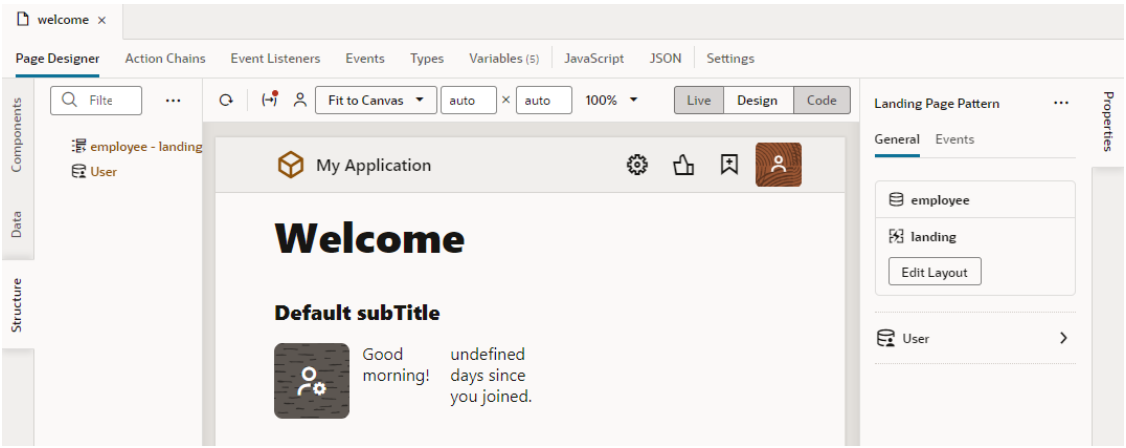


   c. Enter the name of the new page in the **Page ID** field of the Create Page dialog box. By convention, a page name has its flow name as a prefix.
   d. Under **Choose Page Content**, select the fragment that you want to use with the page.



e. Click **Create**.

A new page, with the contents of the selected fragment added to it, opens in the Page Designer. You can now wire up the page for the necessary fragment parameters.
If your fragment's properties were customized, what you see on the page's Properties pane will be different. Instead of the standard page-level properties, you'll see fragment properties that the fragment author chose to highlight. Here's an example where the author has customized the fragment's properties, so that some input parameters show in a separate section when the page is selected:



(To view the page's Properties pane, make sure no component or element is selected on the page. If you need to, simply click an empty area on the page.)

The page's Structure view also changes, with the fragment considered the root element, instead of the page.

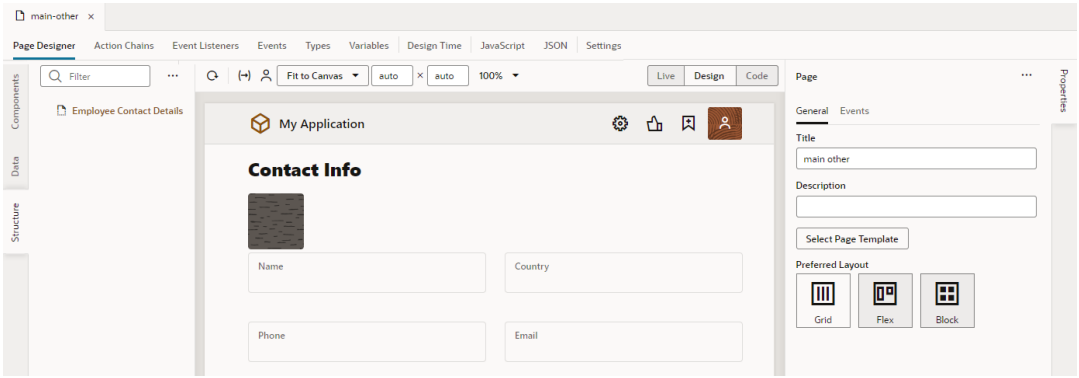
Change Page Templates

When you create pages from templates, you can change patterns and fragments associated with a page, even remove them completely if you want. This way, you keep the same page name but change whatever else is on the page.

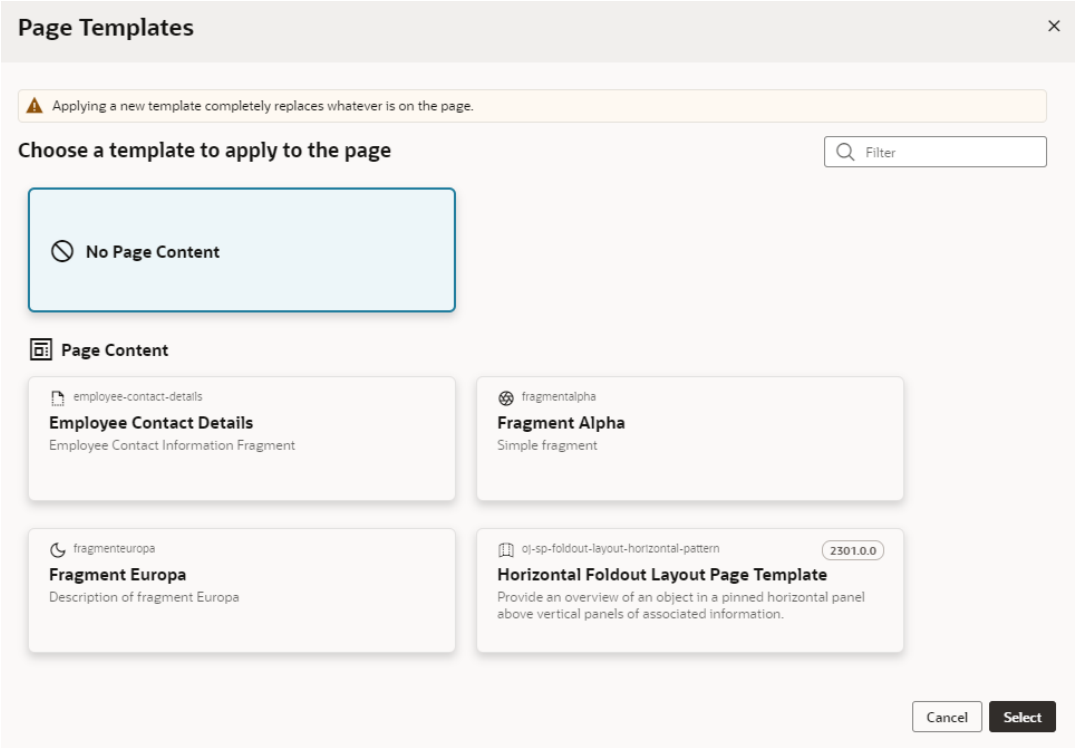
> **Caution:**
>
> Applying a new template removes everything on the page, including associated files—it's as if the page was deleted and a new one created with the same name. Previous user work is removed and all content is reset to an initial state (which may be an empty page or a basic template). Proceed with caution.

To update or remove a pattern or fragment on a page:

1. Open the page with the associated pattern or fragment. For example, here's a page that was created with the `employee-contact-details` fragment as a page template:



2. To view the page's Properties pane, make sure no component or element is selected on the page. If you need to, simply click an empty area on the page.
3. Click **Select Page Template** in the Properties pane.
4. In the Page Templates dialog, make your choice:



- To clear all existing content on the page, select **No Page Content**.
- To replace existing content with a fragment, select the fragment you want to use. The fragment you want to use must be tagged with the `page` metadata in the **Used For** field, which allows it to surface as page content.
- To replace existing content with a pattern, select the pattern you want to use. The pattern you want to use must be installed to your application from the Components tab in the Navigator.

5. Click **Select**.

If you replaced page content with a new pattern or fragment, you can now wire up the page for the necessary parameters.

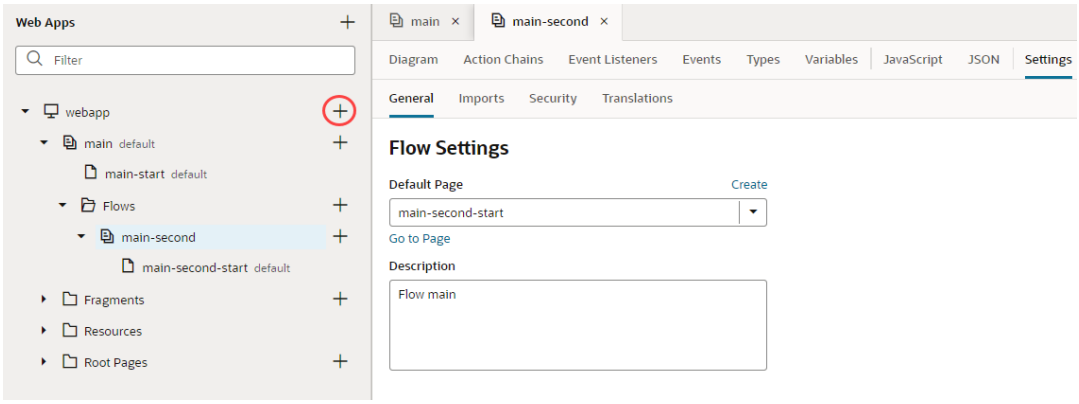
Create and Manage Flows

You can create a flow to group pages that you want to treat as an independent unit that performs some function in your application, for example, a flow that contains pages and artifacts to register a new user. Your application can have multiple flows, even a flow within a flow.

> **Note:**
>
> Typically, you create flows in the Page Designer, but if you want to visually build flows with pages and action chains, use the Flow Diagram view. See Add Pages and Action Chains to a Flow in Diagram View.

To create a flow in your application:

1. Open your web (or mobile) application in the Navigator, then click the Create Flow icon (+) next to the application node:



2. In the Create Flow dialog box, enter a name for the flow and click **Create**.
Creating a flow creates a start page for that flow automatically and sets it as the flow's default page.

   Depending on your application type, one or more flows are created by default when you create an application. For example, in web apps that use the None navigation style, the default flow is named `main`. The `main-start` page (created as the start page for the `main` flow) is set as the default page for that flow, and the `main` flow is set as your application's default flow, details you can view in the app's Settings editor.

   You can expand a web (or mobile) application's flow in the Navigator to see pages and artifacts within a flow:

Artifact	Description
Flow artifact	Open the flow artifact in the Designer to edit metadata such as variables, types, action chains, and JavaScript functions that can be used on every page in the flow. Click the flow's **Settings** editor to set the default page in the flow. You can expand the flow artifact in the Navigator to see pages contained in the flow. To create a new page in the flow, click the Create Page icon (+) next to the flow artifact, then **Create Page**. To embed pages within other pages, you can create sub-flows (see Embed a Flow Within a Page).
Page artifact	Open each page artifact in the Designer to edit the page's layout and other page metadata.

   Each flow has access to the application's `Resources` folder containing images and translation files that can be used in the flow's pages. See Work With Application Resources.

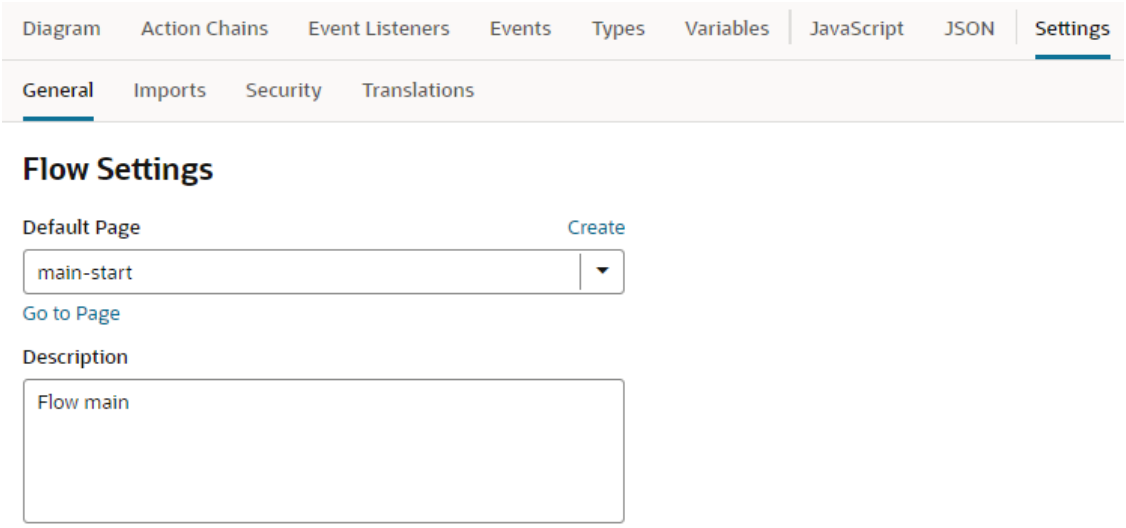
Manage Flow Settings

Each flow in your application includes a Settings editor, which you use to manage its default page as well as imported resources such as custom components, CSS files, and modules. Just as with page-level settings, you can manage security and create flow-scoped translation bundles. 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 flow artifact's Settings editor.

Because an application can have multiple flows, you configure settings for each flow individually, so your settings apply only to pages within that flow.

To configure settings for an application's flow, open the flow, then click **Settings** to open the Settings editor:



Here's how you can use the different flow-level settings:

Setting	Description
General tab	Manage general flow settings:
Default Page	Default page of the flow. Every application has a default flow (defined in the root page's settings), and every flow has a default page. The default page serves as the entry page for the flow and, by default, is set to the start page created automatically when the flow was created (for example, the `main-start` page created for the `main` flow). When the app is run, the default page of the default flow is rendered. Select a page to change the flow's default page. You can click **Create** to create a page directly from here and set it as the flow's default page, then use the **Go to Page** link to design the page in the Page Designer.
Description	Optional description of the flow.
Imports tab	Manage resources such as custom CSS files, modules, and components imported at the flow level, allowing you to create declarative references in the flow's pages to those resources. See Manage Custom Component, CSS, and Module Imports.
Security tab	Add user roles (defined at the visual application level) to control access to the flow. Only users granted one of the assigned roles can navigate to the flow. Note that permissions are inherited from the parent, so the flow inherits permissions from the application. See Restrict User Access to an Application, Flow, or Page.
Translations tab	Create translation bundles for the flow, in addition to the app-level translation bundles, for use with a third-party translation tool. If you create a translation bundle for a flow, strings and keys are added to the flow's bundle when you externalize strings in the flow's pages. Strings in other flow's pages are not added to that bundle when they are externalized. See Create Translation Bundles.

1. Open the flow artifact's Settings editor.
2. In the General tab, use the Default Page drop-down list to select the page in the flow that you want to be the default. If you want, click **Create** to create a new page and set it as your default.

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 tab of the page's Settings editor.

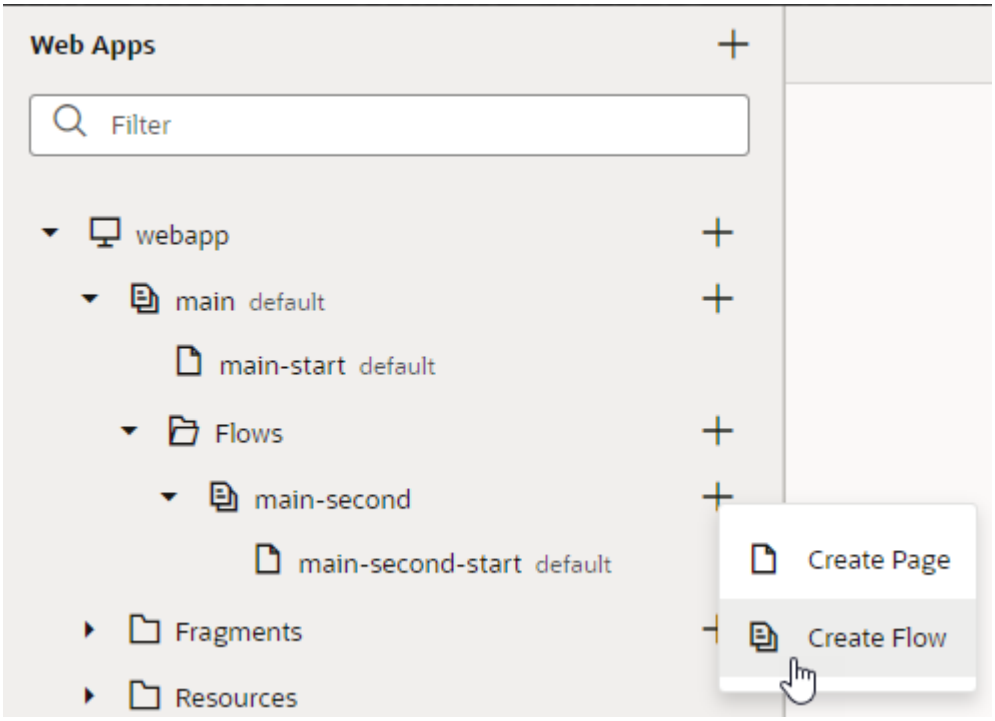
In the page containing the embedded flow, you can only edit the page content outside the Flow Container component, and the embedded pages are not visible on the canvas. After setting the default sub-flow for the page, you can run the app to see the embedded content in the page.

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 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.

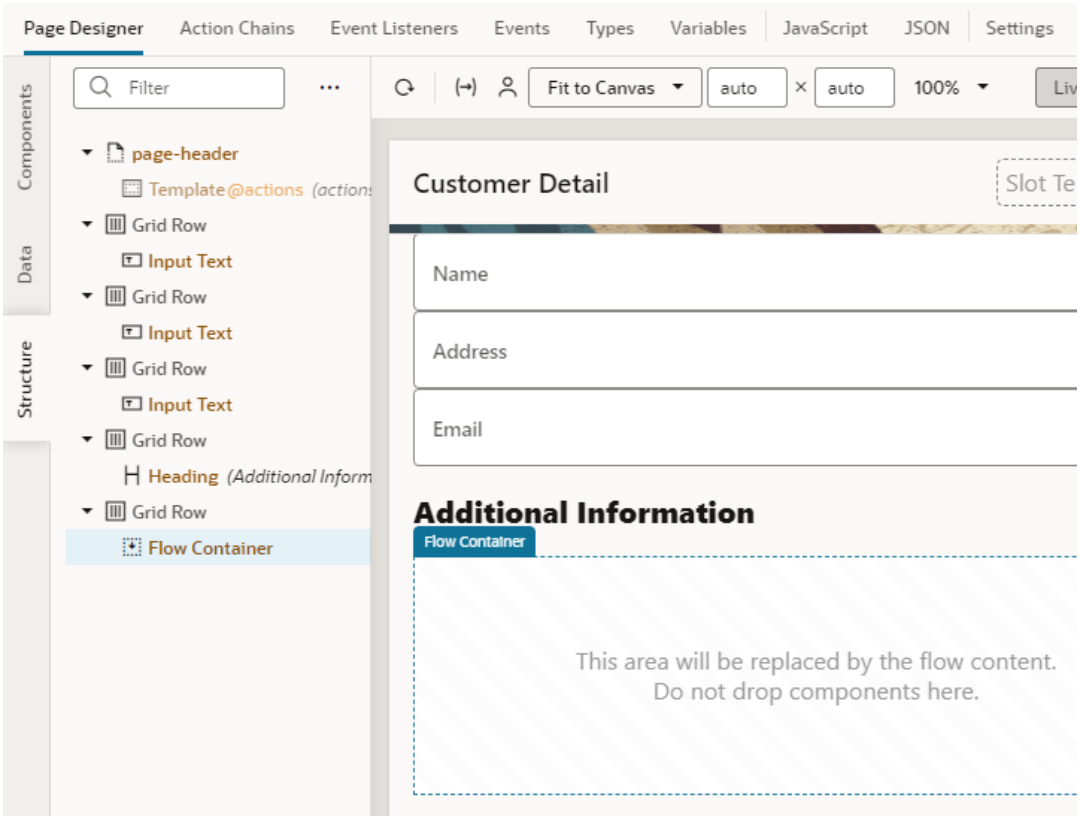
To embed a flow in a page:

1. In the Navigator, locate the flow containing the page where you want to embed the flow.
2. Click Create Page (+) next to the flow containing the page, click **Create Flow**.



3. Enter a name for the new sub-flow in the Create Flow dialog box and click **Create**.

4. Open the page where you want to embed the new flow.
5. In the Page Designer, drag the Flow Container component from the Layout category in the Components palette and place it on the canvas.



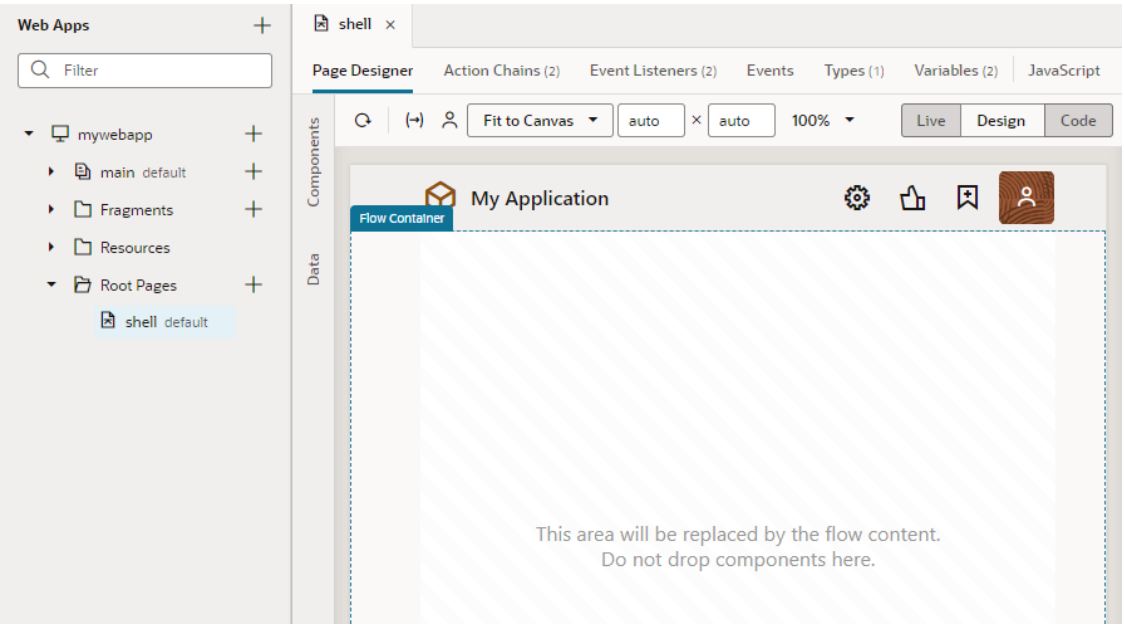
6. In the page's Settings editor, select the default sub-flow in the General tab.

   The Default Flow drop-down list displays all flows within the current flow.

Customize Your App's Root Page

A root page is a special type of page which provides the entry point for your app. It typically contains a shell with common elements such as the header (with the application title and logged-in user info), the footer, and a container for the content you create.

All pages in your application are created within flows, except for root pages. If you look at your app's tree view, you'll see the `Root Pages` node at the same level as your app's flows (`main`, for example):



The `Root Pages` node contains the default root page, which is named `shell` for web apps and `app` for imported mobile apps. The root page embeds the flow that is invoked when your app is first launched. It also describes areas outside the page flow for elements such as the header, footer, and other navigational elements.

An application typically contains one root page which can be completely customized, but you can choose to create a custom root page, say to provide your own header that will appear on all your application's pages. Root pages are created similar to standard pages, except that you click + next to the Root Pages node in the Navigator. After the root page is created, you can design it as needed. The default root page for all web apps includes a header and a footer, defined in separate fragments for reusability. Your custom root page will not automatically include these elements.

> **Note:**
>
> When your application first runs, it launches the default root page (defined in the app-level Settings editor), which in turn opens the default page in the default flow (defined in the root page's Settings editor). Remember to change these settings if you create custom root pages or want to use a different flow as the starting flow for your app.

Edit an App's Header, Footer, and Navigation Items

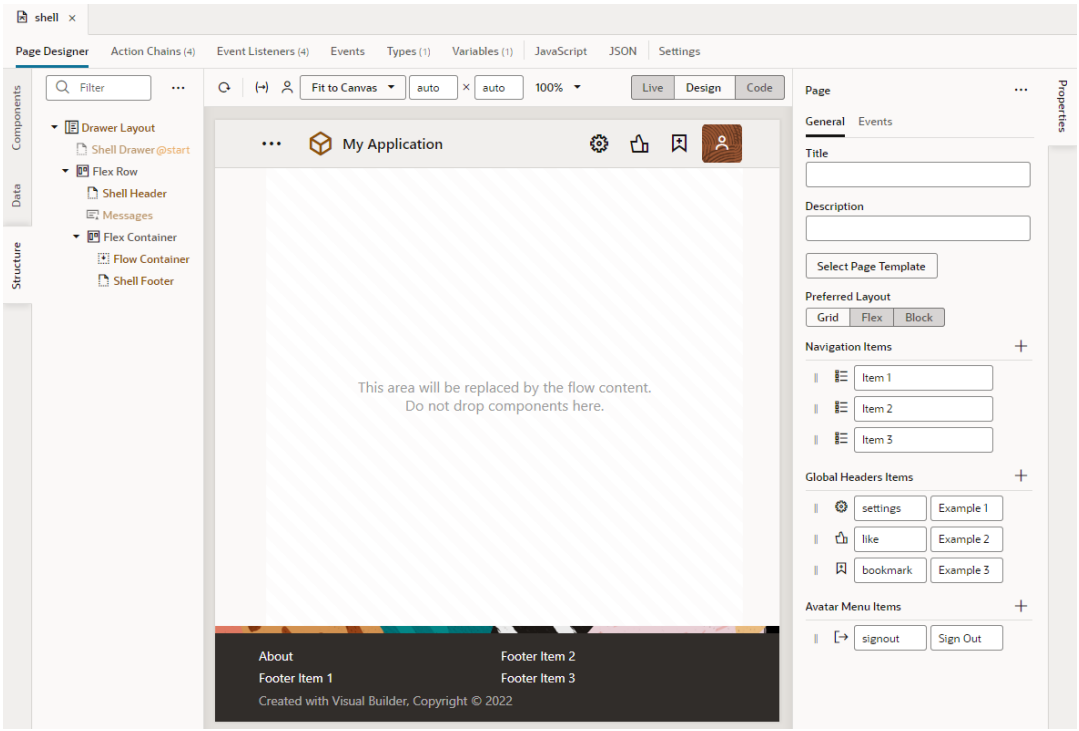
You can edit your app's default header and footer elements to add a description, add graphics like a company-specific logo, or insert text functions. You can also customize navigation items built into the web app templates, even add new ones after the app is created.

1. Select the root page under the **Root Pages** node in the Navigator.

   The header and footer elements for these apps are defined in separate fragments and consumed by the app's root page in fragment containers. The default root page is named `shell` and set as the default entry page when a web app is first created. This entry page contains fragments that define the header, the footer, and navigation items such as tabs

and navigation drawers. It also contains a flow container where the app's default flow is embedded.

For example, here's the `shell` page for an app that uses the Navigation Drawer template:

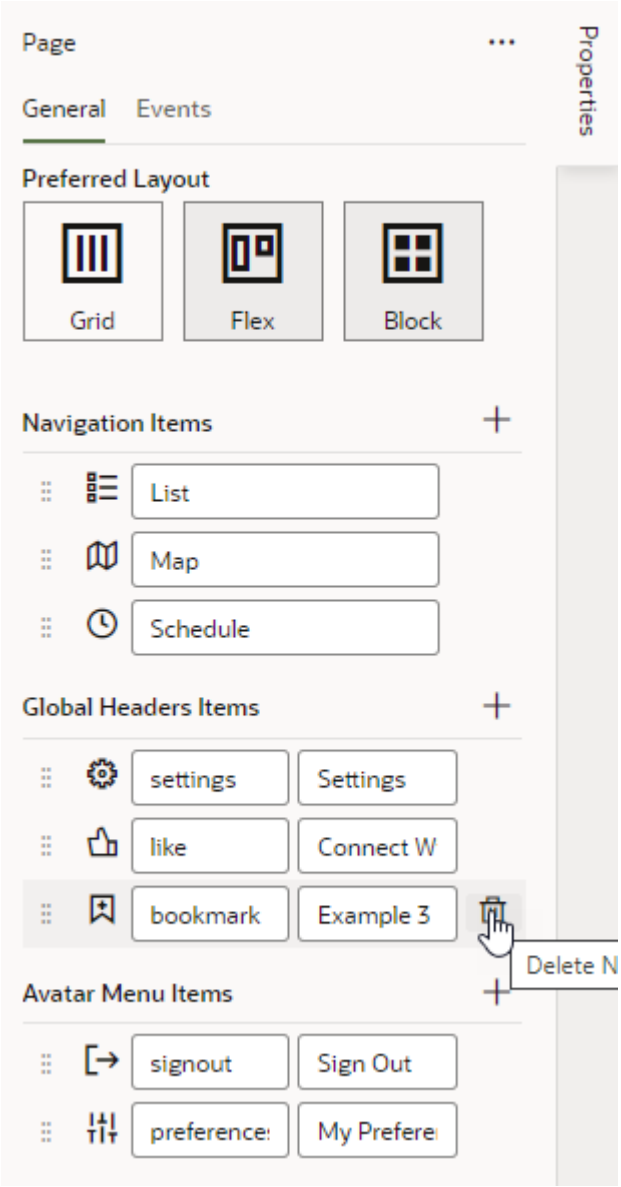


When the entire page is in focus without any component selected, you can use the options in the Properties pane to customize the app's header elements. By default, the header includes the Settings, Like, Bookmark, and Avatar elements. You can also customize the navigation items built into the Navigation Drawer and Bottom Tabs templates or add new items if you use the None template.

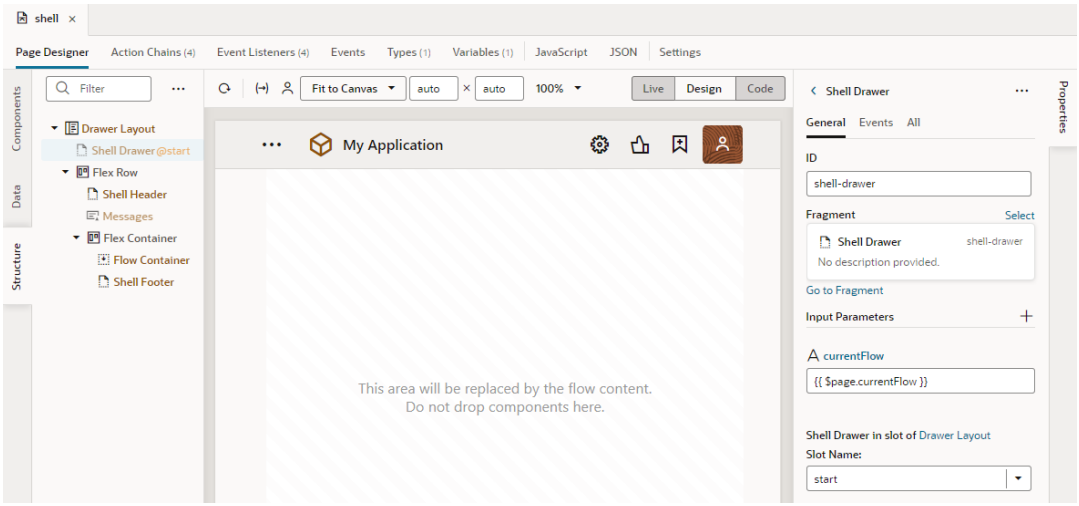
For each option, you can:

- Update the default label in the text box.
- Click the icon to choose a different one from the Redwood icon gallery.
- Click to add a new item.
- Click to delete an item.
- Drag to reorder an item.

Here's an example:



2. Select a fragment to view and edit its properties. You can select the component on the canvas, but sometimes, it might be easier to select the component in the Structure view, as shown here for the `Shell Drawer` fragment that defines the navigation drawer:

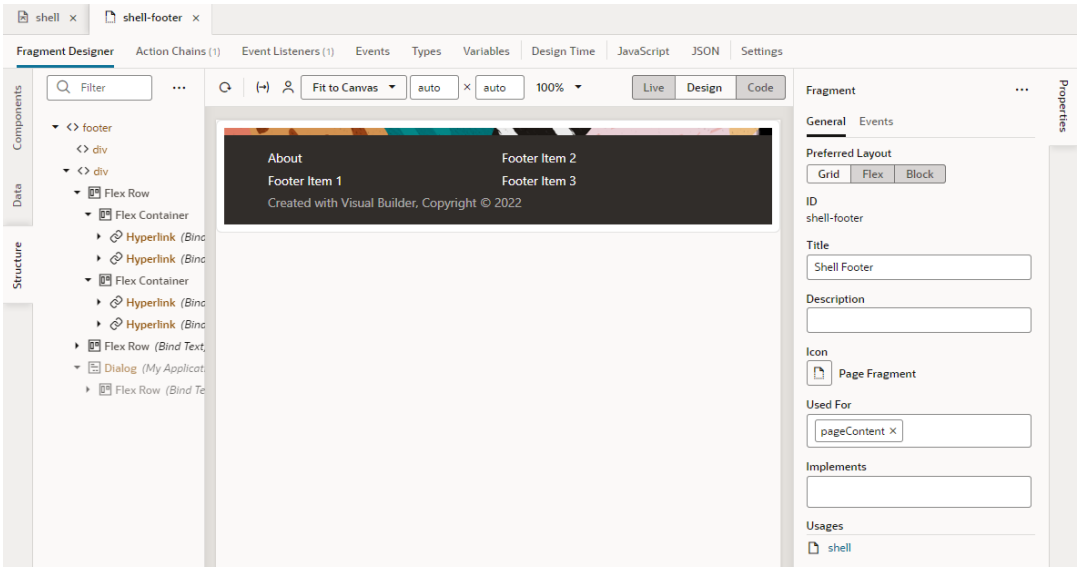


Because fragment variables can be enabled as input parameters to a page, input parameters defined for the fragment become available to you under **Input Parameters** in the fragment's Properties pane. You can click **Go to Fragment** if you want to access and update the fragment in an editor.

You select different fragments to access the editor for the header and footer fragments. Alternatively, you can access each fragment directly from the **Fragments** node in the Navigator, as described in the next step.

3. If you want to further customize header, footer, and navigation items, click the **Fragments** node in the Navigator, then select the fragment you want to edit. To update header elements, select `shell-header`; to update footer elements, select `shell-footer`. Use `shell-drawer` to update the main navigation panel that comes with the Navigation Drawer template.

   For example, here's the `shell-footer` fragment for a web app:



In the Design view of the Fragment Designer, you can select a component and edit its properties in the Properties pane.

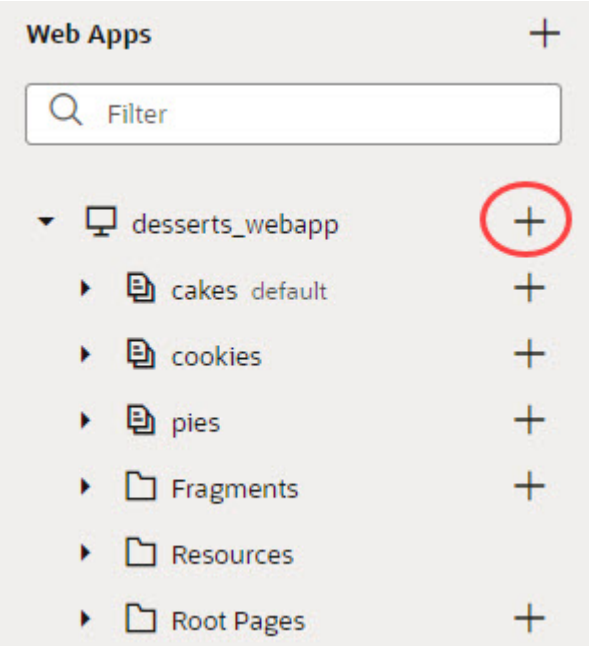
The Fragment Designer is similar to the Page Designer, except that it builds a fragment instead of a page. You can add components to your fragment and bind them to events,

action chains, variables, and functions, much like what you'd do when developing a page. You can also pass parameters from a fragment to the pages consuming it, so you have the option of overriding the default parameter value with an alternate value on a particular page. For more on fragments, see Work with Fragments.

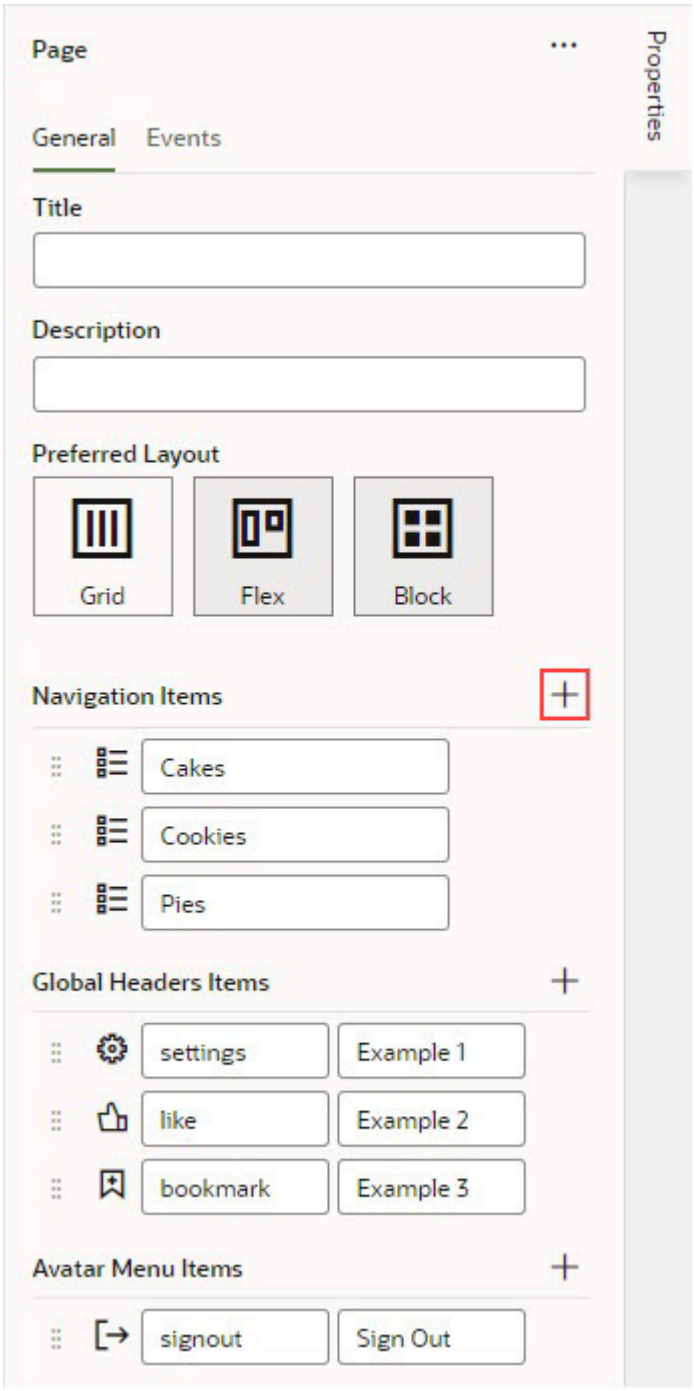
Add a Navigation Item for Navigation Drawer Apps

To create a new navigation item for your web app based on the Navigation Drawer Template, and to add it to the Navigation Drawer panel:

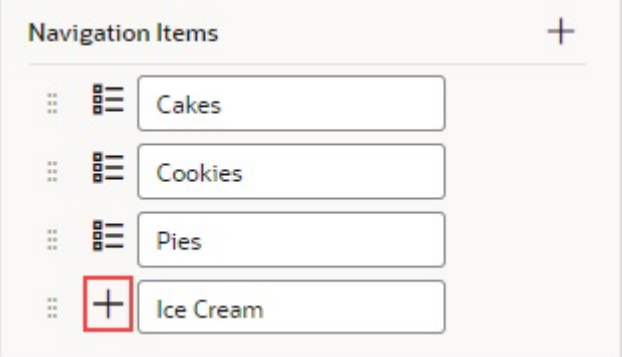
1. Click the **Create Flow** (**+**) icon next to the *application* node in the Web Apps pane.



2. In the **Flow ID** field, enter a name for the new navigation item and click **Create**.
3. In the **Web Apps** pane, expand the **Root Pages** node and select the **shell** page.
4. In the **Properties** pane, click the **Add Navigation Item** (**+**) icon in the **Navigation Items** section:

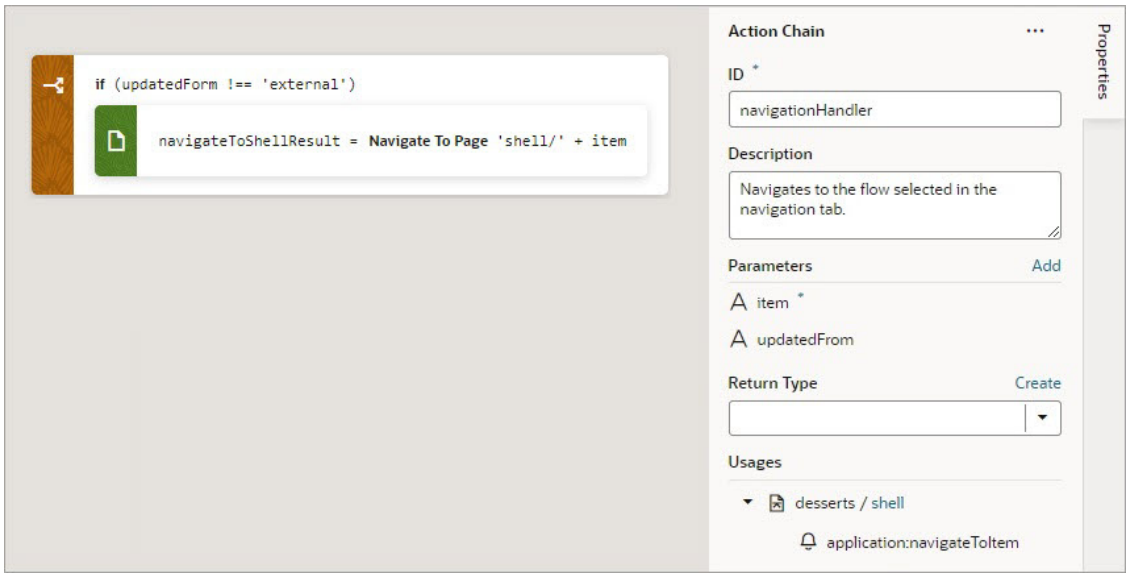


5. In the pop-up, select the appropriate page flow and click **Select**.
6. Click the **Select** (**+**) icon next to the new navigation item to select its icon, which users see in the Navigation Drawer panel.



7. If necessary, change the label for the new item, which users see in the Navigation Drawer panel.

To view the action chain that handles the navigation, on the shell page, click the **Action Chains** tab and select the **navigationHandler** action chain:



Navigate Between Pages and Flows

When you create multiple pages and flows, you can set up navigation to go from one page to another or from one flow to another.

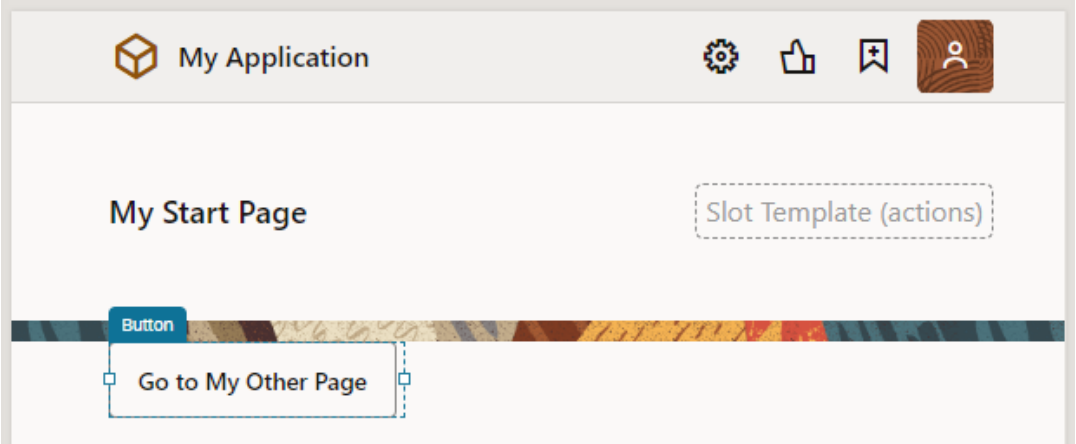
Navigate Between Pages in the Same Flow

To navigate between pages in the same flow, you associate a page component with an event that sets off a navigation action chain.

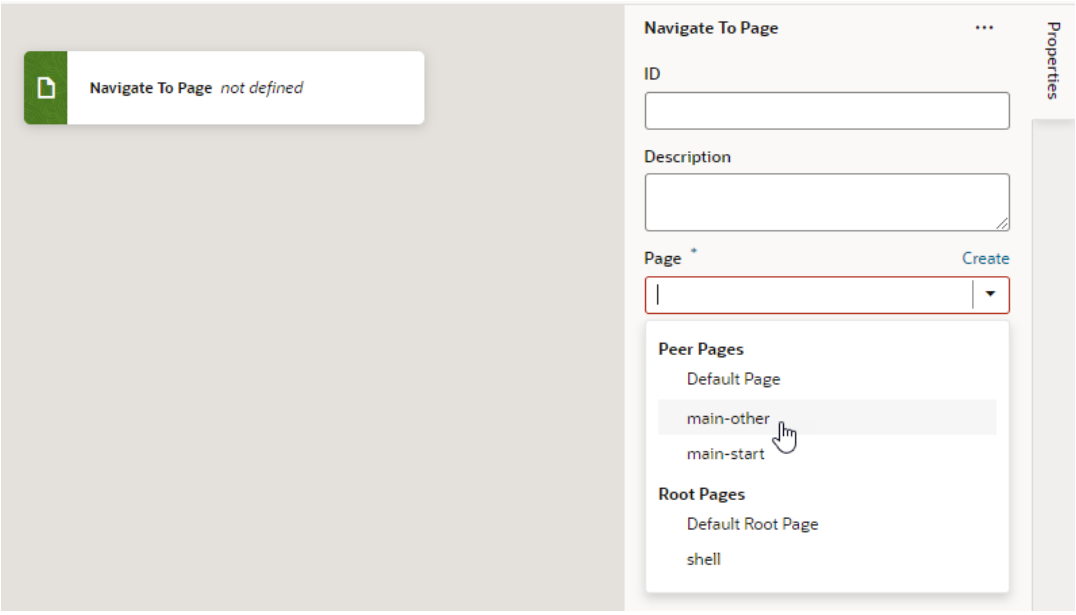
Let's say you've defined two pages within a particular flow: `main-start` and `main-other` within the `main` flow. And you want users to click a button on the `main-start` page to get to the `main-other` page. To do this:

1. Open the page you want to navigate from (`main-start`, for example).

2. In the Page Designer, drag a component that you want to set off navigation and drop it onto the page canvas. Here's an example of a button on a page:



3. Select the page component, then click the **Events** tab in the Properties pane.
4. Click the **+ Event Listener** button and select **On 'ojAction'**, the default action for a button click. You might see other options suggested for your particular component.
5. When a new action chain is created, drag the **Navigate to Page** action from the Navigation section of the Actions palette and drop it onto the canvas.
6. In the Navigate action's properties, select **Page** (if necessary), then select the page you want to navigate to from the Page drop-down list (for example, `main-other`).



7. Optional: To enable users to navigate back to the original page, associate a page component with an event that sets off a Navigate Back action chain:
   a. Go to the page that you set up navigation to (for example, `main-other`).
   b. Drag and drop a component onto the page canvas (for example, a button with the label `Back`).

c. Click the **+ Event Listener** button and select **On 'ojAction'**.

d. When a new action chain is created, drag the **Navigate Back** action from the Navigation section of the Actions palette onto the canvas.

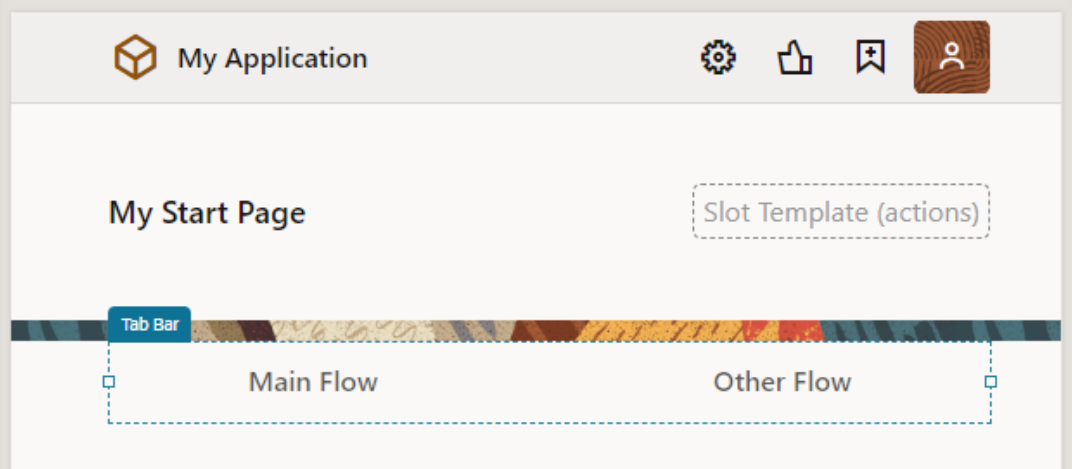
8. Preview your application to test navigation between the two pages.

Navigate Between Pages in Different Flows

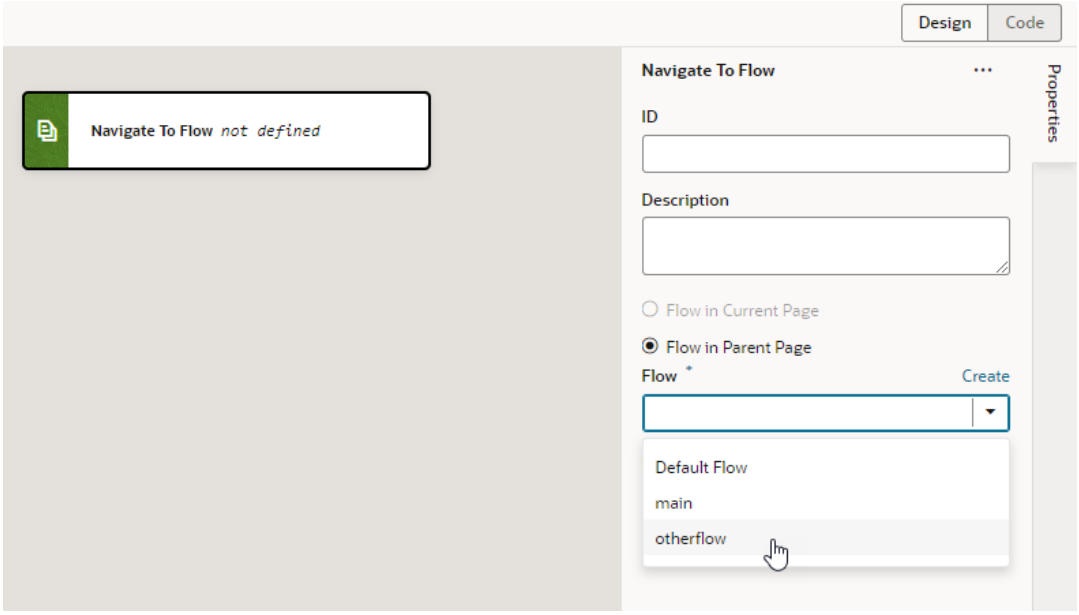
Navigating between pages in different flows within an application is similar to how you'd navigate pages within the same flow, but instead of selecting the page to navigate to, you select the flow containing the page.

To navigate between pages in different flows within an application (for example, to navigate from `myApp/main/main-start` to `myApp/otherflow/otherflow-newpage`):

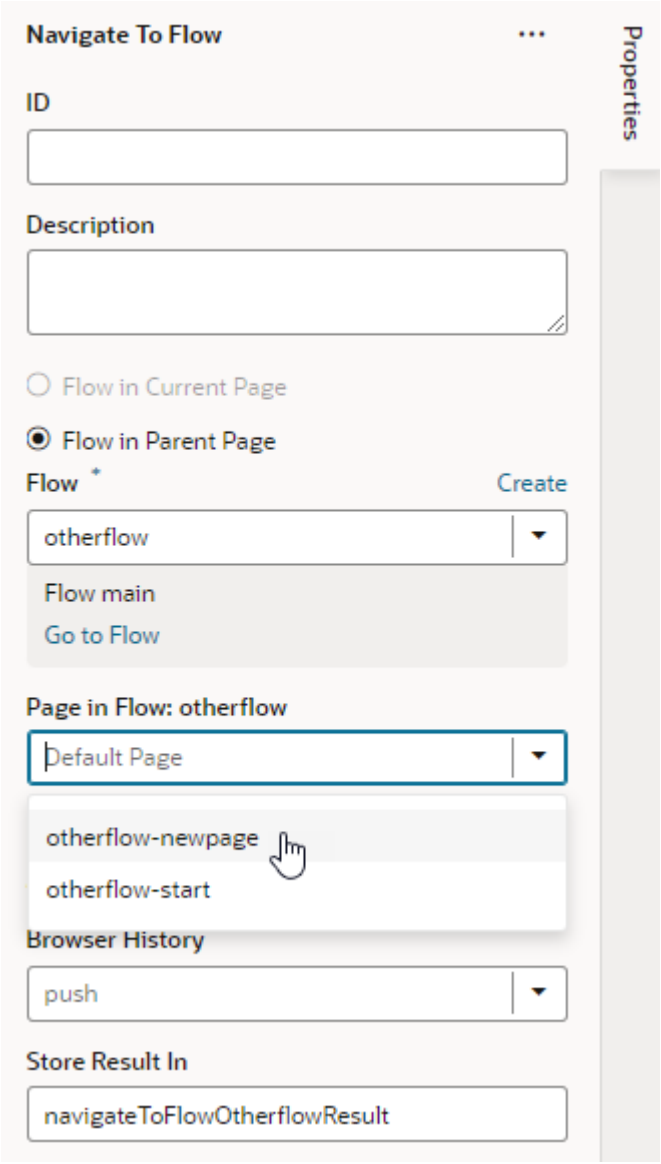
1. Open the page you want to navigate from (`main-start`, for example).
2. In the Page Designer, drag a component that you want to set off navigation and drop it onto the page canvas. Here's an example of a tab bar, with each tab meant to navigate to a different flow:



3. Select the page component, then click the **Events** tab in the Properties pane. In the tab example, you select the hyperlink nested within the tab bar component.
4. Click the **+ Event Listener** button and select **On 'click'**.
5. When a new action chain is created, drag the **Navigate to Flow** action from the Navigation section of the Actions palette and drop it onto the canvas.
6. In the Navigate to Flow action's properties, select **Flow in Parent Page**, then select the flow containing the page you want to navigate to from the Flow drop-down list (for example, `otherflow`).



7. By default, navigation to a flow navigates to the flow's default page. If you want to navigate to a page other than the default, select the page from the **Page in Flow** list (for example, `otherflow-newpage`).



8. Preview your application to test navigation from one page to another in a different 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 application'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 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.

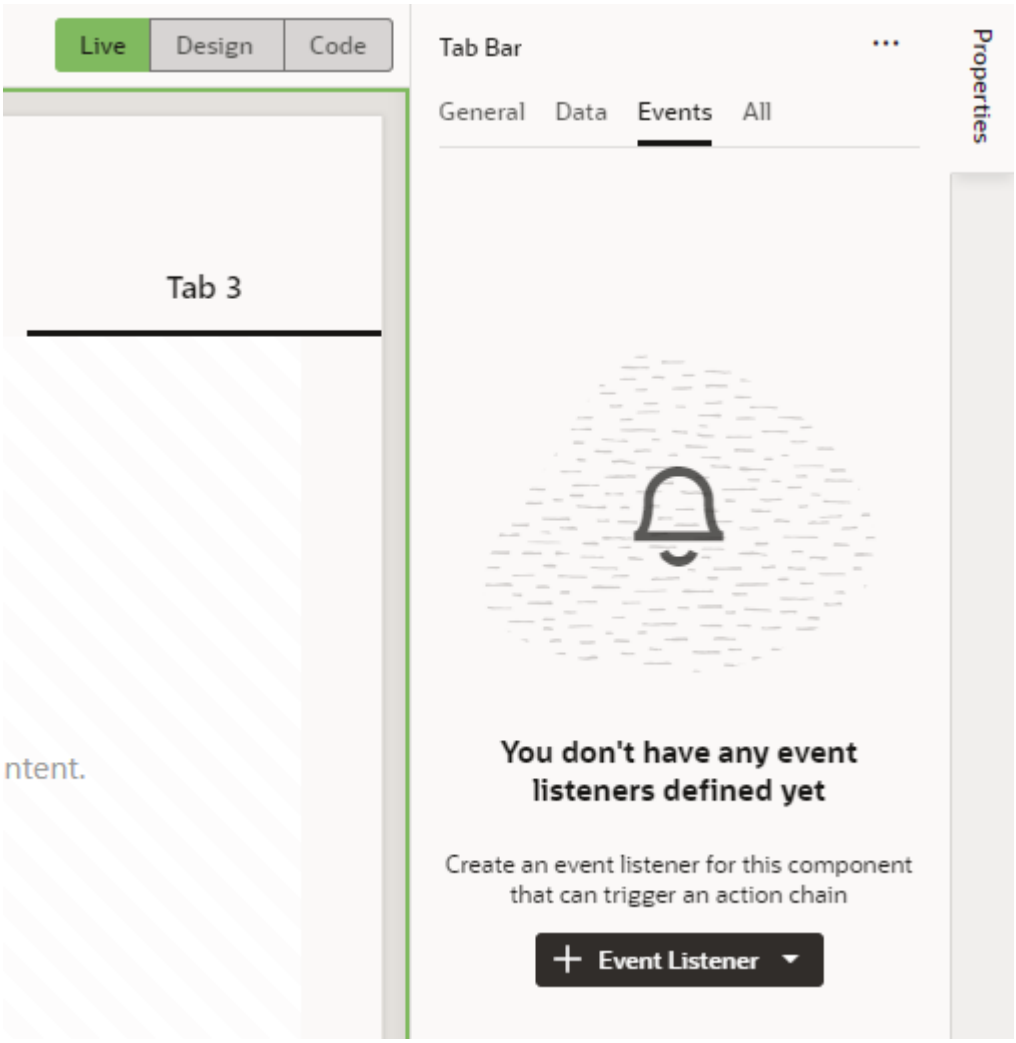
For example, when you create an app from a template that uses navigation elements such as a tabs or a drawer, the app will contain separate flows that can be loaded into the app's root page. If you create a web app with two flows (for example, "item-1" and "item-2"), and you want to use tabs in your 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.

First, for the UI component that's to open the start page of the flow, you need to create an event that starts an action chain to handle the navigation:

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'll need to open the root page and select one of the navigation tabs.



2. In the Events tab of the Properties pane, click **+ Event Listener** and select the suggested option (for example, **On 'selection'**).

   A new action chain is created and associated to the event, and you're taken to the Action Chain editor to implement the action chain.

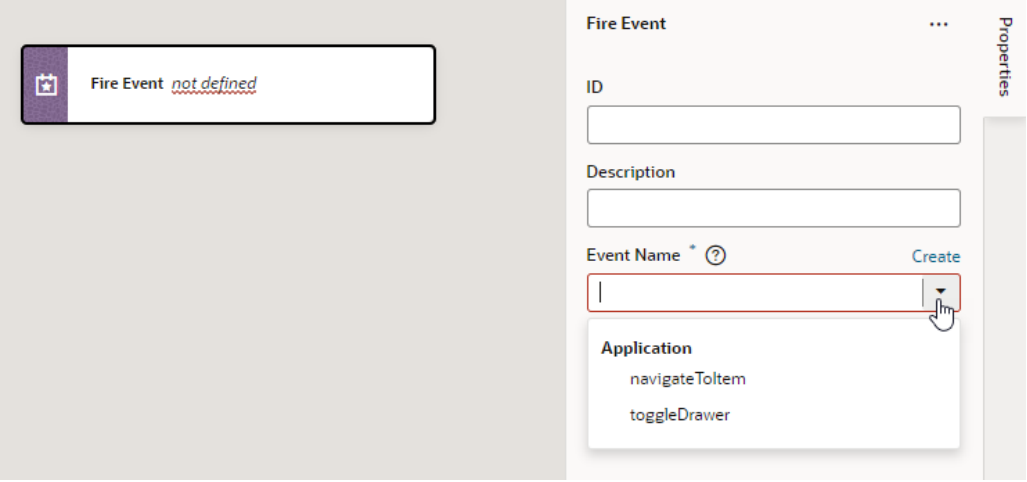
3. Now create the action chain:

   a. Enter a name for the action chain in the Properties pane's **Id** field.

   b. Add the **Fire Event** action to the canvas.

   c. In the Properties pane, select **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.



d. Under Parameters, hover over the `item` property and click to open the variables picker, then select the target flow (for example, `ojTabBar3010995061SelectedItem`).

8

Work With Components

Components are the basic building blocks of your application's user interface. You use them to build the layout of your pages and to add elements that display content or accept input from a user.

The components that you can use in your application's pages become available to you in the Components palette and range from static ones like headings and avatars to charts and gauges that visualize data, dynamic components that display content based on rules you define, even custom web components that you can create to support your app's specific needs. You also have access to Redwood layouts, styles, and templates based on the Oracle standard for user experience. You can use these components—all based on Oracle JavaScript Extension Toolkit (JET)—to create advanced layouts with rich data visualization capabilities.

Add a Component to a Page

You add a component to a page usually by dragging it from the Components palette in the Page Designer and dropping it onto the canvas or in Structure view. After the component is added to the page, you can set its properties and apply styling.

The Components palette contains an extensive list of Oracle JET components that you can add to your page, including dynamic components. Because dynamic components help you develop UIs that dynamically change what's displayed to users based on your own rules, working with them also involves *layouts* and *display logic*. See Add Dynamic Components to Pages.

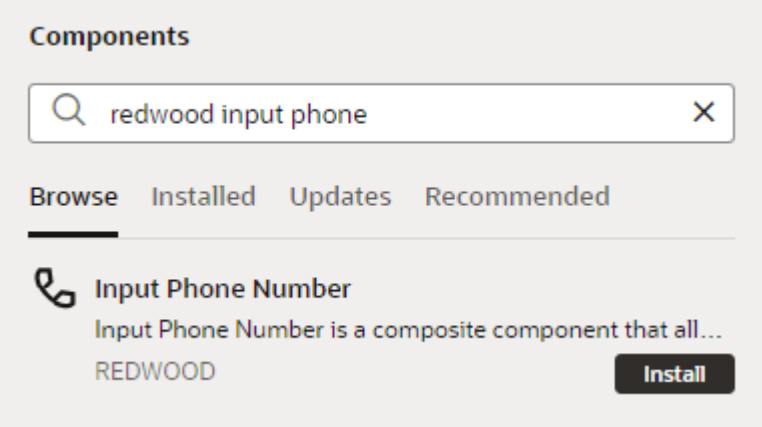
Visual Builder also gives you access to Redwood resources based on the Oracle standard for user experience. If you create a page using a Redwood pattern, the components of that pattern become available to you in the page's Components palette. You also have access to individual Redwood components that provide a rich user experience—from single UI elements, such as a button, to complex modules connected to backend services, such as a form—but you must install them from the Component Exchange before you can use them in your page.

> **Note:**
>
> If you'd like to use the next generation of JET components, known as Core Pack components, you can opt to show them in the Components palette under an **Early Access** category. Core Pack components provide new implementations that can improve performance, but do **not** support theming. If you use these components in your app's pages, you cannot theme your app. The only way to apply themes then is to roll back Core Pack component usage to Legacy components and re-implement your application. Be aware of this limitation before using Core Pack components in your app's pages.

1. If you want to use Redwood components in your page, follow these steps:
   a. Explore the Redwood component library and take note of what you'd like to use in your page.

b. Open the Components tab ( ) in the Navigator.

c. Select the **Browse** tab and search for the component you want to use. For example, you might use "redwood input phone" to look for the Input Phone Number component:



You can double-click the component to open it in the canvas area and learn more about it.

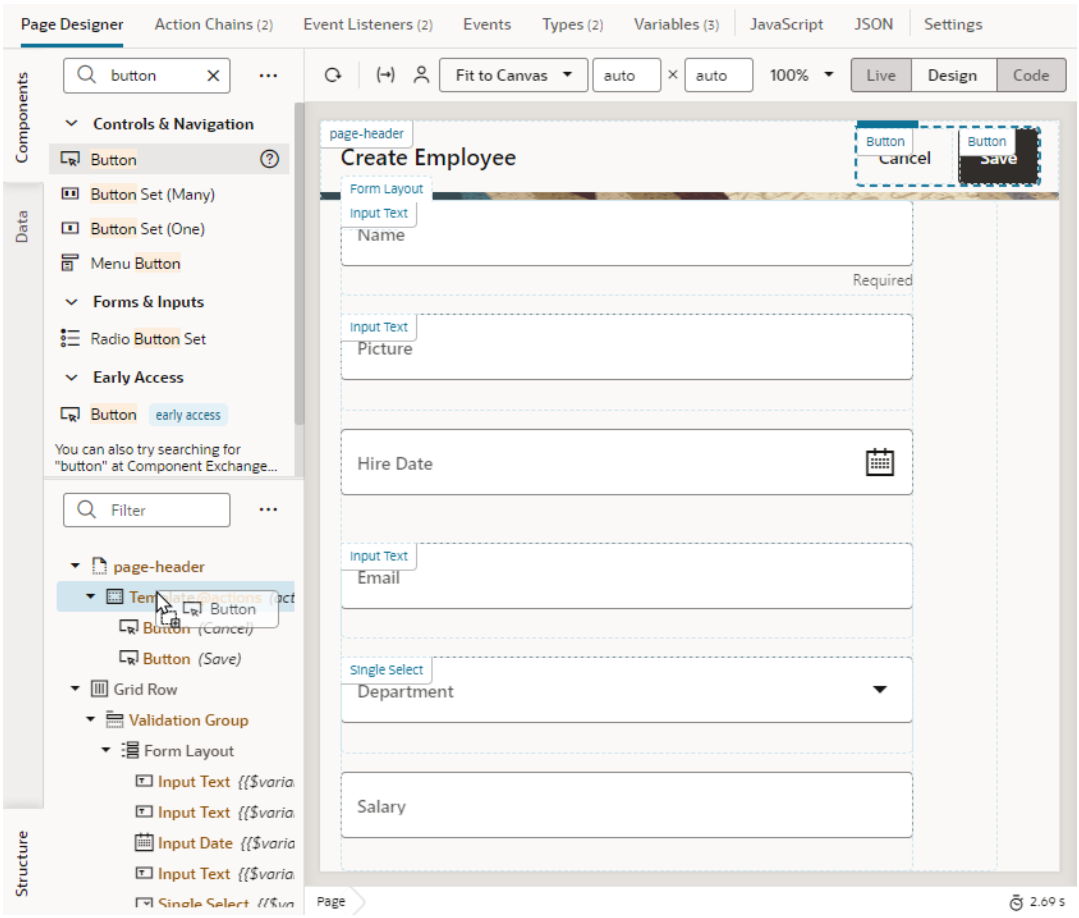
d. Click **Install** (or **Install Component** in the canvas).

Once you install a component, it will become available to you in your page's Components palette. Simply filter to find the component.

2. Open the page you want to work with in the Page Designer.

When the page opens in the Page Designer, you'll see a canvas used to display the page's layout, a Components palette containing a list of components, and a Structure view that displays a structural view of the page's components. You'll work primarily with these, though you can switch to the Data palette to display data in components suggested by Visual Builder.

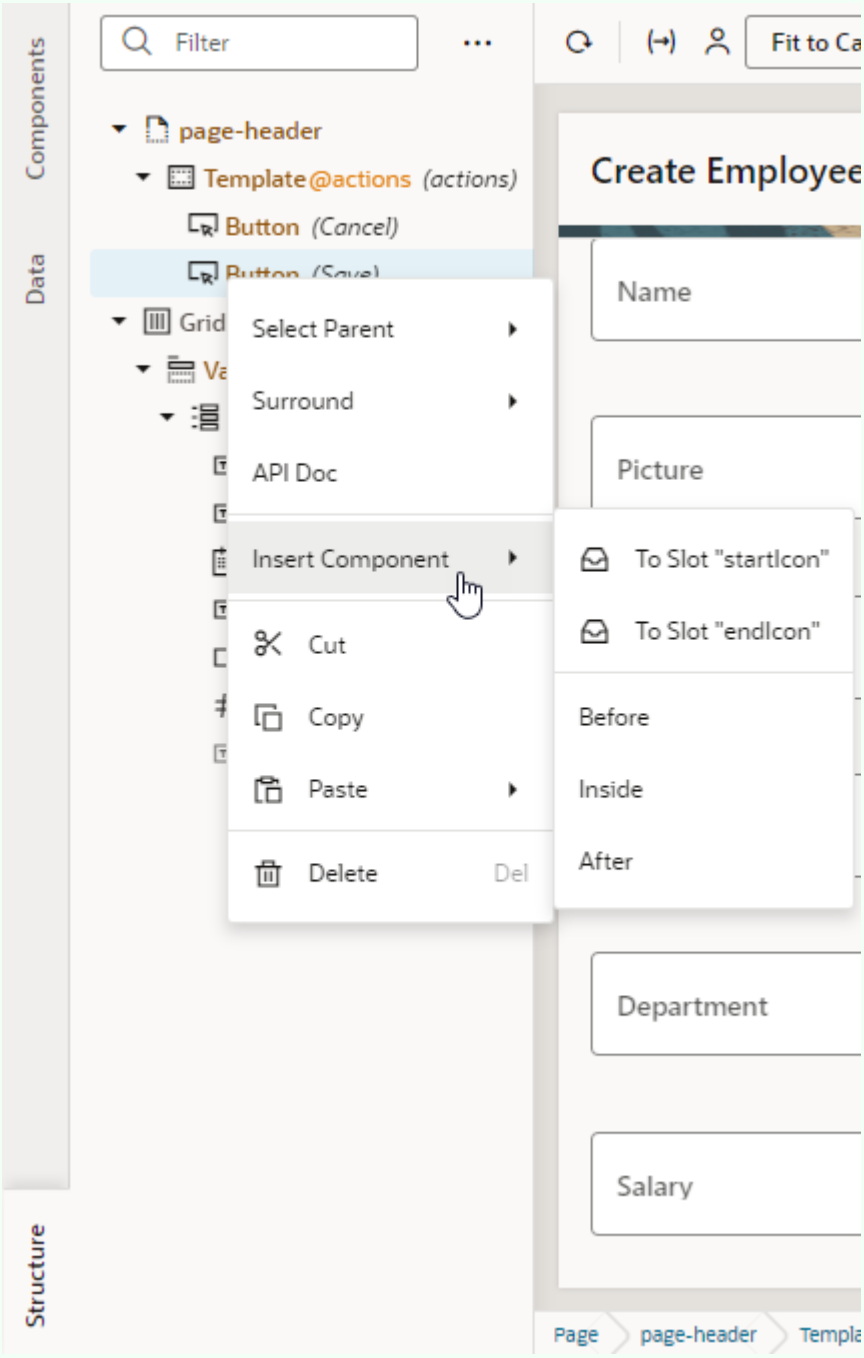
3. Drag the component from the Components palette and drop it into position on the canvas or in the Structure view. It is sometimes easier to locate, select, and position components in the Structure view:



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 will be displayed if you try to place a component on the canvas where the component is not allowed.

Tip:

You can also add a component by clicking **Insert Component** in the context menu. This option is available when you right-click anywhere on the canvas and in Structure view, both with and without a component selected. It's most useful in Structure view, especially in complex layouts, to help you position the component exactly where you want to add it. When you choose this option on a selected component, you'll be able to add a component before, inside, or after the selected one. If the selected component has multiple slots, you will have the option to drop the component into a particular slot as shown here:



4. After adding the component, you can define its behavior by editing properties that show in the Properties pane when the component is selected. Component properties are organized in tabs in the Properties pane. The component type will determine the tabs that show.

Component Properties	Description
General	Contains the most important properties of the selected component, such as layout and style. The slot value of a component inside a parent component's slot also shows here. Select the sub-component added to a slot (for example, a button's icon) and change its slot value to move it from the `startIcon` slot to the `endIcon` slot. This way, you can modify the slots of dropped components without accessing the HTML code. The slot used by the sub-component is also visible in the sub-component's All tab and can be modified there (for example, to bind it to a variable).
Data	Contains properties that are expected to be bound to data. The General tab and All tab also contain properties that can be bound to variables and expressions. See Bind a Component to Data.
Events	Used to bind a component's events to trigger action chains that define its behavior (for example, to open a URL when a button is clicked). See Start an Action Chain From a Component.
All	Contains more advanced component properties and shows all properties, including custom properties. Custom properties are those not defined in component metadata, for example, `data-*` attributes, and can be added by clicking **+** next to General Attributes.
Quick Starts	Contains a list of Quick Start wizards available for the component. When you add a collection component such as a table or list, this tab contains a list of wizards to help you add some actions that are typically associated with the component, such as mapping the collection to data and adding Create and Detail pages.

Opt In to JET Core Pack Components

It's possible to build your app's pages using the next generation of JET components, known as Core Pack components. Core Pack components can improve your app's performance—but they do **not** support theming. If you don't ever plan to theme your app, you can choose to enable Core Pack components in the Components palette for use in your app's pages.

> **Caution:**
>
> Core Pack components were previously available under the **Early Access** category in the Components palette. If you use these components in your app's pages, you cannot apply themes to customize the app's appearance for your users or switch your work environment's color palette. The only way to apply themes then is to roll back Core Pack component usage to Legacy components and re-implement your application.

Core Pack components, written entirely using the VComponent API and the JET Virtual DOM architecture, provide new implementations that improve performance and, in many cases, introduce extra functionality. For more information, see Core Pack overview in JET documentation.

To use Core Pack components in your Visual Builder application, you must first enable them to show in the Components palette. Once enabled, they become available to you under the **Early Access** category as a completely new set of components. As a result, you can use both Core

Pack and Legacy components side by side. Core Pack components have a DOM element prefix of *oj-c-** (instead of the existing *oj-** prefix used by Legacy components) and are badged early access for easier identification.

To enable Core Pack components for use in your application:

1. Switch to **Source** view, then select **visual-application.json**.
2. Add the `"corePackComponentsEnabled": "onAndAcceptTheAppCanNoLongerBeThemed"` flag to the file as shown here:

```
    "features": {
         "corePackComponentsEnabled":
"onAndAcceptTheAppCanNoLongerBeThemed"
     },
```

3. Refresh your browser to make the change take effect.
4. Switch to your app's pages to see Core Pack components added to the **Early Access** category in the Components palette.

   Hover your cursor over a component in this category to view its Info icon, then confirm that the component's tag name starts with *oj-c-*.

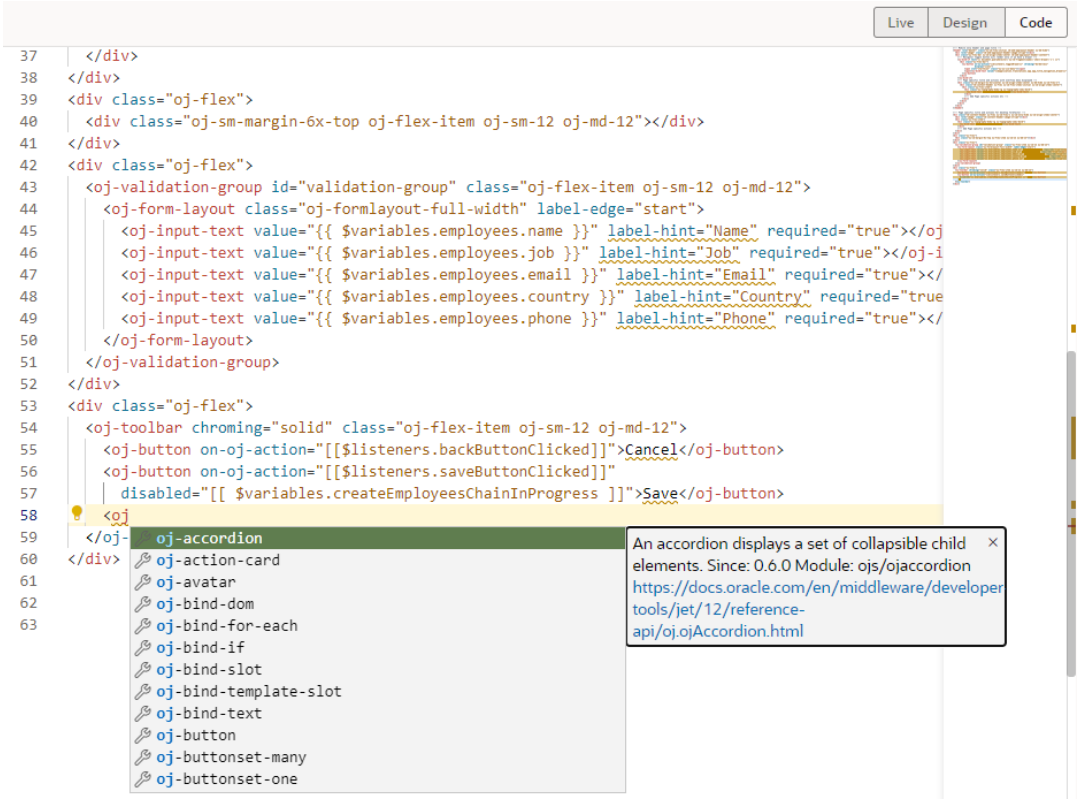
Add a Component Using Code Completion

Edit a page's HTML in the Page Designer's Code view, where you can use code completion to add components to the page. You can also drag components from the Components palette and drop them directly into valid places in the code editor. It's also possible to use standard HTML5 tags to extend functionality.

To add a component to a page in Code view:

1. Open the page in the Page Designer.
2. Click **Code** to open the page in Code view.
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.

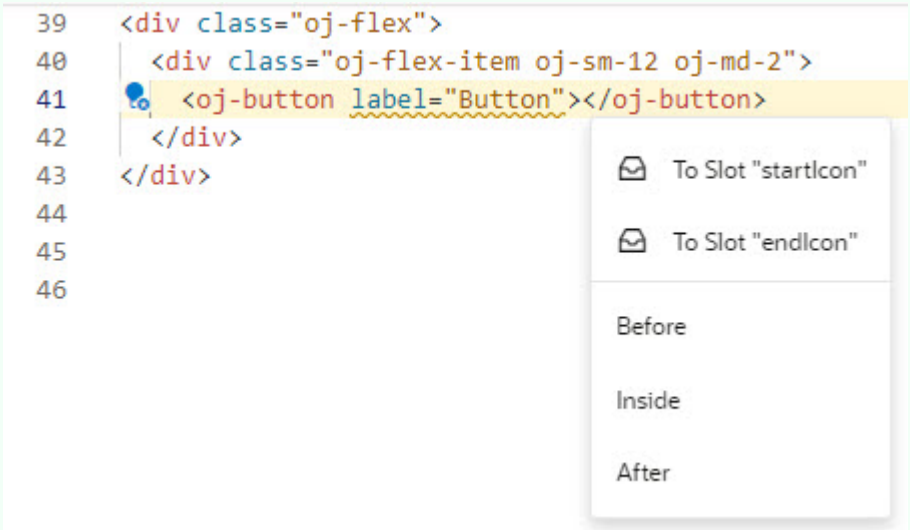
   For example, when you start typing `<oj` in the editor, the code completion window appears with a list of component tags that match the text you type:



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

Tip:

Add a component by using the code editor's right-click (or context) menu, where you can insert a component before, inside, or after an existing component. You can then choose from a list of components, identical to what's shown in the Components palette. If the component you're working with has multiple slots, selecting **Insert Component** in the context menu will give you the option of dropping the component into a particular slot, as shown here for a button component:

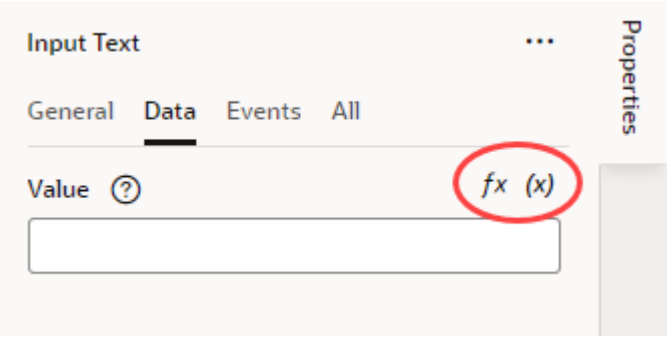


Once the component tag is added to the HTML code, you can define its properties using code completion. Alternatively, use the Properties pane in Design mode.

Bind a Component to Data

To display data in your user interface, you bind the component used in your application's pages to its data source. Every component has properties that you can bind to variables or expressions. These properties can affect the state of the component (for example, an input text field's value) or its rendering (for example, enabled or disabled).

When a component is bound to a variable, the data stored in the variable is displayed in the component. For example, you might bind an Input Text component to display a particular field's data from a REST endpoint, say, a user name. You can do this manually by clicking the **Select Variable** icon, available to you on the component's Value property in the Data tab. If you want to assign values to a component based on an expression, say, display a user's full name by combining the first and last names, you need to build an expression by clicking the **Expression Editor** icon, also available to you on the Value property in the Data tab.



For example, when you select a string-type `name` variable as the data source for an Input Text component, you should see the Value set to `{{$variables.name}}`, which is shorthand for saying your component is now bound to the page-level variable `name`. The double curly braces around the value `{{ }}` indicate that the variable can be updated or read by the component. In this case, the variable is automatically updated whenever the component's value changes. Conversely, the component is automatically refreshed whenever the variable changes. Double square brackets `[[ ]]` mean that the variable's data can only be read.

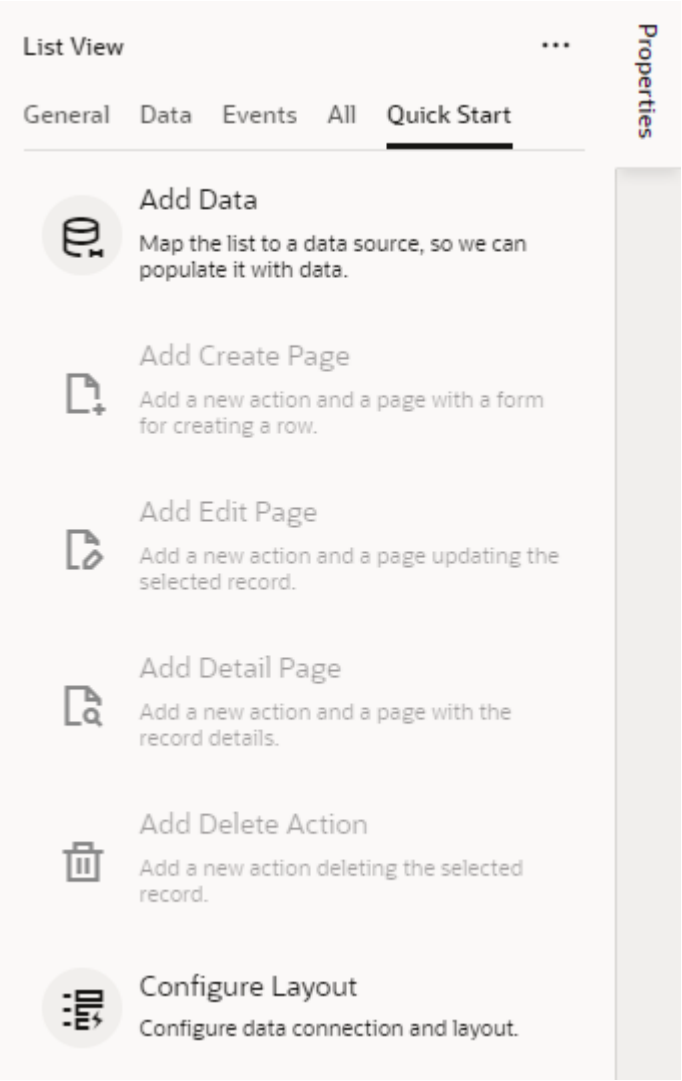
You can also use Quick Starts to bind a collection component, like a Table or List View, to a data source. See How Do Quick Starts Work?

How Do Quick Starts Work?

A *quick start*, as the name suggests, provides a way for you to build the basics of your application quickly. It's a wizard that walks you through complex processes, helping you do the tasks required to quickly build your application's functionality.

A variety of Quick Starts are available to help you perform some of the tasks required to add common application functions and behavior, for example, binding a table or a list in a page to a data source or adding a page to create new records. To add functionality, you need to create the artifacts that perform that function. And depending on how complex the behavior is, adding a functionality might involve creating several variables, types, action chains, and page events. If there's a quick start for your task, you can use it to quickly create many of the artifacts for you.

So how would you know if a quick start is available for what you want to do? Just look for the **Quick Starts** tab in a selected component's Properties pane when designing pages in the Page Designer. The Quick Starts tab will display a list of tasks that are typically used to add functionality or behavior to the selected component. These tasks are based on common tasks that developers typically do when creating applications. For example, here's a list of quick start tasks available for you when you are working with a List View component:



You'd start, for example, with the Add Data quick start to populate the list with data from your data source (typically, a GET MANY REST endpoint). Once that's done, other quick starts that let you generate other pages or set up functionality are enabled for you.

> **Tip:**
>
> If you cannot find the endpoint you want in the quick start or prefer to manually set up your endpoint, click the Manual Setup of Endpoint icon ( ) in the wizard to complete your endpoint configuration.

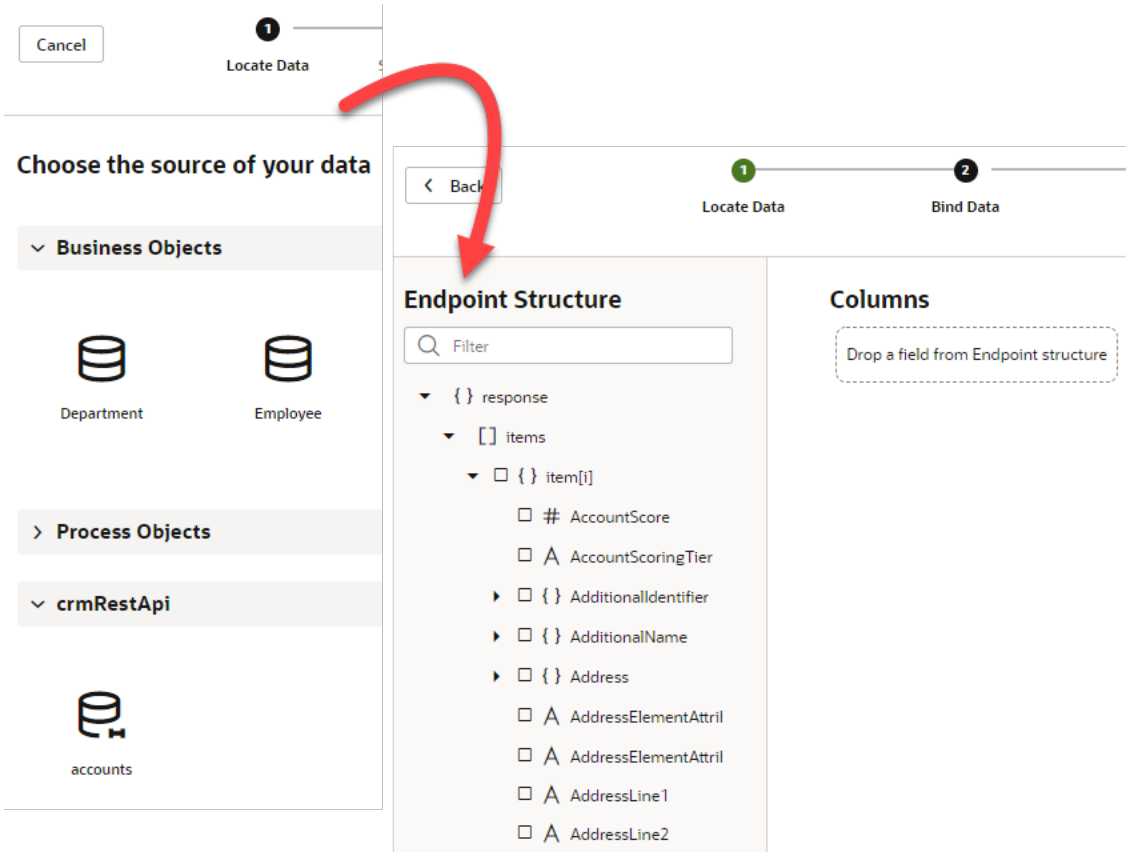
Control the Layout

It's possible to customize the fields you want to show as part of a component on a page. The steps to customize fields in a layout are the same, whether you're selecting endpoint attributes from a service connection or a business object in your application.

When you use collection components such as a table or list view to display data, you're binding the component to an endpoint, exposed by a service connection or a business object. The

easiest way to do this is by using quick starts—and if you're looking to customize the layout, you can use the Bind Data page of the wizard to select only the fields you want to display.

Here's an example of what you'll see in a quick start when you've got business objects and a service connection as the source of your data. When you bind data, you'll see the attributes of the source you select – this is where you can customize your collection component's layout:



Quick starts automatically create the variables and types you need for a particular functionality, but if you decide to add an endpoint attribute that you didn't include originally, you'll need to modify the type (using the **Edit From Endpoint** link next to the type on the **Types** tab), then modify the UI. In this case, it might be simpler to delete the table or list view and start over with the quick start.

Add Data to a Table or List

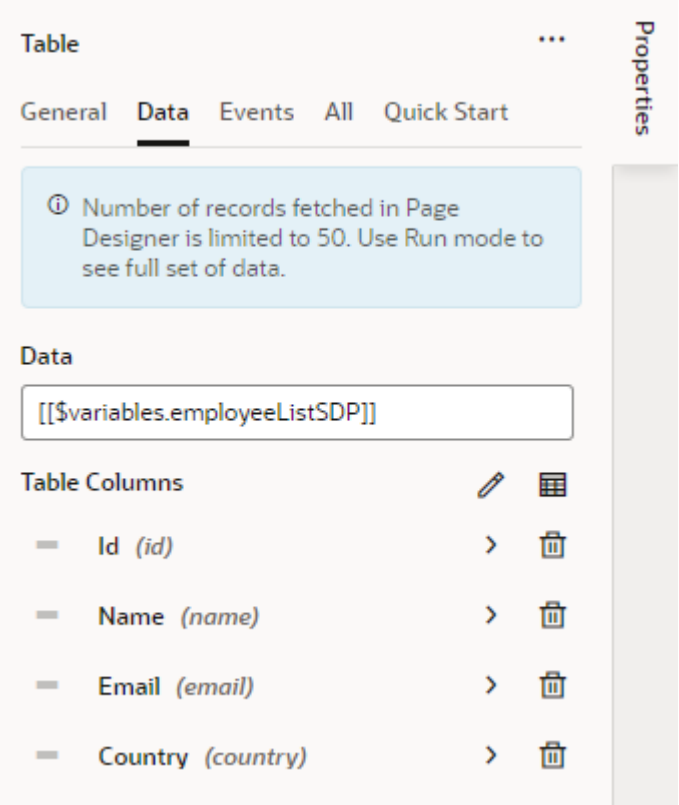
Use the Add Data Quick Start to populate a table or list in a page with data from a business object.

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.

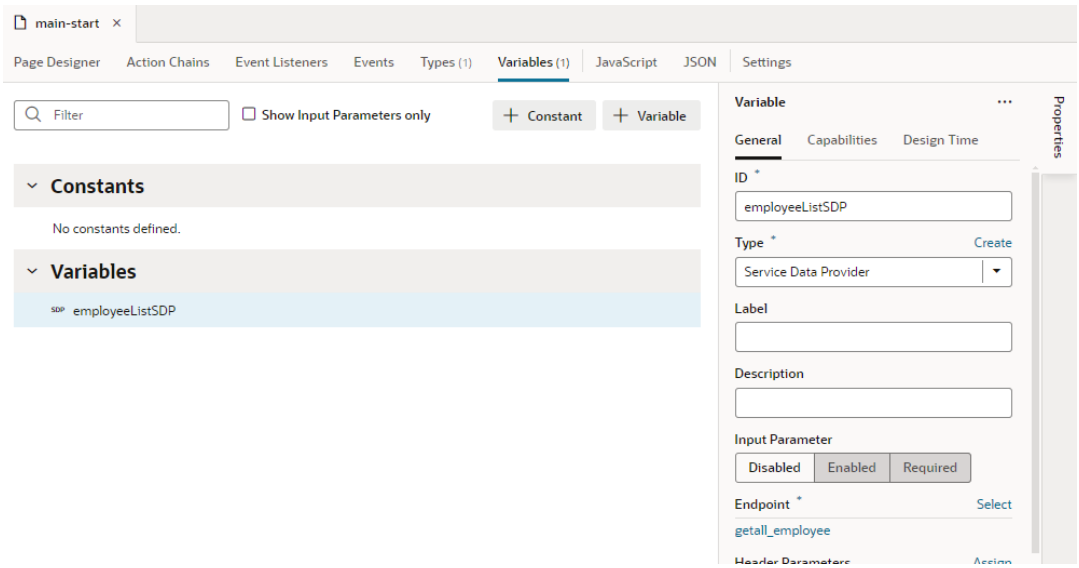
The Add Data Quick Start does the following for you.

- 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 `contactListSDP` 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 Properties pane:



You can see the details of the new variable if you open the Variables editor of the page. In the General tab of the Properties 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 Properties pane you can see that the type for the response is a new custom type named `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 Types 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 **Add Data** in the list of Quick Starts.
3. Select the data source that you want to bind to the collection. Click **Next**.
4. Select the fields you want to add from the Endpoint Structure. (You can also drag the fields you want from the Endpoint Structure on the left onto the Fields pane in the middle).

   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 `Id` field. Click **Next**.
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:

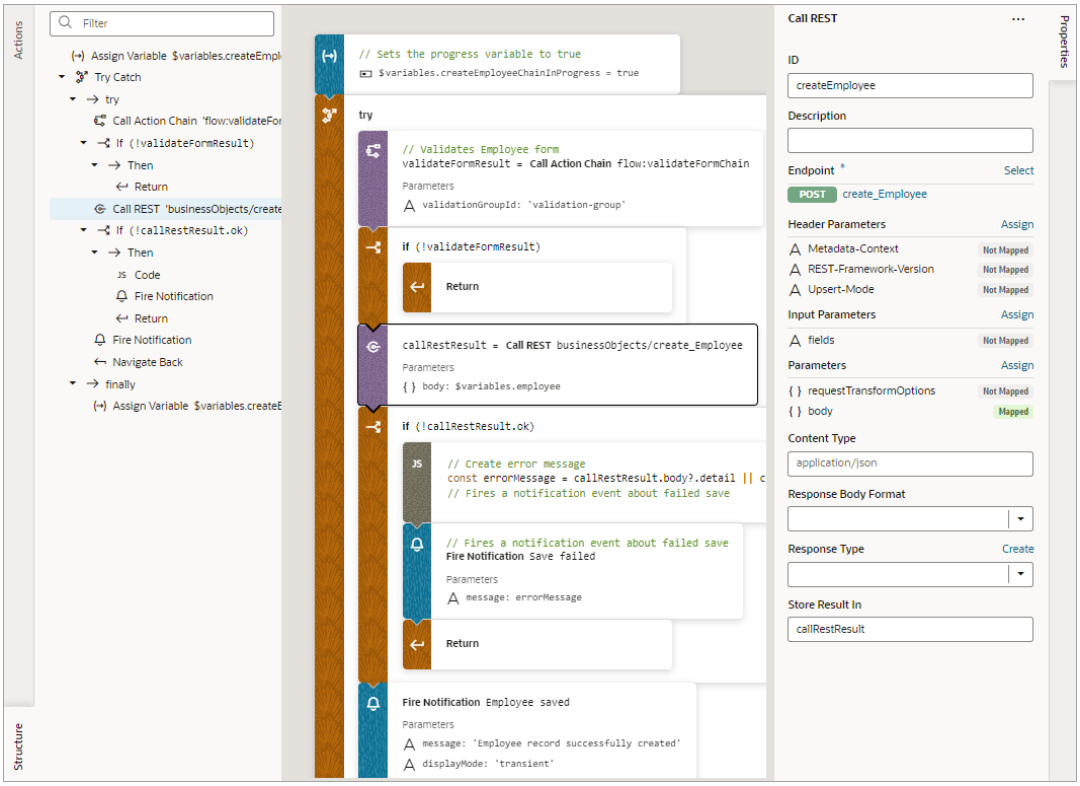
- Creates an action chain for navigating to a page.
- Adds a button component with an `ojAction` event that starts the action chain.

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 use a data source endpoint with a `POST` method. When your data source is a business object or a service with expected endpoints (such as `GET`, `POST`, `PATCH`, or `DELETE`), Visual Builder automatically selects the data source endpoint for you.

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 Properties pane.
2. Click **Add Create Page**.

3. If you land on the Select Endpoint step, select the data source where you want to create a row and click **Next**.
4. On the Page Detail step, select the fields you want to add from the Endpoint Structure. (You can also drag the fields you want from the Endpoint Structure on the left onto the Fields pane in the middle). 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.

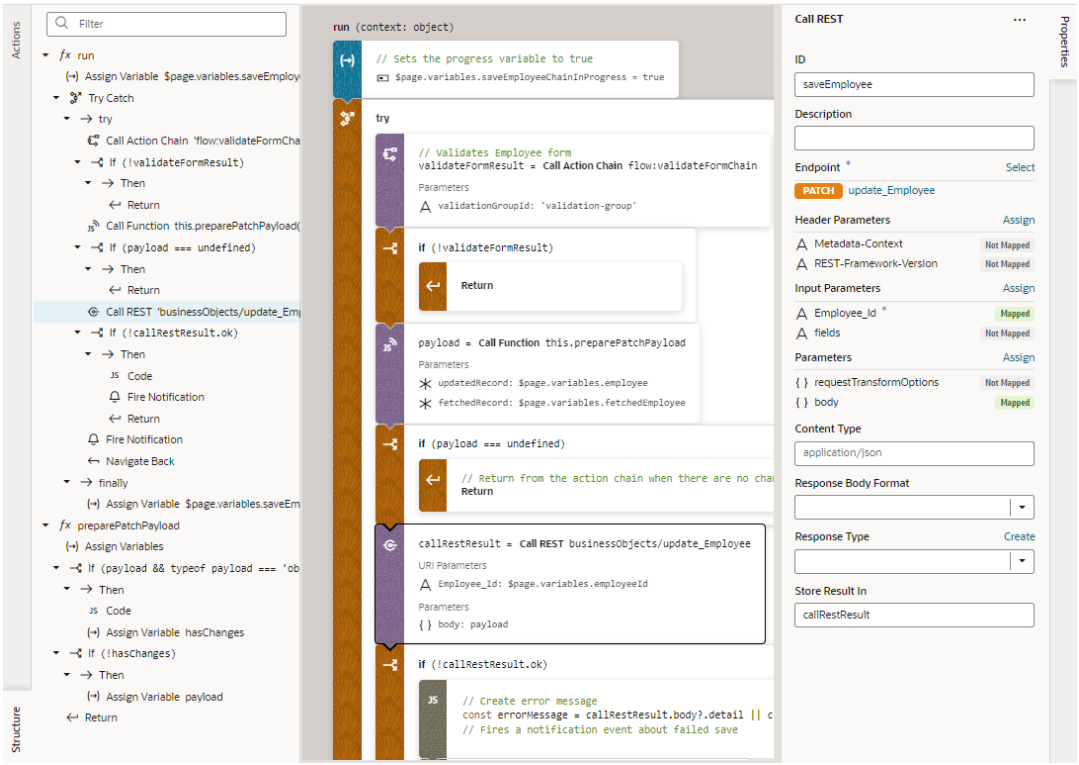
The Quick Start does the following in the page containing the collection:

- 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.

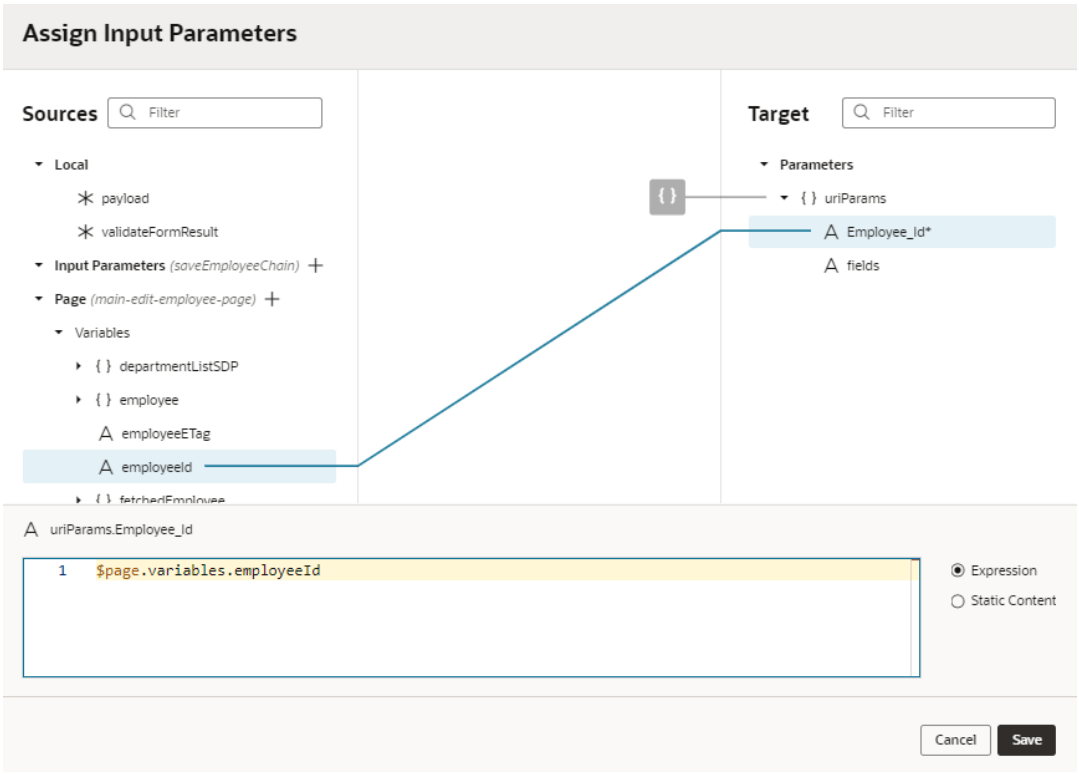
The Quick Start does the following in the Edit page.

- 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. When adding an edit button, you use 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. When your data source is a business object or a service with expected endpoints (such as `GET`, `POST`, `PATCH`, or `DELETE`), Visual Builder automatically selects the correct endpoints for you. You only need to 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 Properties pane, if not already open.
3. Click **Add Edit Page**.
4. If you land on the Select Read Endpoint step, select the data source containing the GET endpoint to get the data you want to edit. Click **Next**.
5. If you see the Select Update Endpoint step, select the data source containing the UPDATE endpoint to send a request to update the selected record.
6. On the Page Details step, select the fields you want to include in the Edit page from the Endpoint Structure. (You can also drag the fields from the Endpoint Structure on the left onto the Fields pane in the middle). The Edit page will include these fields in the form.

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. When adding a detail button, you use the endpoint with the `GET` method (Get One) to display data. When your data source is a business object or a service with expected endpoints (such as `GET`, `POST`, `PATCH`, or `DELETE`), Visual Builder automatically selects the correct endpoint for you. You only need to select the 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 Properties pane, if not already open.
3. Click **Add Detail Page**.
4. If you land on the Select Endpoint step, select the data source containing the Get One endpoint to get the data you want to display in the Detail page. Click **Next**.
5. On the Page Details step, select the fields you want to include in the Detail page from the Endpoint Structure. (You can also drag the fields you want from the Endpoint Structure on the left onto the Fields pane in the middle). The Detail page will include these fields in the form.
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.

Quick Starts for Dynamic Forms and Tables

When you add a dynamic form or table to a page, you have access to quick starts that help you configure data connections and layouts to dynamically display content based on rules that you've defined.

The quick starts for dynamic components walk you through similar steps as those for a standard component. For example, a dynamic table's Configure Layout quick start prompts you to select a data source and define parameters to query an endpoint, much like a standard table's Add Data quick start. Though instead of selecting fields to bind to the table collection component, you'll need to define a rule set to show fields in the dynamic table's layout based on conditions.

Here are the dynamic component quick starts that with the exception of rule sets, provide functionality similar to that of standard components:

Dynamic component	Associated Quick Start	Similar to:
Dynamic table	Configure Layout	Add Data
Dynamic form	Configure as Create Form	Add Create Page
	Configure as Edit Form	Add Edit Page
	Configure as Detail Form	Add Detail Page

When working with either set of quick starts, keep one key difference in mind: standard component quick starts *add* pages to your visual application (for example, the Add Edit Page quick start creates a separate page for editing an object's details); dynamic component quick starts *configure* an existing form or table on the current page.

Add an Image to a Page

To add an image to a page, you position an image component on the canvas and specify the path to the image in the Properties pane. You can select an existing image in the Image Gallery or import a new image from your local system. Use the tabs in the Properties pane 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 Properties pane. 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.

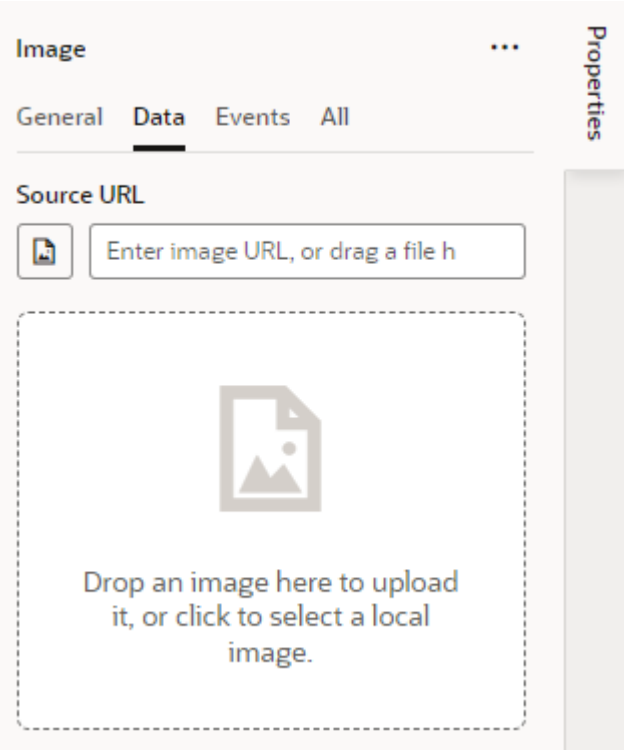
When you drag an image into the drop target area in the Data tab, the image is imported into the `images` folder for the flow, and the path to the image location is added to the Source URL field. For example, the path to an image stored in a flow's `images` folder will be similar to the following:`{{ $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. You can use Audits window to help you locate image paths in your app that might not be formed correctly.

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 Properties pane and specify the height, width and alt text for the image.
3. Open the component's **Data** tab in the Properties pane.

   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 Properties pane to view and edit all of the component's attributes.

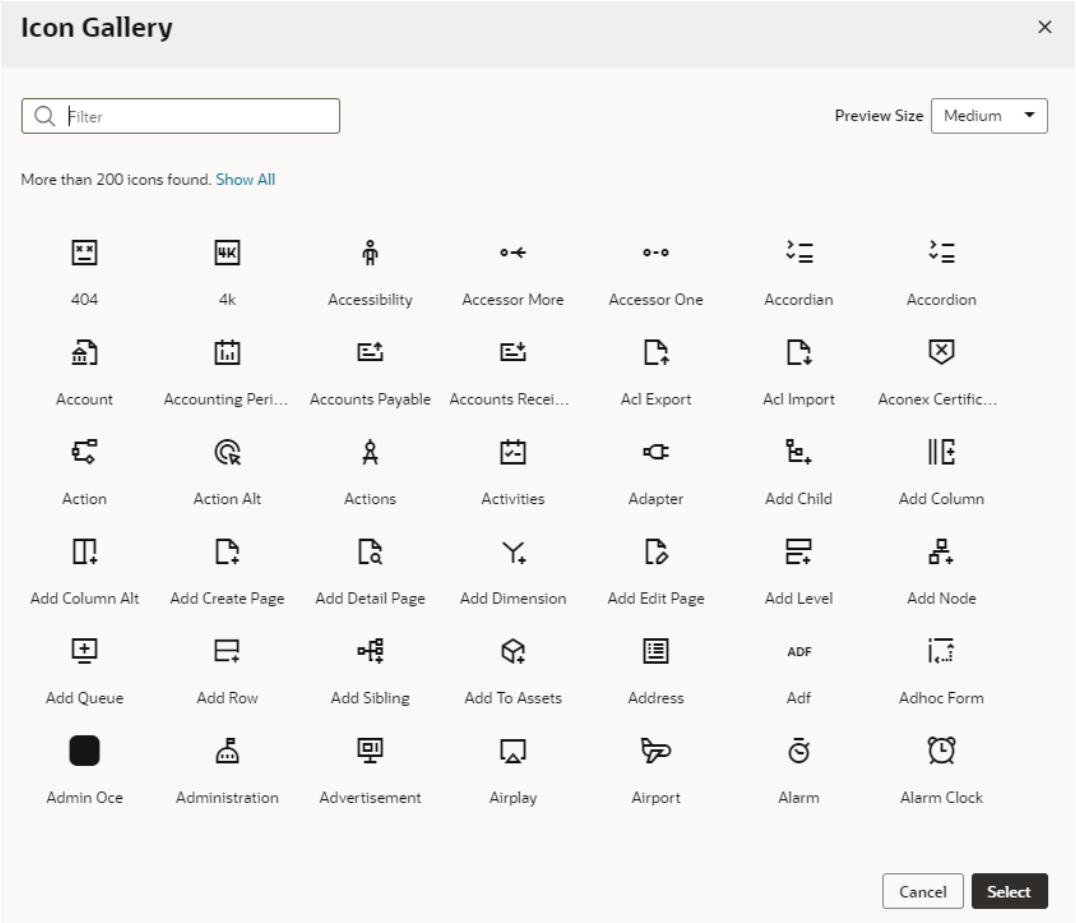
Add an Icon Component to a Page

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 Properties pane 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 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 ( ) in the **General** tab of the Properties pane.
3. Select the icon in the Icon Gallery window. Click **Select**.



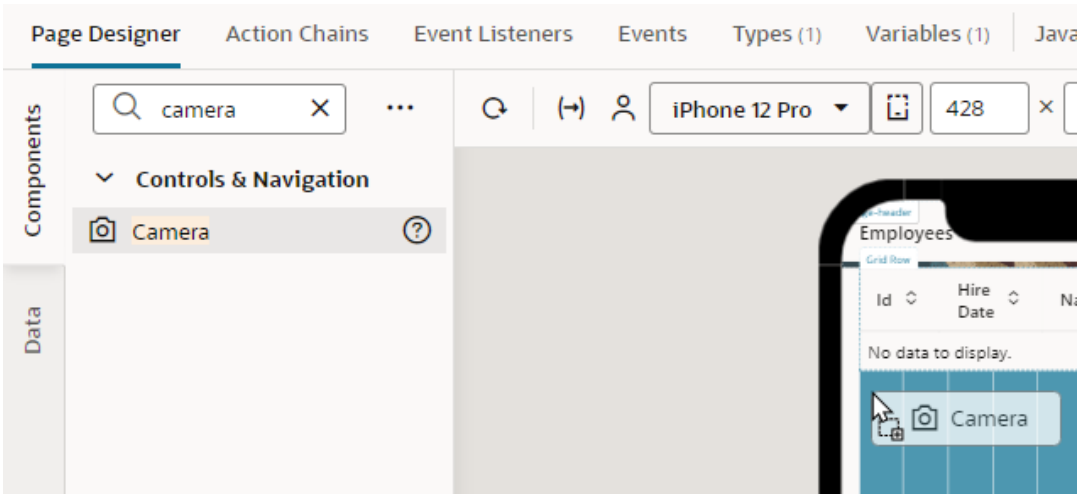
4. Specify properties or events in the Properties pane.

Add a Camera Component to a Page

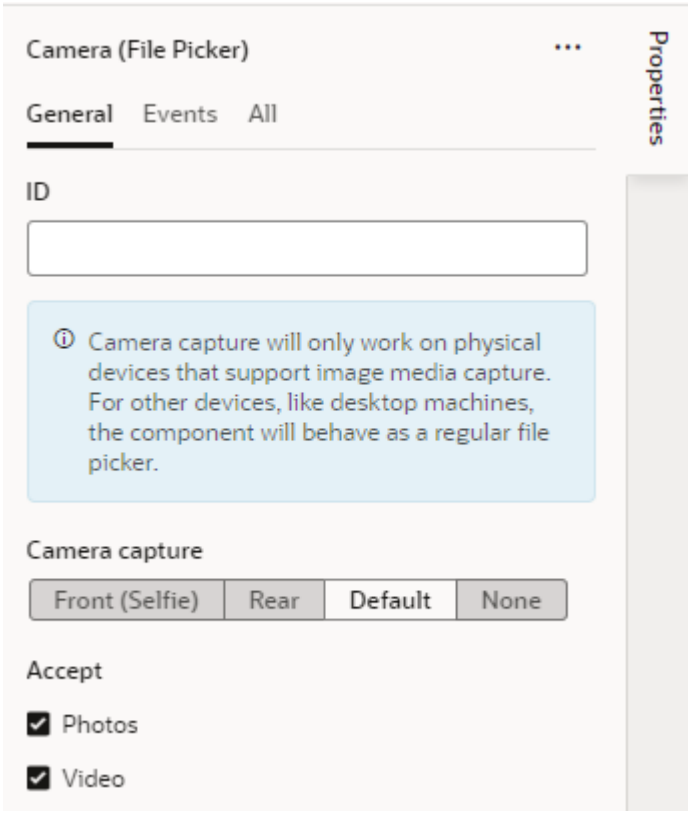
When you want your application to access the camera capabilities of the devices on which it is installed, you can add the camera component to your application's page for users to take a photo or a video from the device's camera.

The camera capture works only on physical devices that actually support media captures. On other devices (like desktops), it works as a regular file picker that lets users select files from the device's storage.

1. Open the page in the Page Designer and drag the camera component from the Components palette onto the page canvas:



2. In the **General** tab of the Properties pane, configure the component's properties for your use case:



   a. Use the **Camera capture** property to launch the camera on the user's device in a preferred camera mode:

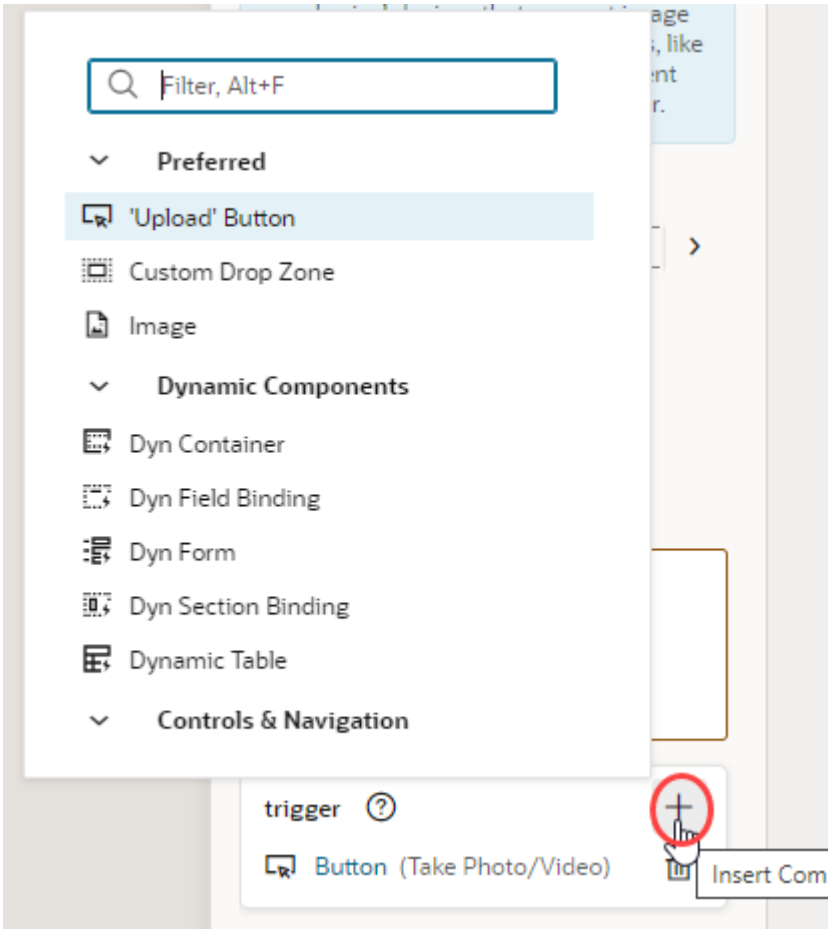
      - To let the user take a selfie photo or video, select **Front (Selfie)**. This option will directly launch the front-facing camera on the user's device.
      - To let the user take a photo or video of their environment, select **Rear**. This option will directly launch the rear-facing camera on the user's device.

- To enable the device's default behavior for the preferred camera mode, select **Default**. This option will directly launch the camera on the user's device in the default camera mode, which on iOS devices (for example) is to open the rear camera.
- To provide the user with choices, say, choose from the photo library or take a photo or video (instead of directly launching the camera), select **None**.

b. Set the **Accept** property to the file types you want to accept from the device: photos, videos, or both. By default, both **Photos** and **Videos** are selected to provide the user with options to take either a photo or a video once the camera is launched.

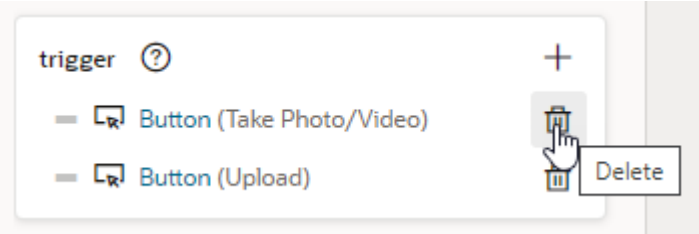
c. Optional: The camera component is preconfigured to use a **Take Photo/Video** button. This button acts both as a clickable control to activate the camera component and as a file drop zone for devices that support drag and drop. You can customize this button or replace it with other trigger components.

To replace the default **Take Photo/Video** trigger button, click **+** next to trigger and select an option from the Preferred components:



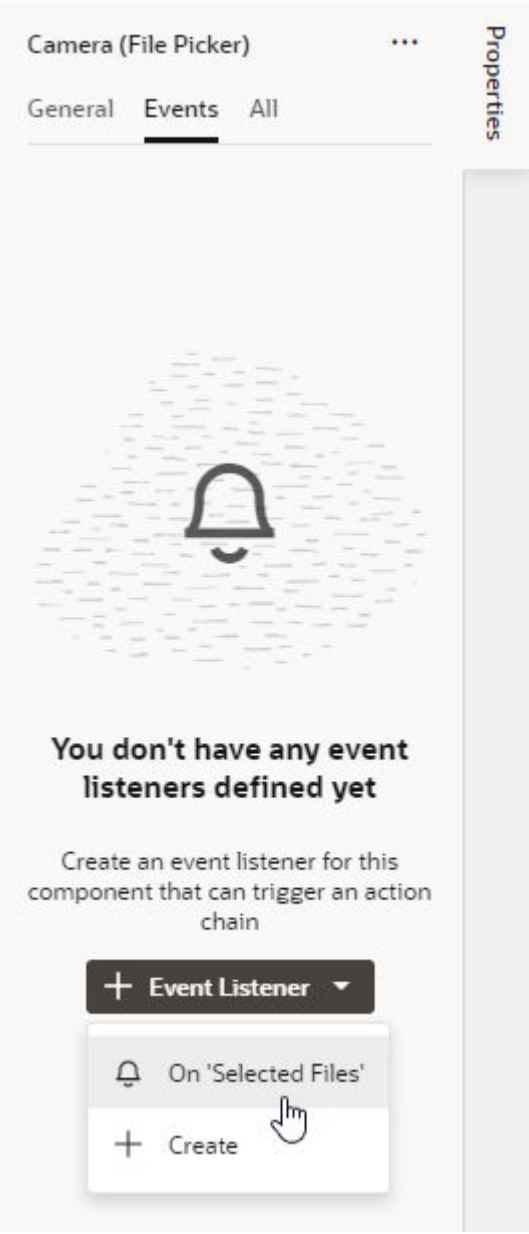
- To add an Upload button that users click to upload files, select **'Upload' Button**.
- To create a basic drop zone/trigger that can be further customized with other components, select **Custom Drop Zone**.
- To add a placeholder for an image that users click to upload images, select **Image**.

Remember to delete the original Take Photo/Video trigger button:



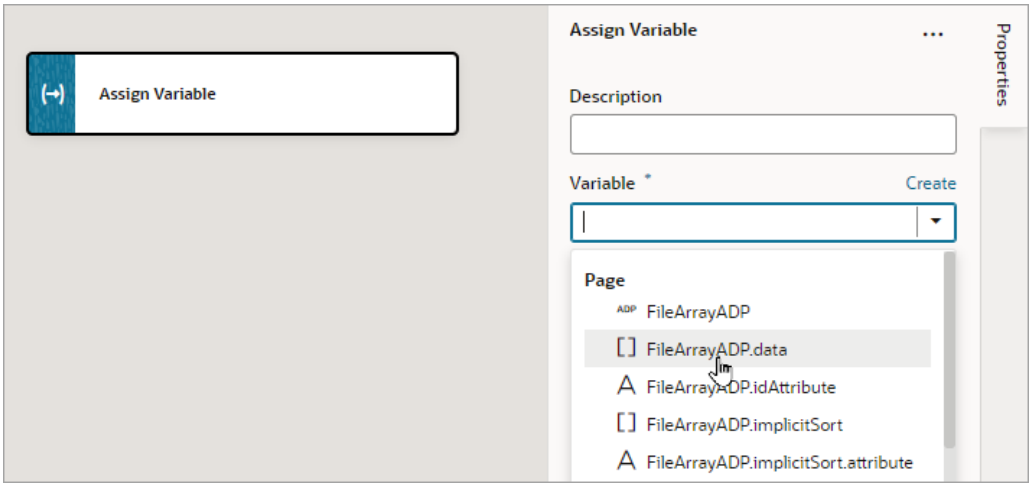
3. In the **Events** tab of the Properties pane, configure the camera component's `Selected Files` event to trigger an action chain when a user selects the camera component. Make sure to select the Camera (File Picker) component, not the button component within the camera component.

   a. Click **+ Event Listener**, and select **On 'Selected Files'**:



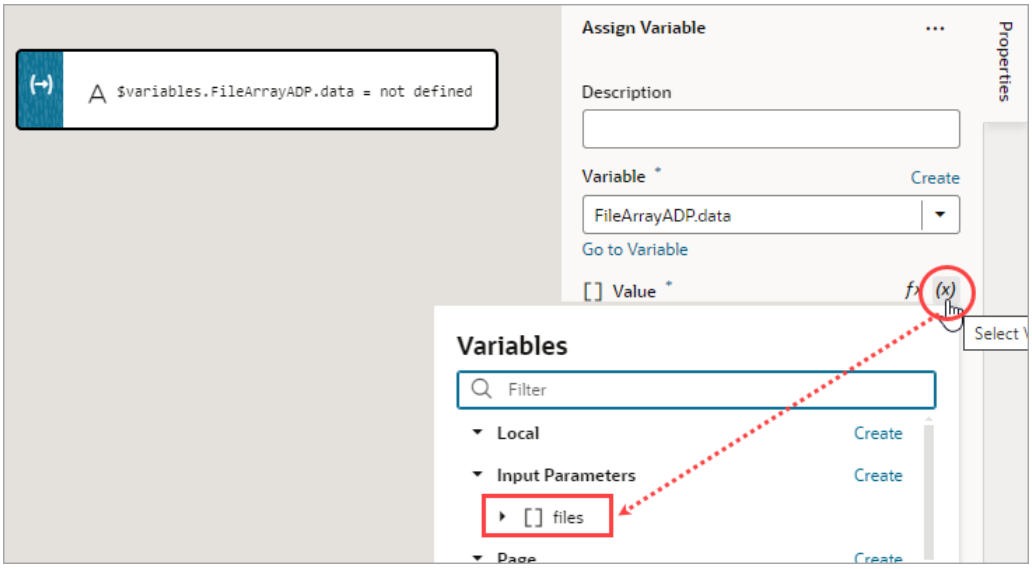
This creates the **CameraFilePickerSelectChain** action chain, which receives the files taken from the device. You can configure the action chain to upload the files to a server (via user-created JavaScript) or assign it to a variable, as we'll do next.

b. Add the **Assign Variable** action to the canvas. For the **Variable** property, select the variable to hold the data from the Camera component (for example, `FileArrayADP` of type `Array Data Provider`):

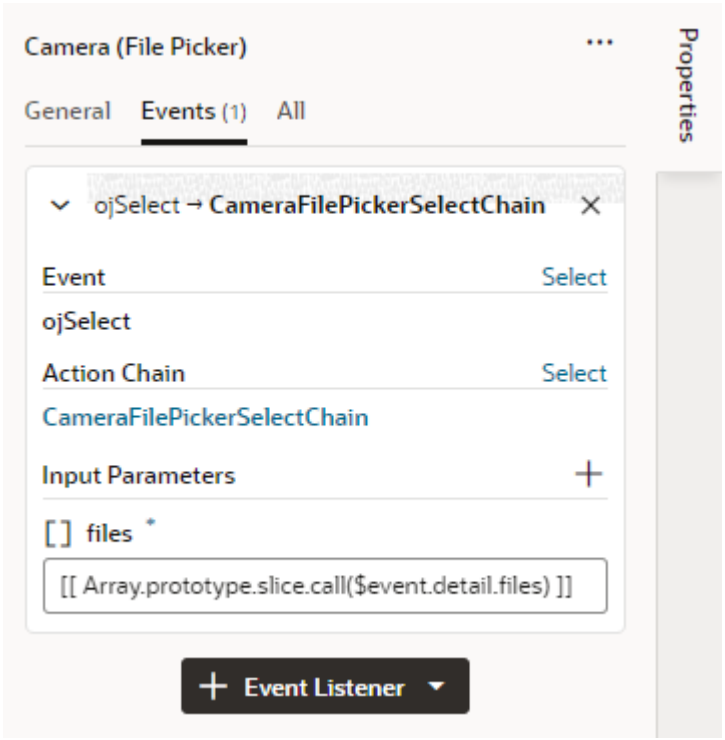


> **Note:**
>
> The items in the array are of type `File` (https://developer.mozilla.org/en-US/docs/Web/API/File). Each file is itself a Blob for the image or video, with properties such as name, size, and type. In Visual Builder, the files are passed to the action chain as an `Array` of type `Object`. When assigning to another variable, it is important that the type isn't changed to a custom type; otherwise, Blob data may be lost.

c. Hover over the **Value** property, click to open the variables picker, and choose the action chain's `files` input parameter, which contains the Camera components data:



d. Click the Page Designer tab to confirm that the array of files taken from the device is mapped to the **CameraFilePickerSelectChain** action chain:



4. Click **Preview** to test the camera functionality.

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 JavaScript and Add a Call 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.

Attributes	Description
`attribute`	Name of the attribute. Typically this is the name of the field that the filter will process.
`op`	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 Oracle JavaScript Extension Toolkit (JET) API Reference.
`value`	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).

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.

Filter Data by Filter Criteria

You filter the data displayed in a collection component, such as a list or table, by defining the `filterCriterion` property of the Service Data Provider (SDP) used 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.

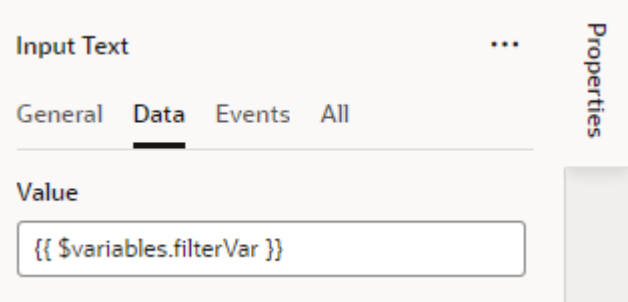
When you use the Add Data Quick Start to bind a collection component to a data source, you can use the Filter Builder in the Define Query step to filter data that you do not need to retrieve. For example, you might want to build a filter to only retrieve the records where the value of a column named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

In summary, to create a filter for your table or list, you'll need to:

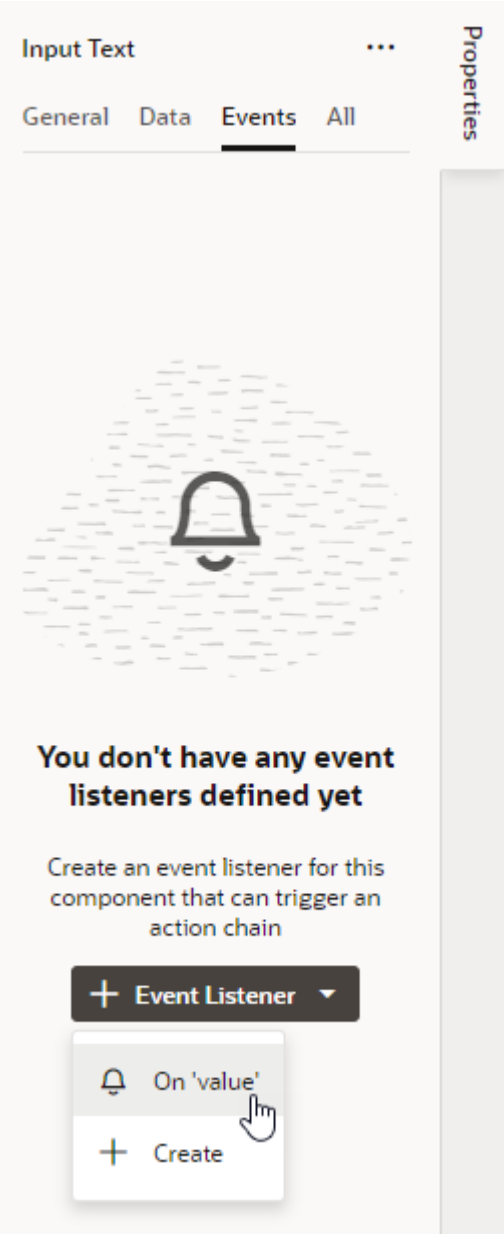
1. Create a page variable to hold the user's value for the filter criterion:
   - The filter criterion is a condition that each record must satisfy in order for it to be shown, for example, "`If name contains user-value`".
   - The user's value is the text that's used to evaluate the filter criterion for each record, to check if the record should be shown.
2. Add a UI component and an event to trigger an action chain that filters an SDP's data using the filter criterion.
3. Create an action chain to handle the filtering.

Here's how you can create a filter:

1. Create a page variable of string type (for example, `filterVar`) to store the user's value for the filter criterion.
   The value of the page variable can be predefined (for example, an input parameter), or you can bind it to a page component such as an Input Text (shown below) or Combobox component to allow users to enter text, or to select a value from a Combobox. In this example, we'll use an Input Text component to enter a value for the filter criterion and to trigger an action chain to filter the displayed data.
2. In the Page Designer, add an Input Text component to the page with the table or list. In the Properties pane, use the **Data** tab to bind the Input Text component to the variable that stores the user's value for the criterion:



3. Select the **Events** tab, click the **New Event** button, and select **on 'value'**:

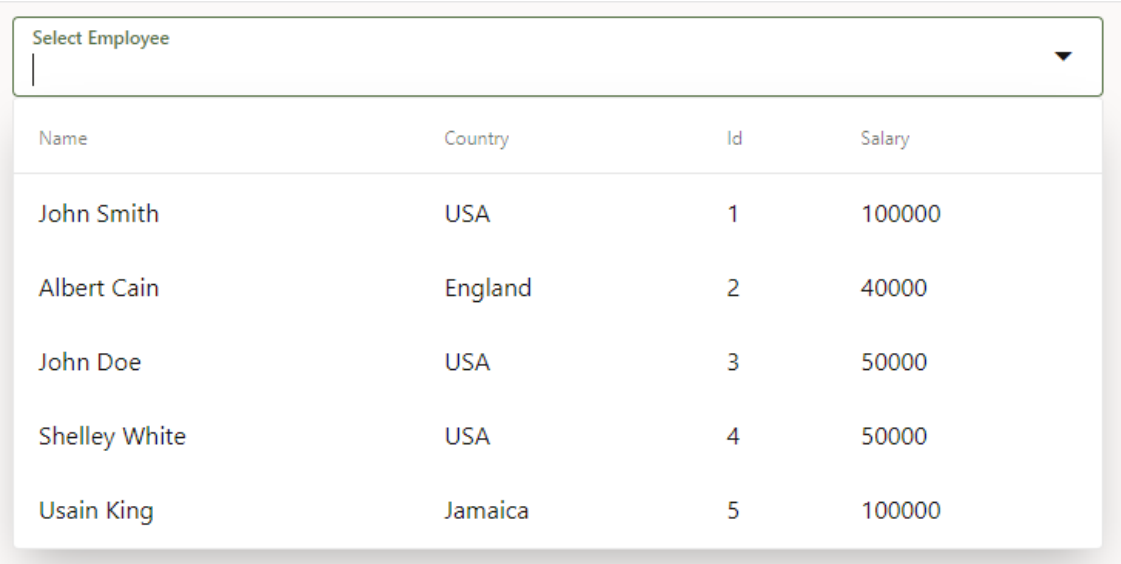


A new action chain is created and associated to the event. You're now taken to the Action Chain editor to implement the action chain for filtering. See Use Filter Builder to Create Filter Criteria for an SDP on how to create the action chain using an Add Variable action.

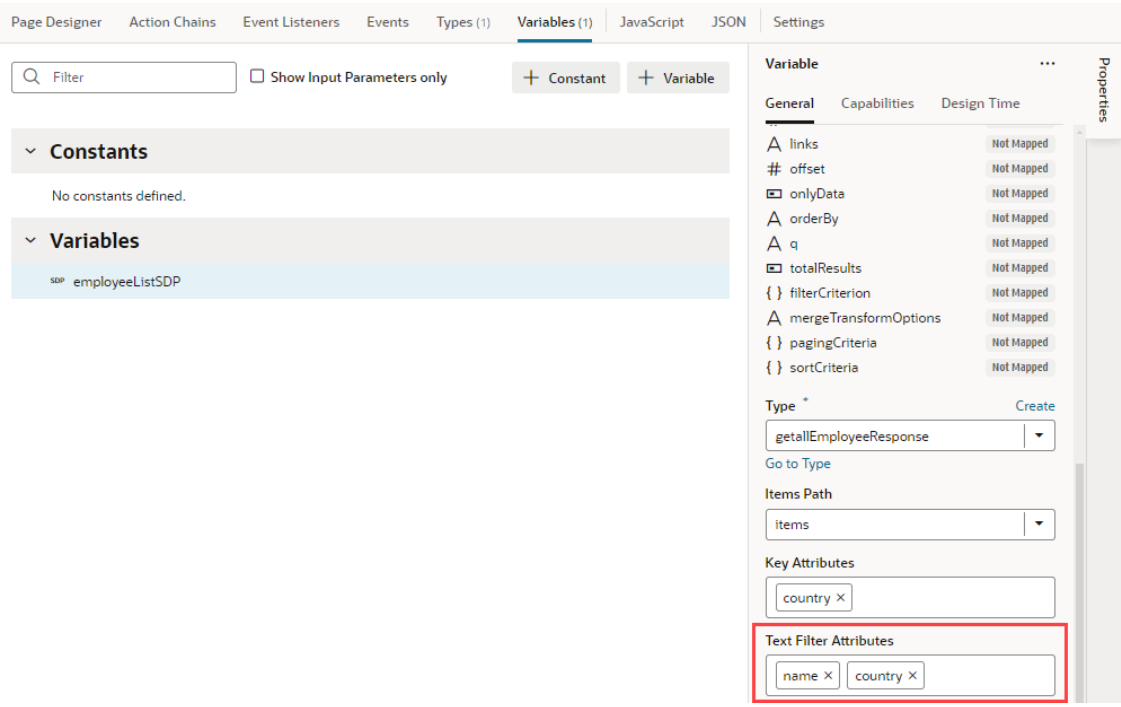
Filter Component Data by Text

When you want user-specified text to filter results shown in a list component like Select Single, you can configure the `TextFilterAttributes` property on the Service Data Provider (SDP) used to retrieve the list's data. The `TextFilterAttributes` property lets you specify data fields whose values you want to search for the text a user enters. Only values that match the user's text in each of those fields will be shown in the list.

Say you've set up a Select Single component to display employee data in a list and you'd like the user to filter the data as they enter some text string in the Select Employee field:

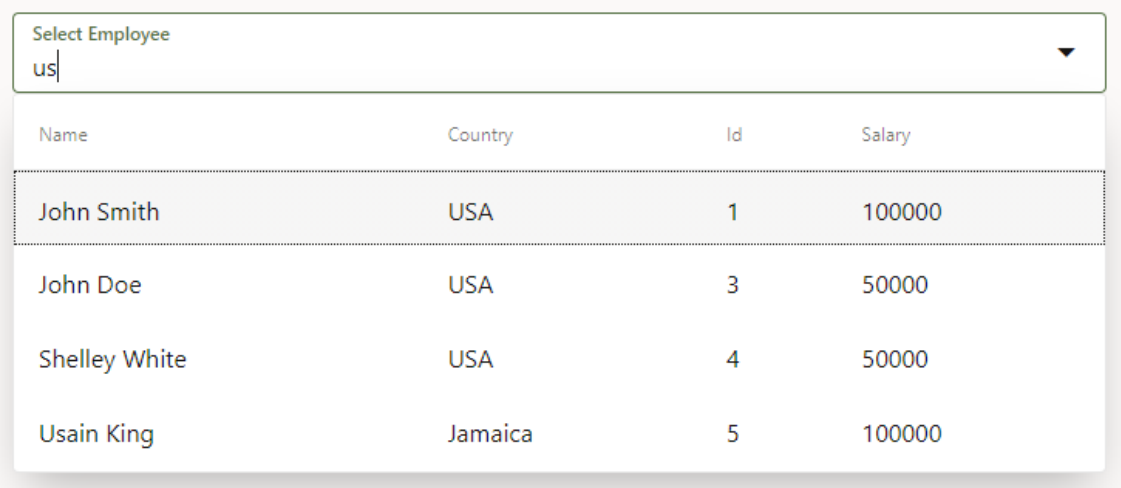


To achieve this, you need to configure the `TextFilterAttributes` property on the SDP variable used to bind the component to an endpoint. For example, your Select Single component might use the `employeeListSDP` variable to request and receive data from the `getall_Employees` endpoint. This variable is usually created for you when you use the Add Data Quick Start to bind a component to an endpoint and can be found on the page's **Variables** tab. Update the variable's `TextFilterAttributes` property to specify one or more data fields, for example, `name` and `country`:



A list that's based on an SDP doesn't hold a client-side array of data, so filtering is done by sending a "`q`" query string to the endpoint. The endpoint returns the matching values to the client, populating the SDP. So now if the user were to enter `us` in the Select Employee field, Visual Builder searches for values in the Name and Country fields that match `us`. The results

(as shown here) include three employees who belong to the Country USA as well as one employee whose name includes those letters:



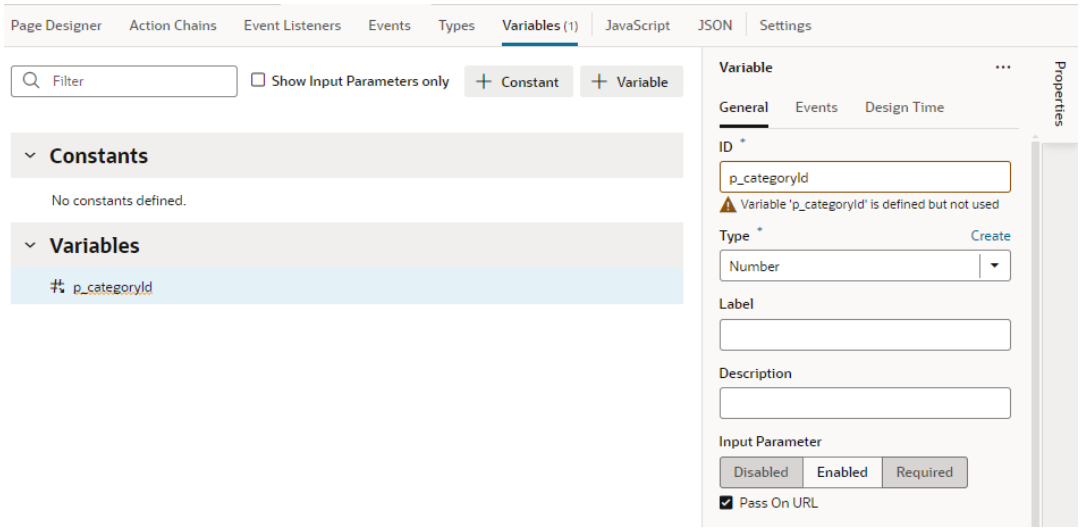
You can also include data fields that don't display to the user—as long as those fields are included in the SDP's definition. For example, if you selected `salary` and `phoneNumber` as the text filter attributes, a user who entered `50` would see three results because the text matches the Salary field of John Doe and Shelley White as well as the phone number of Albert Cain, although Phone Number isn't displayed as a column in the list.

Filter Component Data by URL

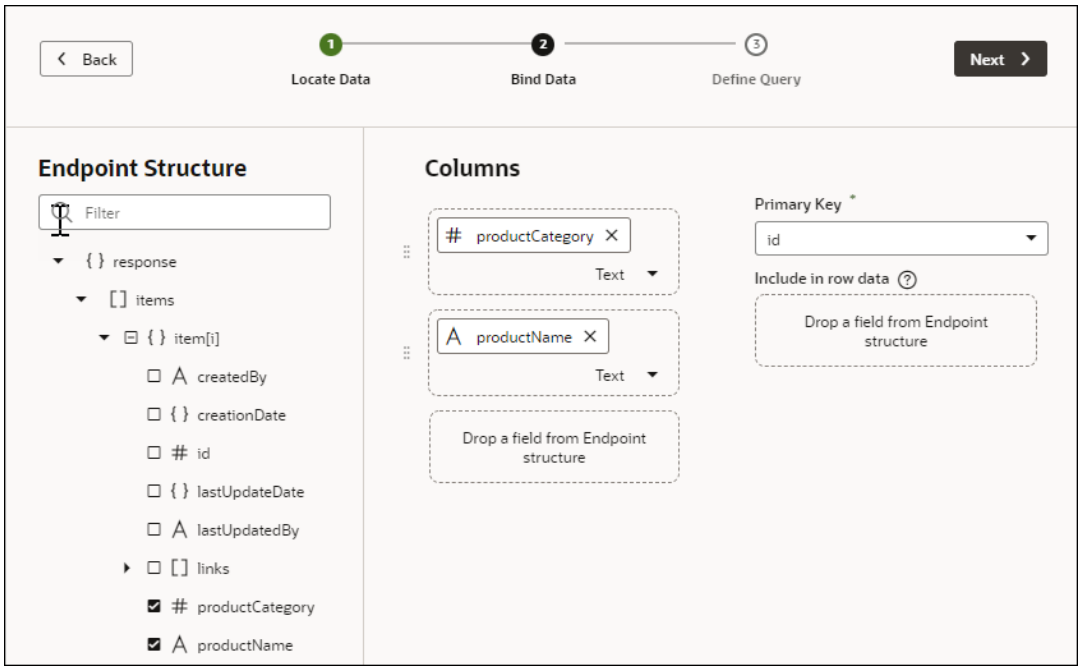
You can specify filter criteria for a collection component, such as a list or table, and append the parameter to the application page URL to apply the filter. This makes it easy to share the page URL with the filter already applied.

To enable the search criteria, set the `filterCriterion` attribute. In this example, we bind the value of the variable `p_categoryId` to the `productCategory` field of the business object.
To assign a page variable as a filter criterion for a table:

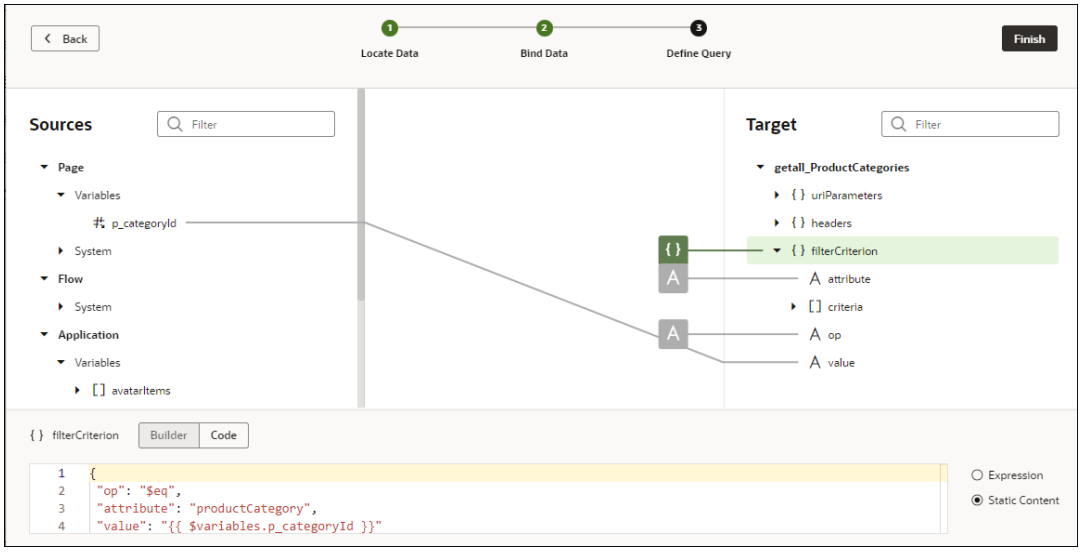
1. Click the **Variables** tab for your page.
2. Click **+ Variable**, then update the **ID** as needed and select the **Type**.
3. Set the Input Parameter to **Enabled** or **Required** and select **Pass On URL**.



4. Open the Page Designer and drag a Table component onto your page.
5. Select the table, then click the **Add Data** quick start to populate the table.
6. On the **Add Data** page, specify the business object you want to use and click **Next**.
7. On the **Bind Data** page, bind the fields you want to display and click **Next**.

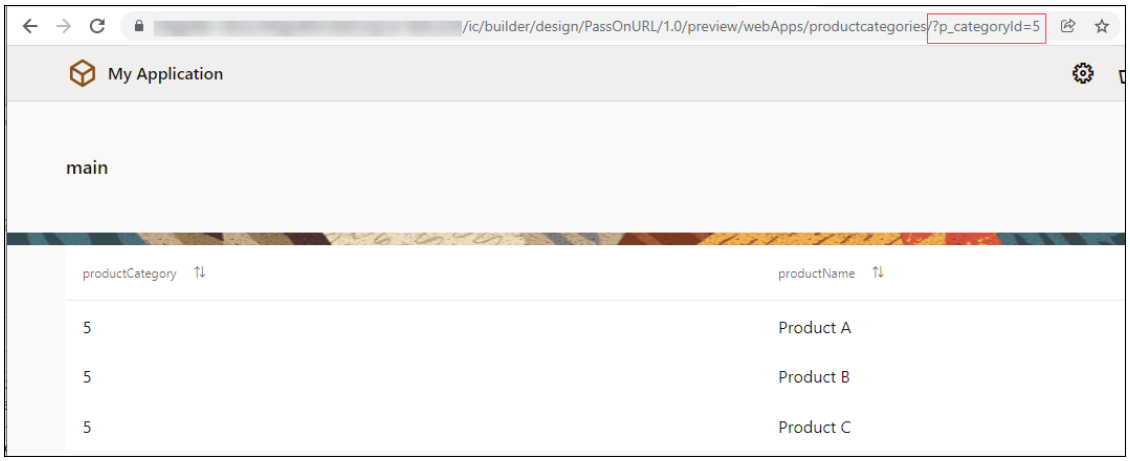


8. On the **Define Query** page of the quick start, expand the **filterCriterion** builder icon on the **Target** pane and build your filter by specifying values for **attribute**, **op**, and **value** individually.



9. Click **Finish**.

To filter the table, add `?paramName=Value` to the end of the app URL.

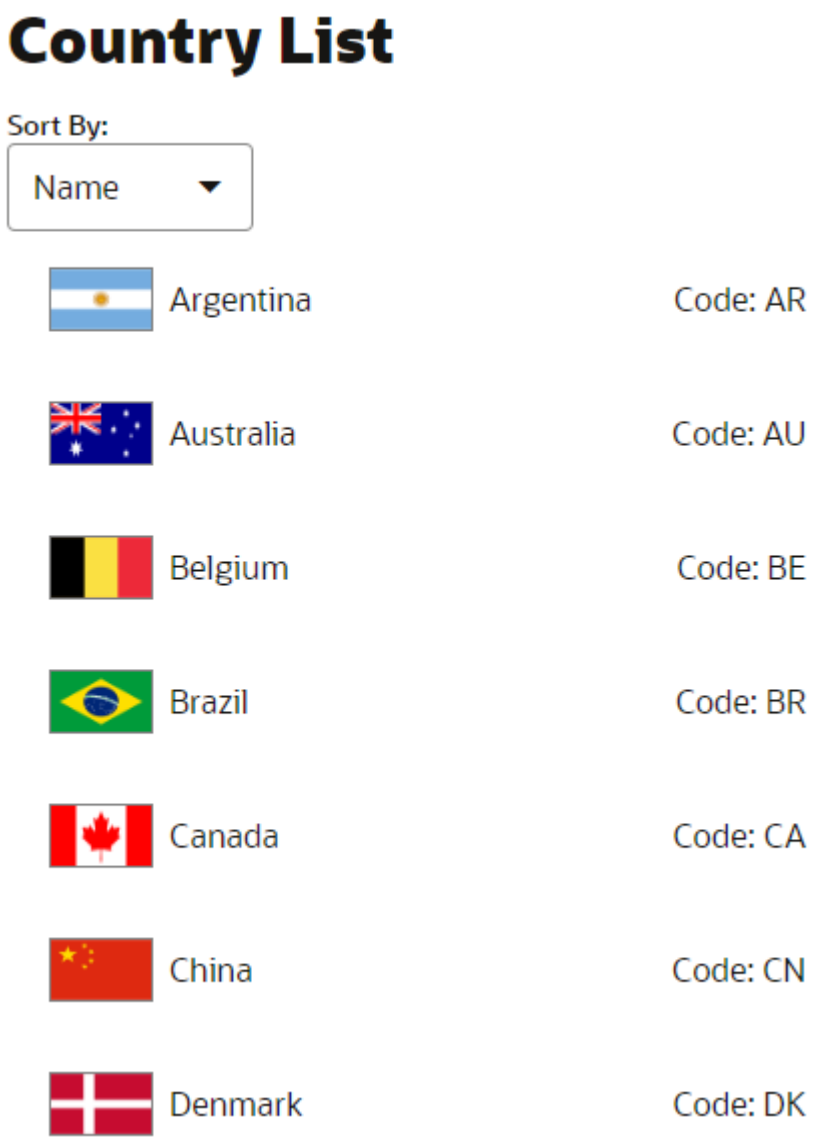


Sort Data Displayed in a Component

Sorting allows you to arrange your data in ascending or descending order. To sort specific fields in your data set, you can use the `sortCriteria` parameter in the Service Data Provider (SDP) used to retrieve the data.

> **Note:**
>
> For sorting to work, the REST service backing the SDP must support sorting. If your SDP connects to endpoints from a Visual Builder business object or an Oracle Cloud Applications service, sort criteria can work out-of-the-box. If your SDP connects to endpoints from an external service, you need to provide a transform function to translate your sort criteria to the format expected by the service.

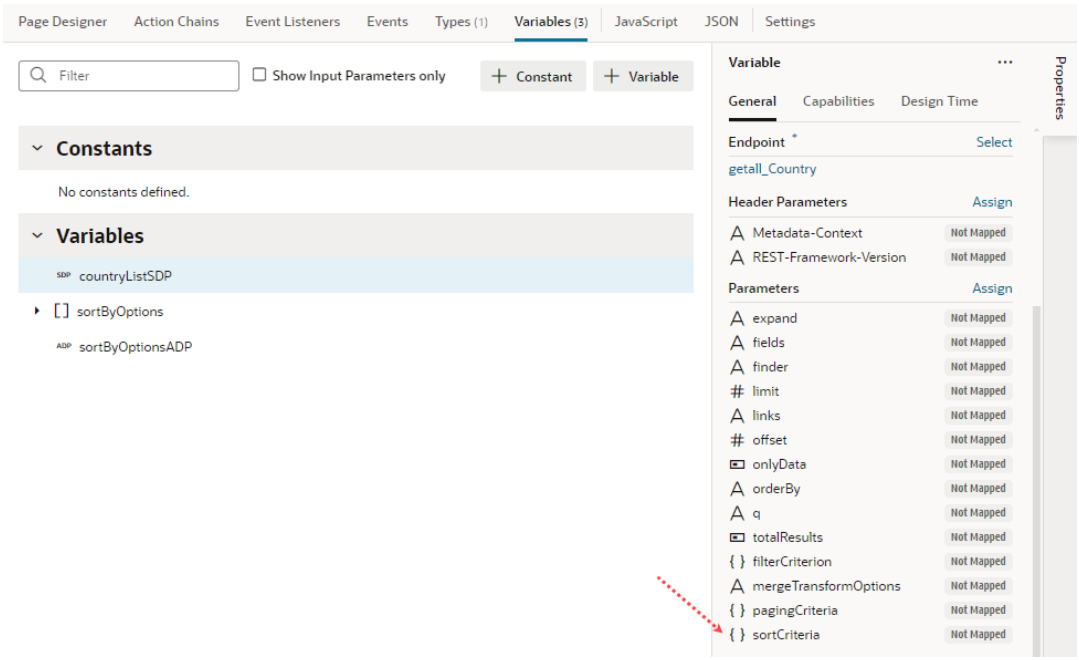
Let's say you have a List View component to display country information and a Select Single component to maintain sort criteria that looks similar to this:



(For details on setting this up, see the sorting recipe for a List View component in the Visual Builder cookbook.)

Assuming your Select Single component is wired to the correct variables, here's how you configure the sort behavior:

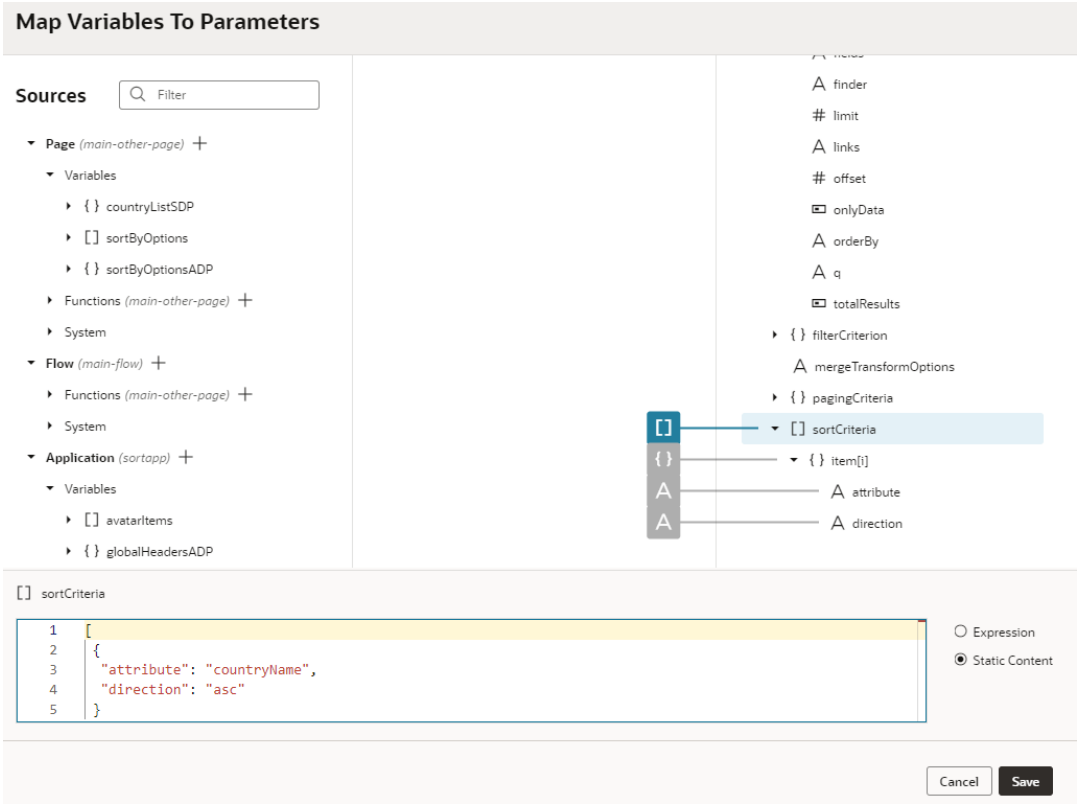
1. In the **Variables** tab, locate the SDP variable underlying the List View component, for example, `countryListSDP`.
2. In the variable's Properties pane, select **sortCriteria** under Parameters:



3. On the Target pane of the Map Variables to Parameters dialog, expand **sortCriteria** and **item[i]**:

   a. Select **attribute**, then in the text area, enter the name of the field for which you want data to be sorted; for example, `countryName`.

   b. Select **direction**, then specify the sort direction, either `asc` for ascending or `dsc` for descending. If you don't specify the direction, the default value (`asc`) is used.



You can add several sort criteria in the array to define sorting by multiple columns, for example:

```
[
 {
  "attribute": "countryName",
  "direction": "asc"
 },
 {
  "attribute": "countryCode",
  "direction": "asc"
 }
]
```

4. Click **Save**.

Sorting in tables works a little differently. See Sort Data in Table Columns.

Use Conditions to Show or Hide Components

You can use an `oj-bind-if` component to conditionally show or hide UI components in your visual application. Use `oj-bind-if` to surround other components and set conditions to determine whether the components should be displayed on a page.

Let's say you have a form for users to submit expense reports with fields like Country and Amount (in US currency). When users from countries other than the United States submit expenses, we want to show additional fields like Exchange Rate and Amount in USD. In other words, we want these fields to show only when the country selected is *not* United States. You can do this by surrounding these fields in an `oj-bind-if` component (available as **If** in the Components palette).

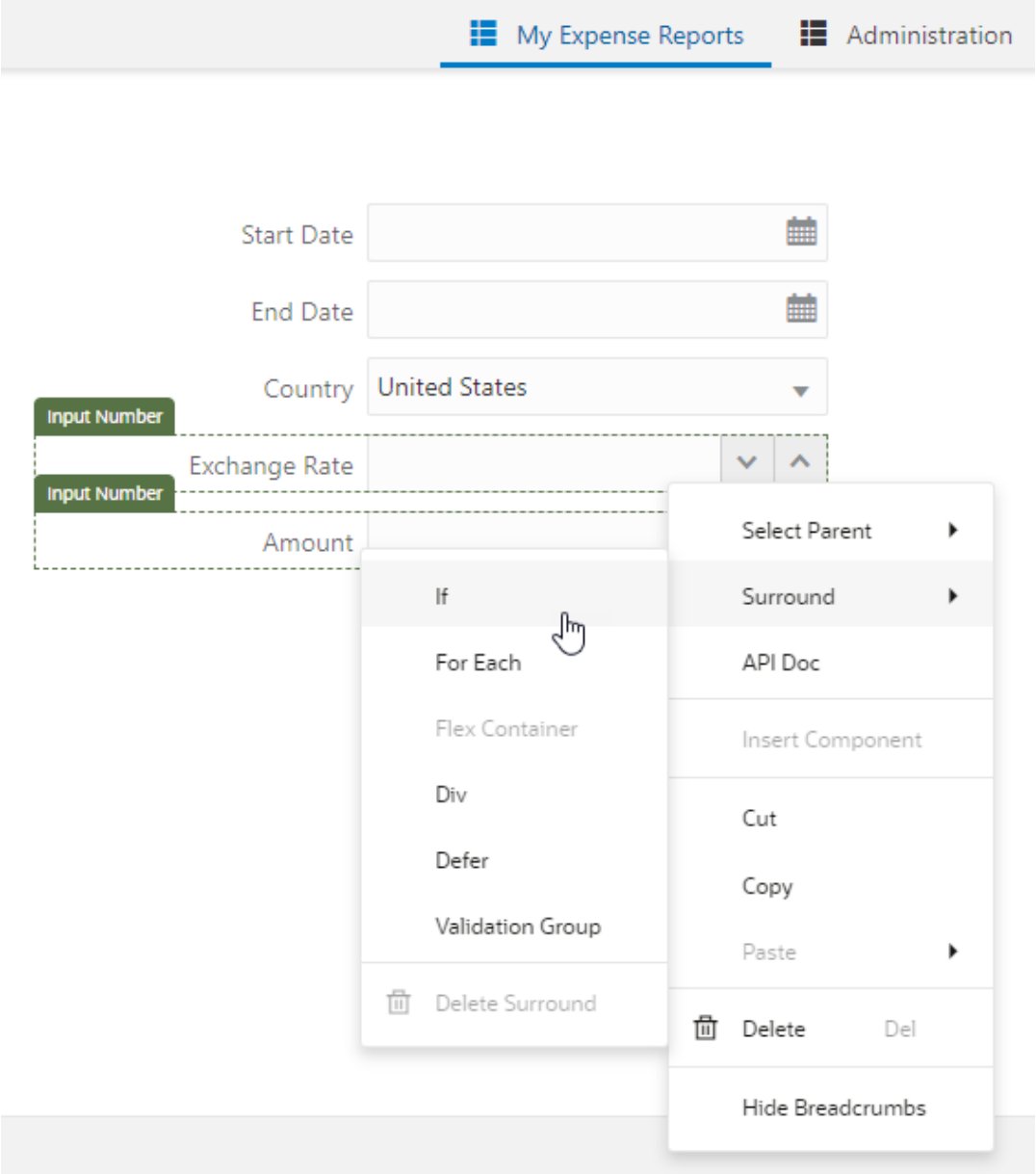
When you add an `oj-bind-if` component, you also set the conditions under which the component should be displayed by entering an expression in its Test property in the Properties pane. For example, you can use an expression that evaluates if the value of a page variable does not equal a predefined value. The surrounded content is displayed if the values are not equal (the expression is true), and hidden if the values are equal.

In our example, we'll build an expression using variables to show the surrounded content when the value selected for Country is something other than United States and to hide these fields when the selected Country is United States.

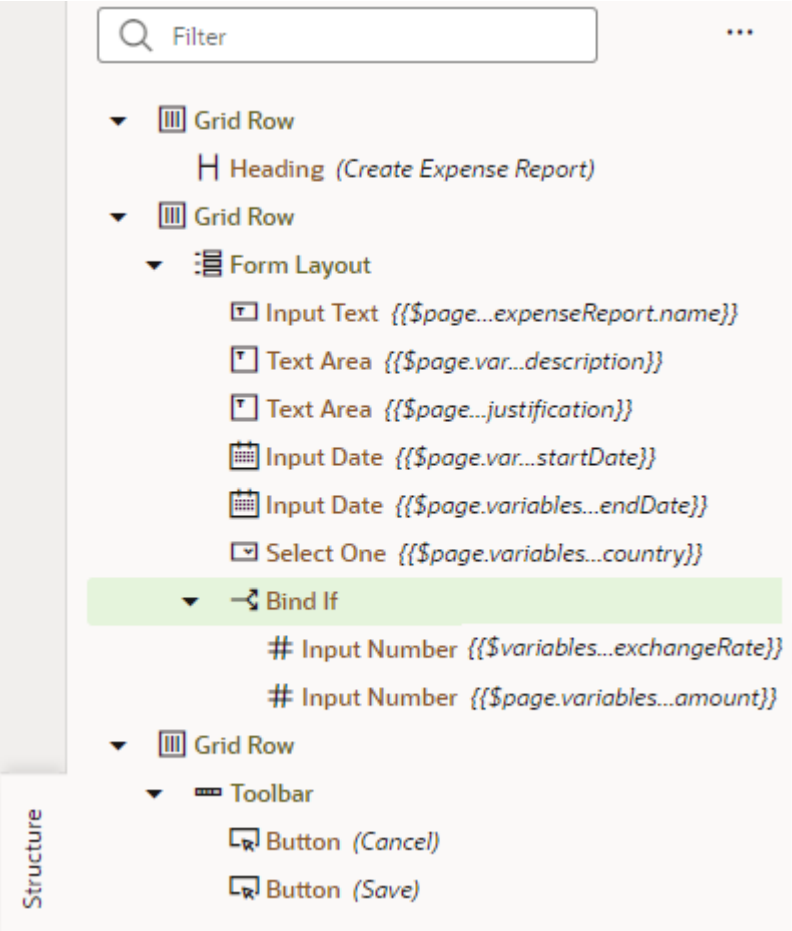
You can also use `oj-bind-if` to dynamically control what components a user sees based on the user's role, for example, to hide buttons or navigational elements, by using `$application.user.roles.`*`role_name`* in the expression. You can set restrictions on entire pages, or just on certain components 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, locate the component that you want to control dynamically.
2. Right-click the component on the canvas or in the Structure view and select **Surround** > **If** in the pop-up menu. In our example, we'll select the two `oj-input-number` components for the Exchange Rate and Amount in USD fields that we want to control dynamically.

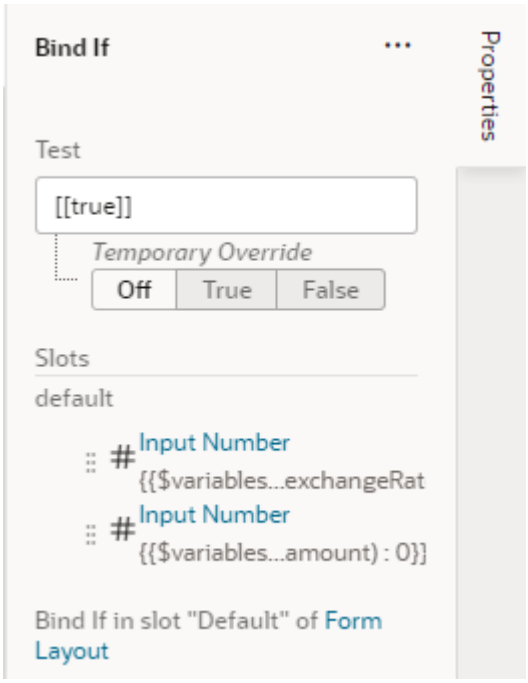


In the Structure view, you'll see the components that are surrounded by a **Bind If** component. (In Code mode, you'll see `oj-bind-if`.)



3. Select the **Bind If** component on the canvas or in the Structure view and open the Properties pane.

   You'll see the Test property which you use to set the condition. The default expression is `[[True]]`.



4. Enter the condition that controls the component's visibility. You can enter the condition in the Test field, or use the Expression Editor to build an expression using available variables, as shown here:

In our example where the `country` variable is of type number, the above expression is used to hide the Exchange Rate and Amount in USD fields when the Country field's value is United States. If you want those fields to be hidden even when the Country field is empty, you can extend your expression as follows:

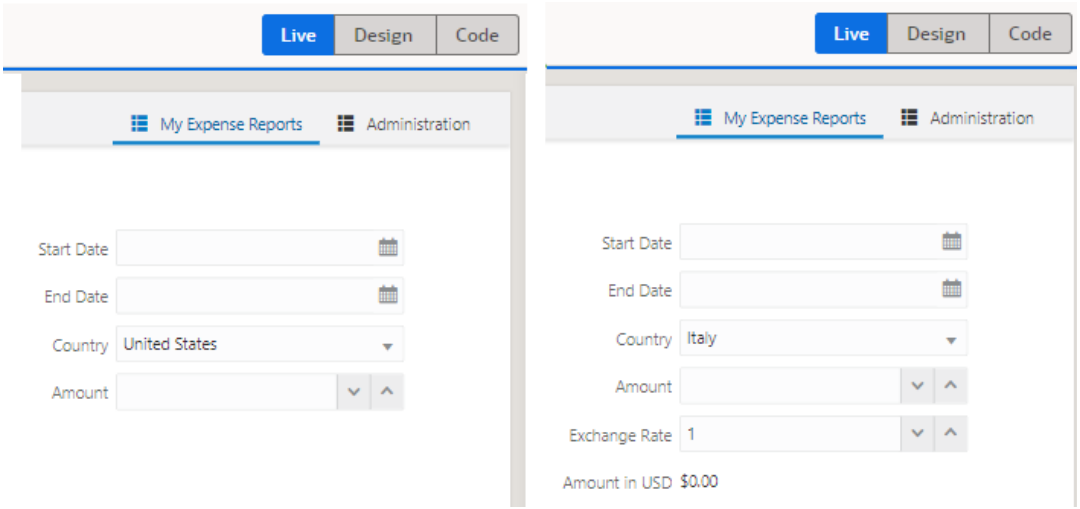
```
[[ $variables.expenseReport.country !== 1 &&
    $variables.expenseReport.country]]
```

If the variable type is not a number, remember to use quotation marks (' ') around the value in the Expression Editor.

5. Optional: In the Properties pane, use the **Temporary Override** property 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 set Temporary Override to True so the content is visible on the canvas, or select Off to temporarily disable the test. This setting is temporary and will revert to Off when you reload the page.

   You can set the Temporary Override in the Properties pane or in the component's popup menu in the Structure view or on the canvas.

6. Test your application in Live mode. Here's what our example form looks like with dynamic UI controls enabled:



Use Converters to Change Date and Time Formats and Time Zones

You can use *converters* to change how date-time, date, and time components display their data, or to convert one time zone to another.

To convert time zones, times must be stored as strings, using the ISO 8601 time format.

> **Note:**
>
> The ISO "Zulu time" format (for example, 2022-01-12T04:00:00Z) is recommended for storing date and time values, as they remain consistent regardless of time zones.

There are two ways to convert date and time formats, and to convert one time zone to another:

- Use the component's built-in converter
- Assign a converter function to the component

While you can use either method for the Input Date Time, Input Date, and Input Time components, you must assign a converter function for Date Time Picker and Inline Date Picker components.

To create your own converter function:

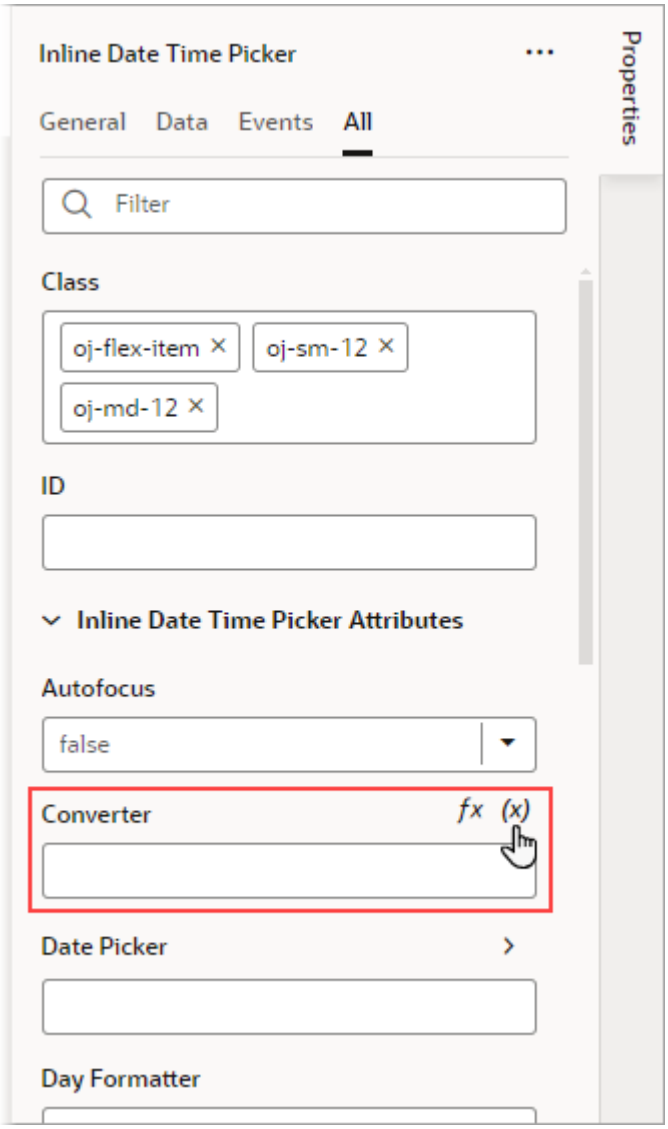
1. Open the page's JavaScript editor.

2. Include the Oracle Jet IntlDateTimeConverter API.
3. Create your converter function with a date input parameter (date is automatically passed in) and by using the IntlDateTimeConverter API, as shown here:

```
define(["ojs/ojconverter-datetime"], function (datetimeConverter) {
  class PageModule {
    constructor() {}

    formatDateTimeNewYork(date) {
      return new datetimeConverter.IntlDateTimeConverter({
        "dateFormat":"short",
        "formatType":"datetime",
        "timeFormat":"full",
        "timeZone":"America/New_York"});
    }
  }

  return PageModule;
});
```

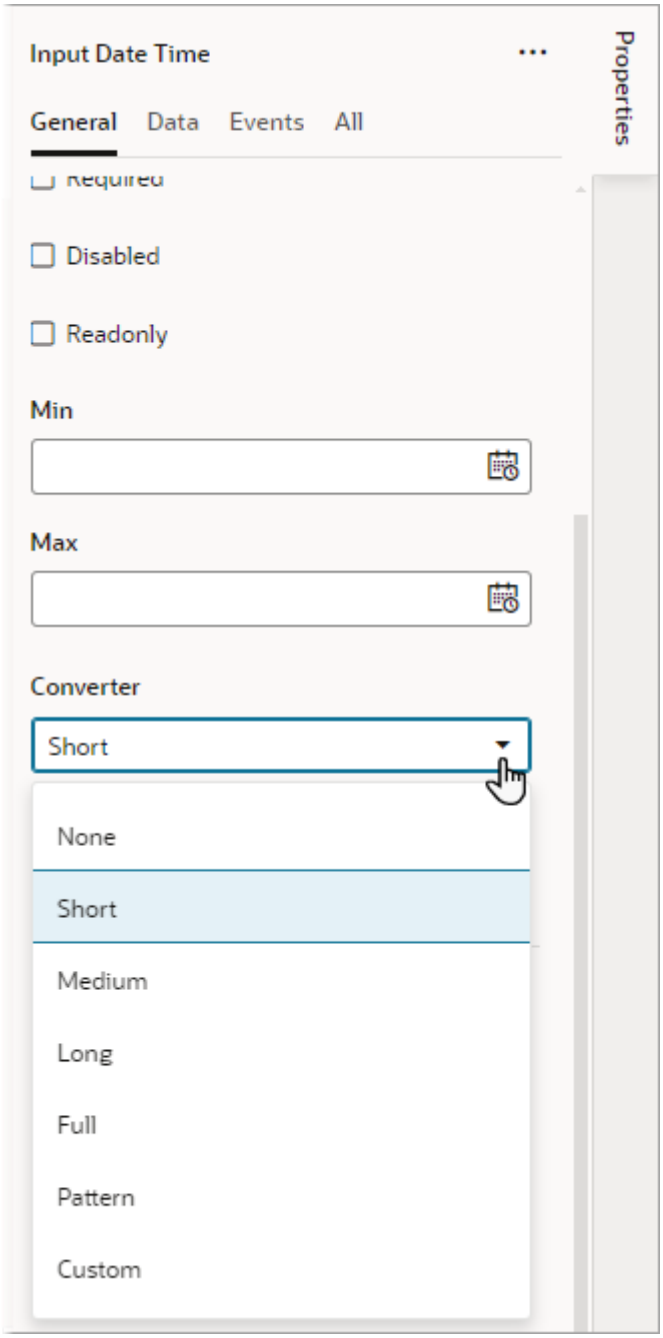
To assign a converter function to an Inline Date Time Picker and Inline Date Picker component:

1. In the Designer, click the component, then click the **All** tab.
2. Hover over the **Converter** field and click :



To change the date/time format for an Input Date Time, Input Date, or Input Time component:

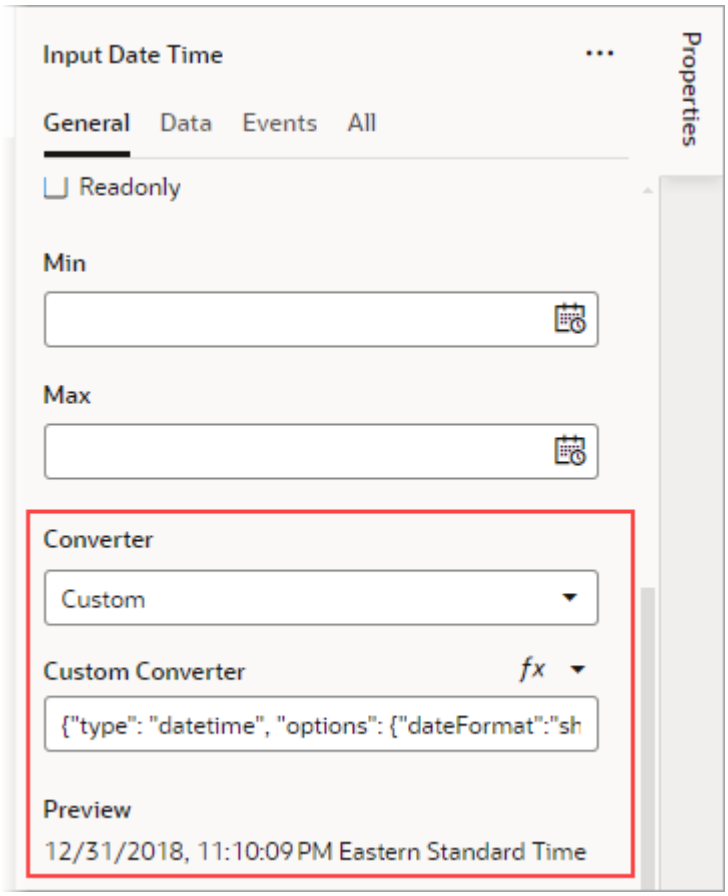
1. In the Designer, click the component, then find **Converter** in the Properties pane's **General** tab:



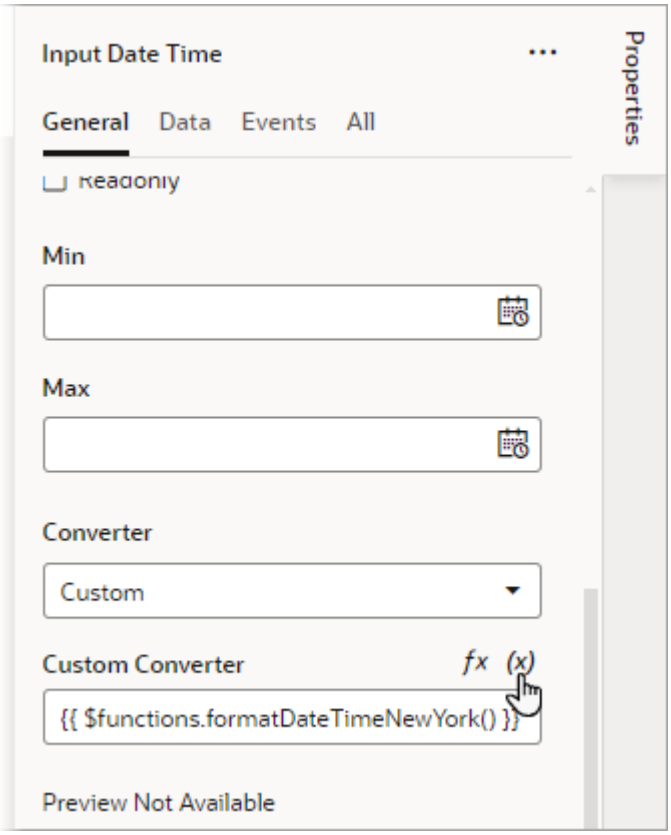
2. From the **Converter** list, choose a value:
   - **Short**, **Medium**, **Long**, or **Full**, if you want a predefined format
   - **Pattern**, to specify a pattern for the format (like `MM/dd/yy hh:mm`)
   - **Custom**, to provide formatting parameters to the built-in converter, or to assign your own converter function to the component.

   If you chose anything other than Custom, your work is done.
3. If you chose Custom, provide the parameters for the built-in converter, OR select your own converter function:
   - To provide the parameters for the built-in converter, enter them in the **Custom Converter** field. These parameters use the Oracle JET IntlDateTimeConverter API.

For details about the parameters that can be provided to the API, see Final Class: IntlDateTimeConverter.



- To select your own converter function, click the symbol that appears beside the **Custom Converter** field label:



Add Dynamic Components to Pages

You can add **dynamic components**, such as a table, form, or container, to your visual application's pages to define rules that control what's displayed at runtime to the user. Dynamic components help to show different items in a page's layout based on conditions in a rule. For example, you might configure a dynamic table so that certain columns are hidden and others are added when the user viewing the page is a manager. Or show a particular layout only when users viewing the page are on a tablet-sized screen or larger.

> **Note:**
>
> Dynamic components (`oj-dynamic-*`) and the If component (`oj-bind-if`) both use conditions to determine what's displayed on a page. While you can use `oj-bind-if` with JavaScript functions to do this, dynamic components provide a more declarative approach, making it easy for you to create layouts and to maintain and modify them after they've been created.

What are Dynamic Components?

A dynamic component, such as a form or table, does not render static content. Instead, it uses *rule sets* with *display logic* to determine what fields should be displayed on the page. Display logic is simply a set of conditions that you define. At runtime, the conditions are evaluated

based on the viewer's current circumstances (for example, the user's role) or the current data context (for example, the value of a field) to determine what is displayed.

You have two main objectives when creating pages using dynamic components: one, to configure the component's content the way you want it using *layouts* and *templates*, and two, to define the display logic that determines the layout and templates displayed in the component. In most cases you define the logic first, then configure the content that will be used in your logic.

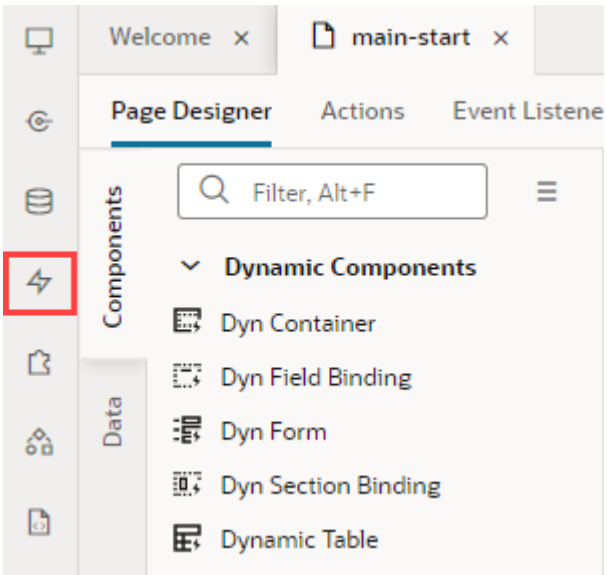
There are three types of dynamic components that can be used in visual applications: *tables*, *forms*, and *containers*. What is displayed in a component and how you customize it depends on what type of component it is:

Dynamic Component	Description
Dynamic table, dynamic form	In dynamic tables and forms, you customize which fields are displayed and how they are rendered. In most cases, you can hide, show, or re-order these fields, and can even create new fields based on existing ones. You can also apply field templates to control how certain fields are rendered at runtime.
Dynamic container	Dynamic containers are predefined areas in a page that can be used to display various types of content. Unlike a dynamic table or form, which can appear on multiple pages, a dynamic container is scoped to the page and can only ever appear on that page.

How to Create Layouts With Dynamic Components

Dynamic components provide a declarative way for you to bind a component's content to fields from your data sources. They serve as the building blocks for UIs that dynamically change what content is shown and how it is presented based on rules that you define.

Dynamic components are listed in the Components palette under Dynamic Components (you can enter `dynamic` in the palette's filter field to locate them). When you use dynamic components to show or hide content, you're defining *layouts*, all of which show up on the Layouts tab ( ) in the Navigator.



You have the option of creating a layout from scratch on the **Layouts** tab, where you can choose your data source, create a rule set with your own layouts and display logic, then

associate the rule set with a dynamic form or table. It's simpler though to start by adding a dynamic form or table to a page, then using quick starts to walk you through the basics.

Because a layout represents a set of data fields that can appear in one or more related dynamic components, you'll need to have your component's data source ready before you can work with layouts. Your data source can be a business object that stores your app's data or a service connection that receives data from REST APIs.

Here are the high-level steps you need to take to create a simple layout using a dynamic form or table:

To perform this action:	See this:
1. Add a dynamic form or table to a visual application's page	• Add a Dynamic Table to a Page • Add a Dynamic Form to a Page
2. Configure the rule set's display logic and layouts.	• Add Display Logic to Determine What's Displayed at Runtime • Create a Layout for a Dynamic Table or Form

For each dynamic component, you usually have a default rule and an accompanying layout. This default set is created for you when you configure a dynamic component on a page using a Quick Start (or, when you create a rule set in the Layouts' **Rule Sets** tab). You can then add additional rules (with matching layouts) to cover other scenarios. The default set is displayed if none of the conditions you define are met.

Besides layouts that control *what's* displayed on a page, you can control *how* something's displayed by using templates to visually design the field's area in a dynamic form or table. You can also set fields to be read-only for specific users and updatable for others.

In addition to rule sets, fields, and templates, the **Layouts** tab provides access to variables, actions, events, and event listeners, much like what's available when you add standard components to a page:

You can use these dynamic component editors just as you would use the editors for standard components, for example, to define events that you can hook action chains to.

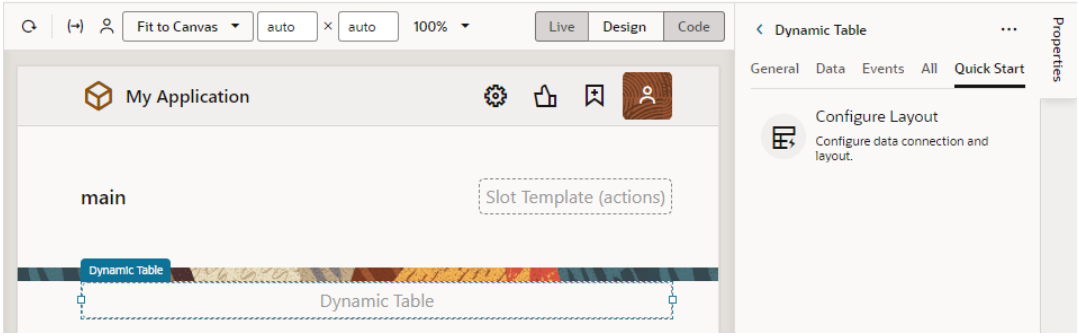
Add a Dynamic Table to a Page

Add a dynamic table component to your application's page when you want to display data in a table and use conditions to determine what's displayed to your users. Once you add a dynamic table to a page, you can use the Quick Start to create a rule set that you can configure with your own layouts and display logic.

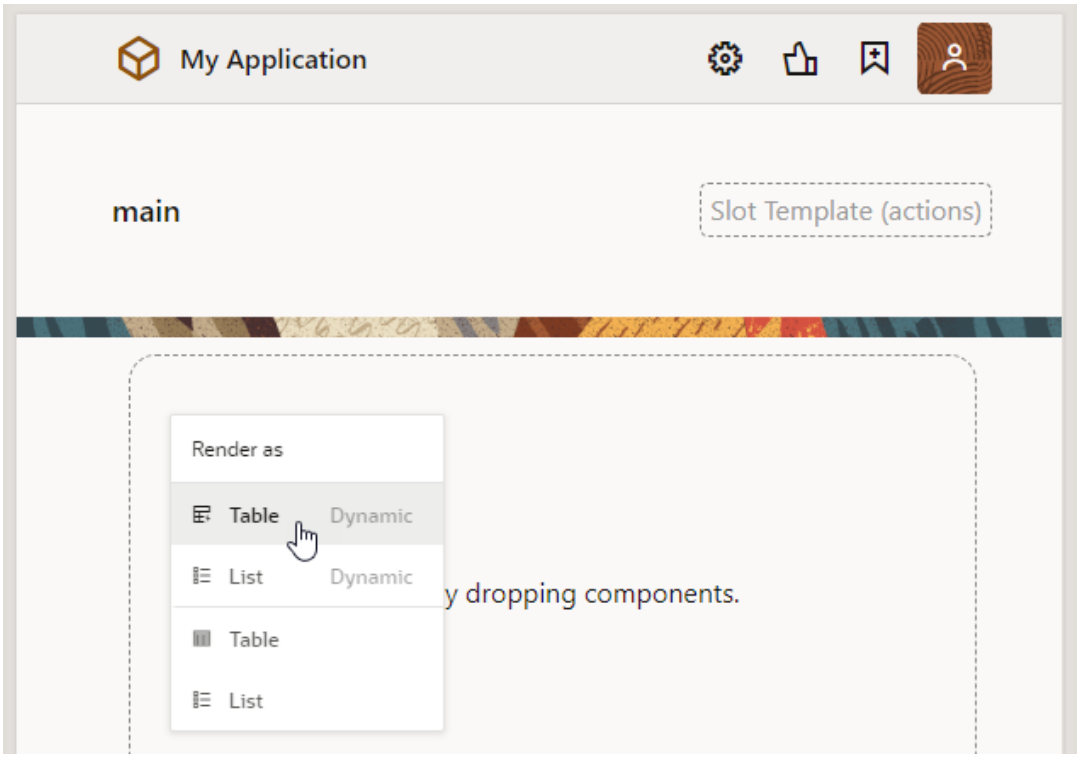
> **Note:**
>
> Before you begin, make sure you've defined a data source, such as a business object or an external service that calls a business object through a service connection. See Work With Business Objects or Manage Service Connections.

To add a dynamic table component to a page:

1. With your page open in the Page Designer, drag the Dynamic Table from the Components palette onto your canvas.



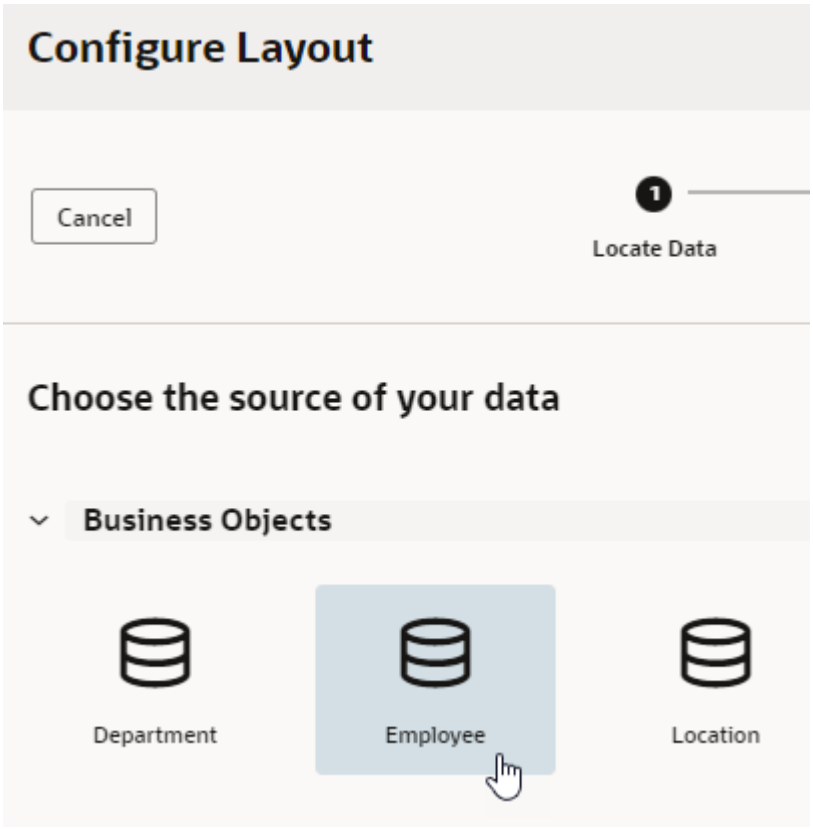
You can also drag the Get Many endpoint for the data source you want to bind to the table from the Data palette onto the canvas. After dropping the endpoint on the canvas, select **Table Dynamic** in the Render as menu:



Selecting the item in the menu will open the Select Rule Set page in the Configure Layout wizard.

2. Click the **Configure Layout** Quick Start in the Properties pane.
3. In the wizard's Locate Data page, select the data source to bind to the dynamic component. Click **Next**.

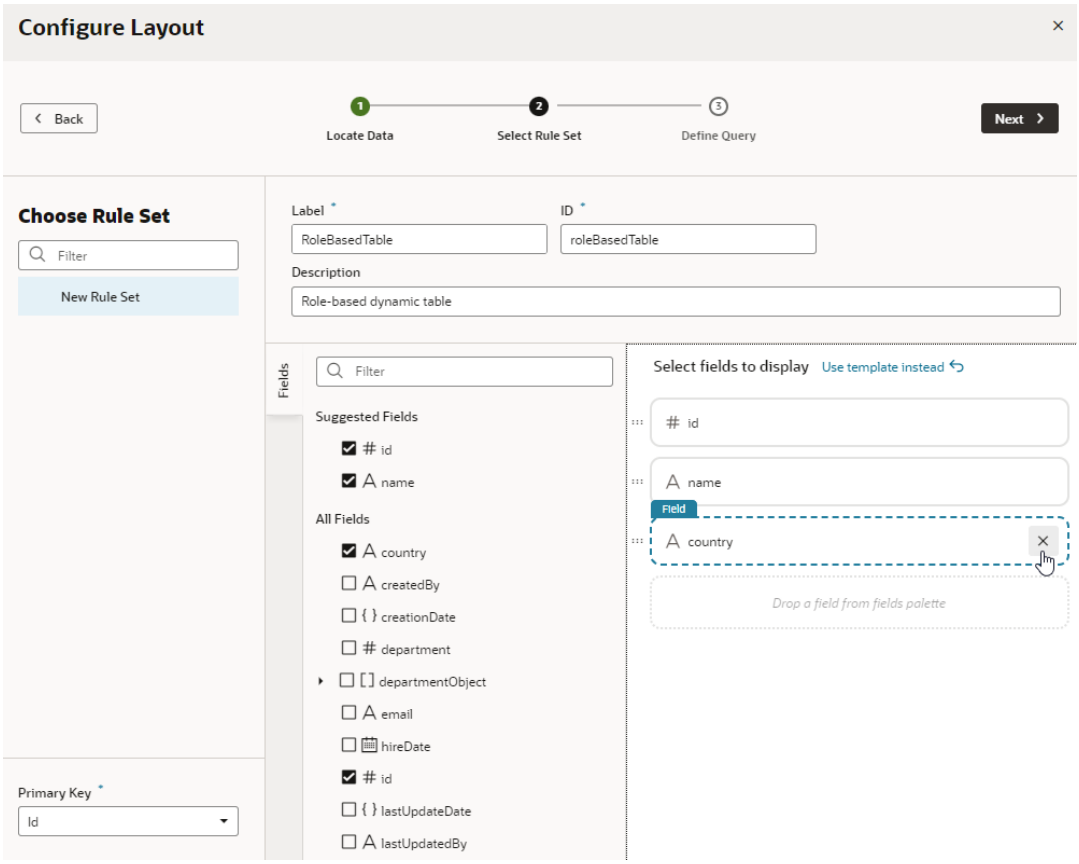
   This example uses the Employee business object as the data source:



4. In the Select Rule Set page, select **New Rule Set** (if necessary) and enter a label, ID, and description to associate the dynamic table with a rule set.

   Let's say you want to show one layout with some employee fields (for example, salary) when the user is a manager, and another layout (without the Salary field) for all other users. To do this, you'd start by creating a rule set (labeled `RoleBasedTable` for example), then select the fields you want to show by default for all users. This set of fields will be added to your initial layout (labeled `default`).

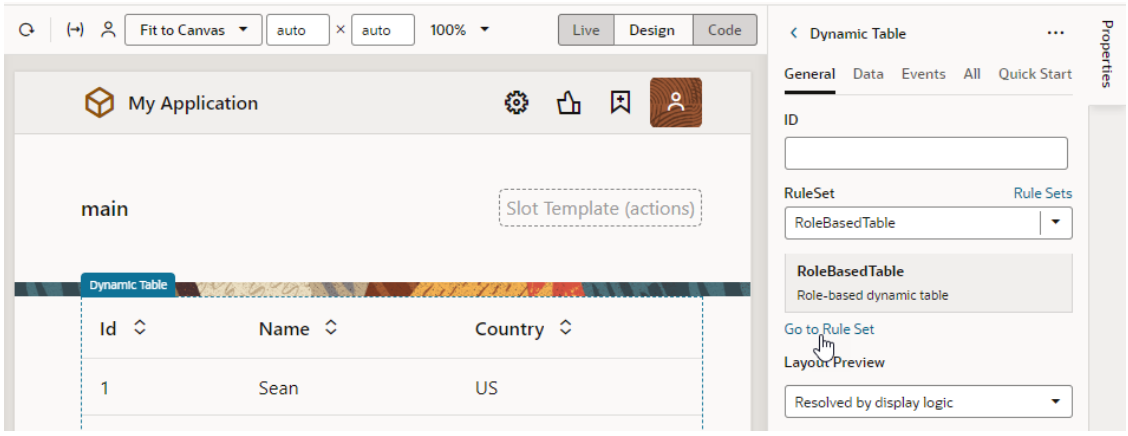
   The Fields palette lists all the fields and objects you can add to your layout. You can add a field or object by selecting its checkbox in the Fields palette or by dragging it from the palette onto the drop target area on the right. The columns appear in the order selected; use the handles to the left of each field if you want to re-order them. You can also remove a field by clicking its Delete icon, as shown here:



When you are done, click **Next**.

5. To limit the number of records returned, define a query on the Define Query page, then click **Finish**.

The dynamic table is created to use the layout with the fields you selected. You'll also see the newly created rule set under the **General** tab in the table's Properties pane.



You can configure the dynamic table's properties much like you would a standard table. For example, you can enable a single table row to be selected and set up an event that triggers an action chain to fetch the data of the selected row.

Click **Go to Rule Set** to open the rule set in the **Rule Sets** tab and configure it with your own display logic and layouts. For example, you might configure the display logic to show the

default layout when the user has the `Employee` role. You could then add another rule to show a different layout when the user has the `Manager` role. See Add Display Logic to Determine What's Displayed at Runtime.

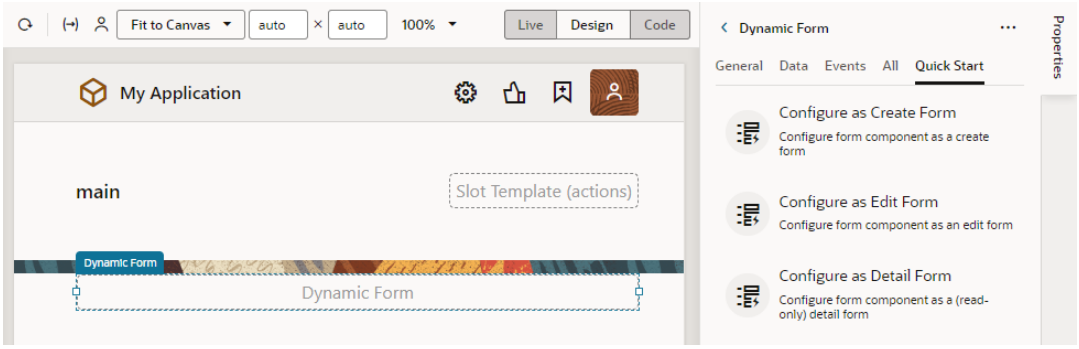
Add a Dynamic Form to a Page

Add a dynamic form component to your application's page when you want to display data in a form and use conditions to determine what's displayed to your users. Once you add a dynamic form to a page, you can use Quick Starts to create a rule set that you can configure with your own layouts and display logic.

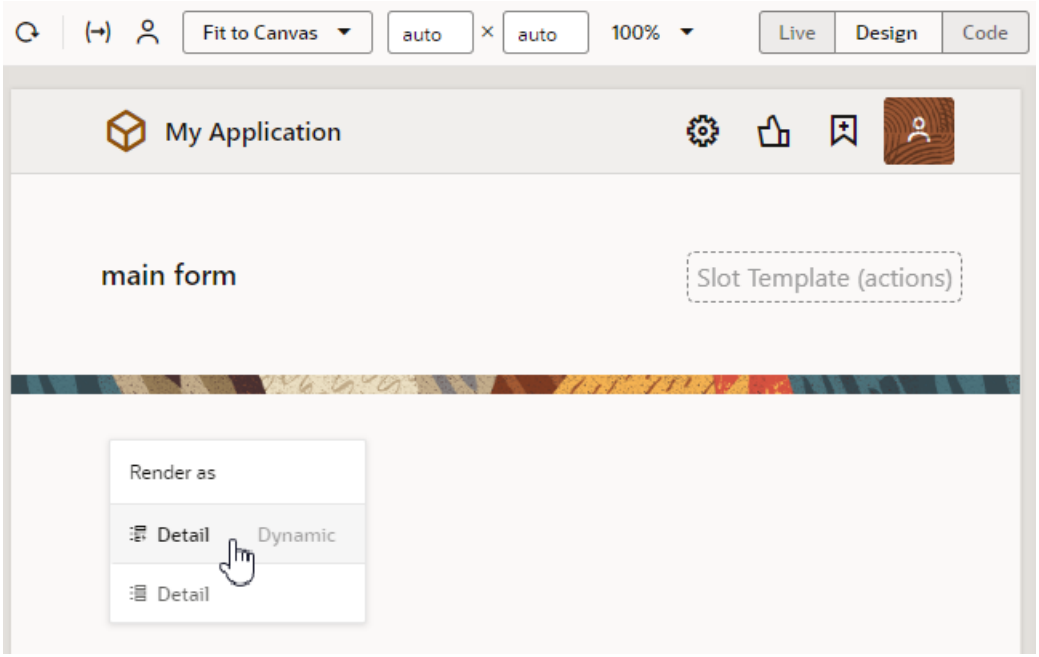
> **Note:**
>
> Before you begin, make sure you've defined a data source, such as a business object or an external service that calls a business object through a service connection. See Work With Business Objects or Manage Service Connections.

To add a dynamic form component to a page:

1. With your page open in the Page Designer, drag the Dynamic Form from the Components palette onto your canvas.



You can also drag an endpoint for the data source you want to bind to the form from the Data palette onto the canvas. After dropping the endpoint on the canvas, select the dynamic form or list component in the Render as menu:



Selecting the item in the menu will open the Select Rule Set page in the Configure Layout wizard.

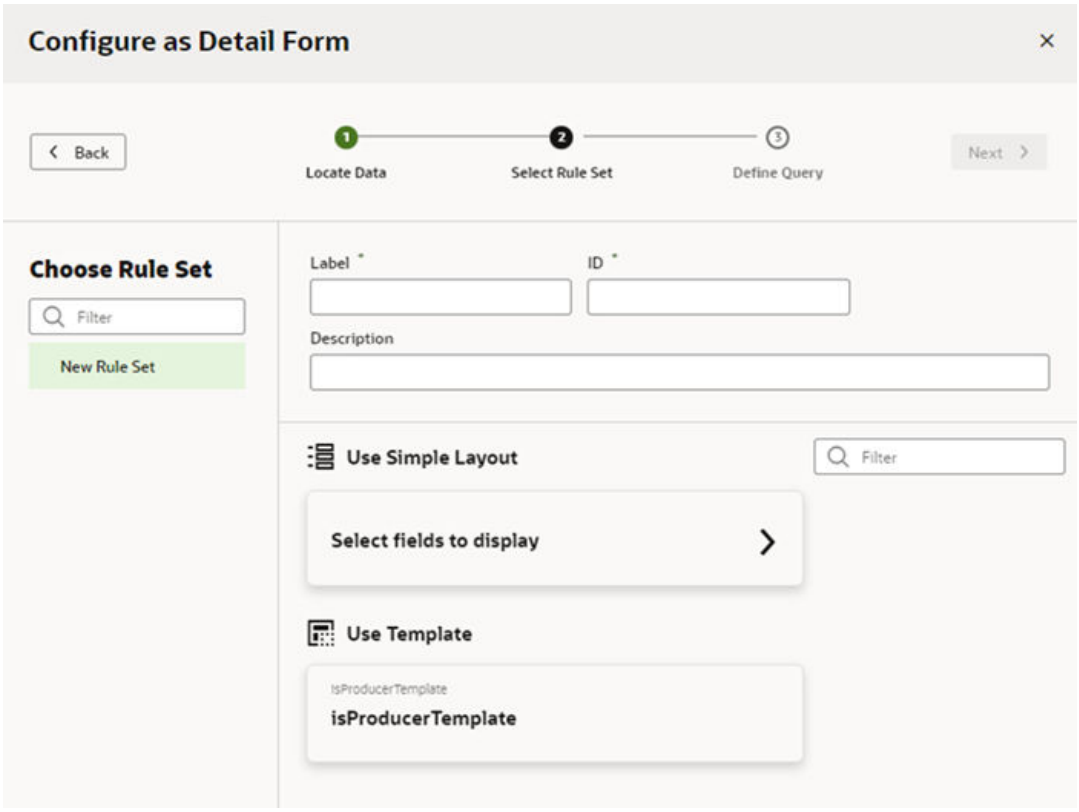
2. Select the Quick Start you want to use for the dynamic form in the Properties pane.
   - Click **Configure as Create Form** to create a form that interacts with the CREATE endpoint to create a new field in the data source.
   - Click **Configure as Edit Form** to create a form that interacts with the GET and UPDATE endpoints to edit a field's details in the data source.
   - Click **Configure as Detail Form** to create a form that interacts with the GET ONE endpoint to view details of a selected field.

   If you plan to use the **Configure as Edit Form** and **Configure as Detail Form** quick starts, you'll be prompted to provide endpoint parameters to be able to fetch and update the data of a particular row in the form. Make sure you create variables that you can map to these parameters before you use the quick start.

   Follow the quick start prompts to select a data source, rule set and fields to display in the form. Depending on the quick start you selected, you might have additional steps to complete.
3. Select the data source you want associated with the form.
4. In the Select Rule Set page, select **New Rule Set** (if necessary) to create a rule set, and provide a label and ID (and optionally, a description) for the rule set.

   You can choose an existing rule set if you've already create one you want to use. If you select an existing rule set, the quick start will open the rule set in an editor where you can add rules for the new form.
5. Select **Select fields to display** under Use Simple Layout.

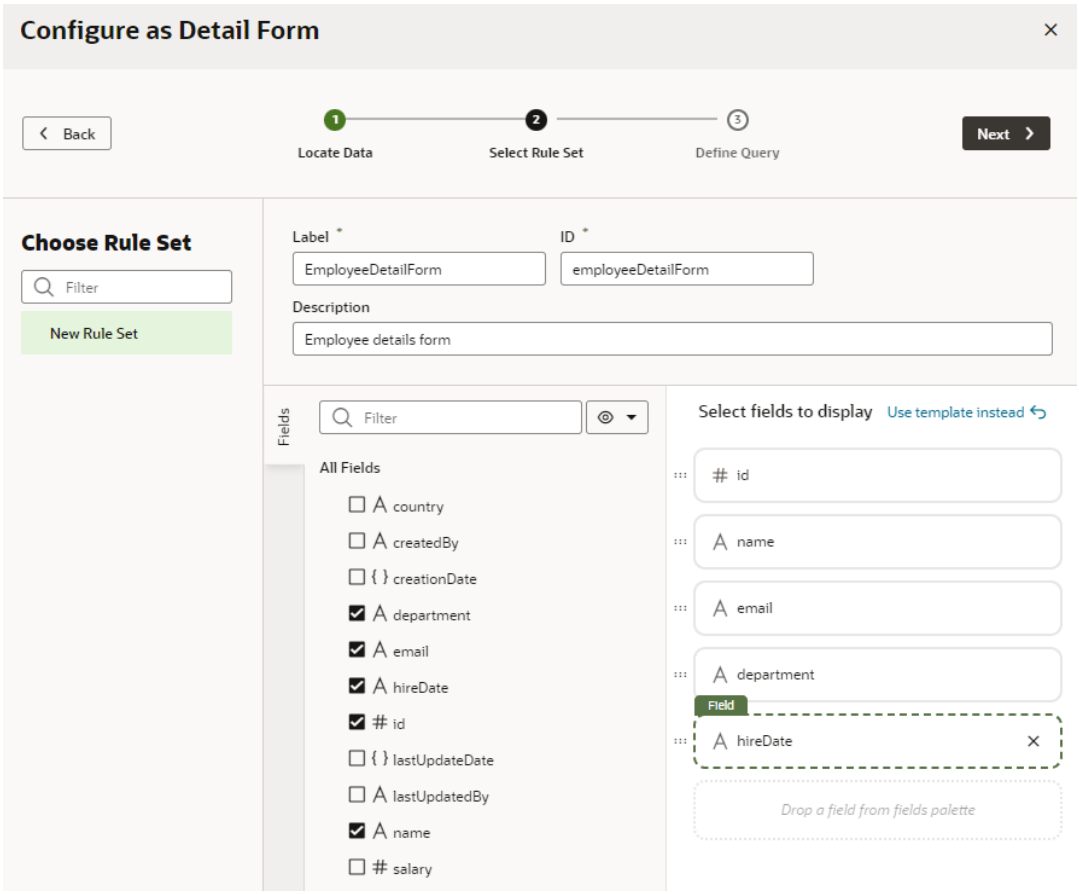
   This example shows the Select Rule Set step in the **Configure as Detail Form** quick start when a form template is also available. If you choose a template, the quick start will show you the fields defined in the template, and you can then add more fields.



If you select a template and it isn't right for this form, you can return to this pane and choose a different template, or click **Select fields to display** to create a layout without a template.

6. Select the fields to display in the form.

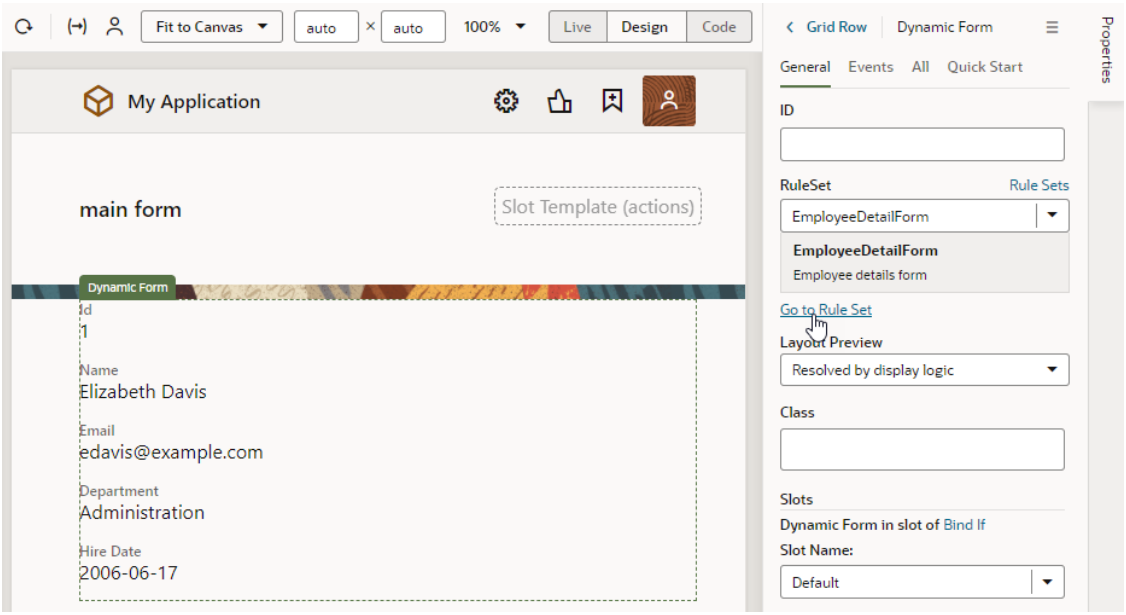
   This example shows the **Configure as Detail Form** quick start with fields selected from an Employee business object. These fields are added to the form's default layout. The form is also configured to fetch the data of a particular row in the form.



Click **Next** when you are done.

7. Map the sources to the target variables in your form, as needed. Click **Finish**.

The dynamic form is created with the fields you selected. You'll also see the newly created rule set under the **General** tab in the Properties pane.



Click **Go to Rule Set** to open the rule set in the **Rule Sets** tab. From here, you can configure your form's display logic and layouts; for example to show employee data to authenticated users viewing the page on a tablet-sized screen or larger.

Add Display Logic to Determine What's Displayed at Runtime

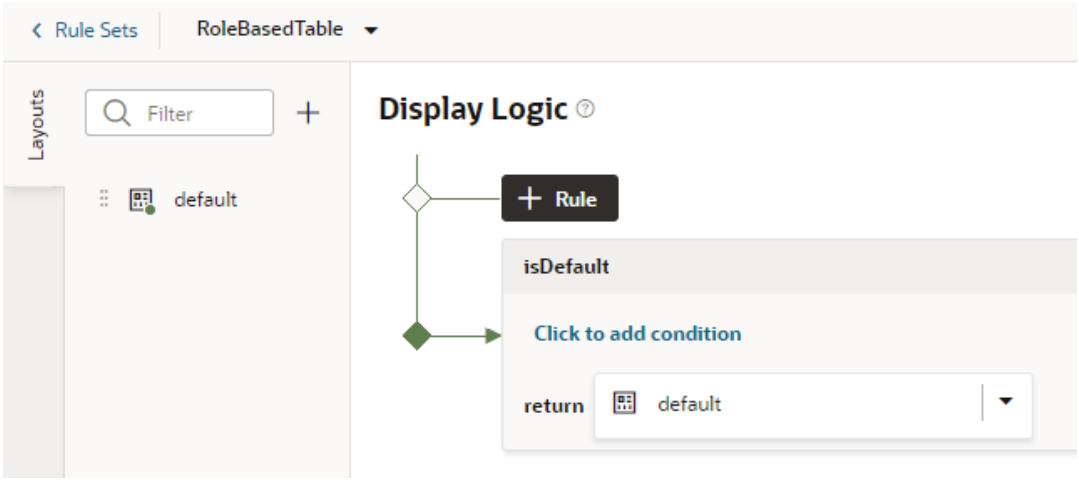
You control what's displayed at runtime on a page through the use of display logic, which you configure on the Layouts' **Rule Sets** tab.

Suppose you want to show employee data (say, an employee's salary) only when the user viewing the page is a manager. You'd then create a dynamic table or form and associate it with a rule set that checks the user's role. If the user has, for example, the Payroll Manager role, the page shows the layout that includes the Salary field. All other users would see the page with the default layout.

You can have more than one rule for a given component, and the rules are listed in a display logic tree when you select the dynamic form or table in the Rule Sets tab. The order in which they appear in the display logic tree is important because at runtime the rules are evaluated from top to bottom. The first rule where all the conditions are met—in this case, the user is a manager—is the one that's used, and the associated layout is applied to the component. No other rules are tested. Keep this in mind as you're working in the Rule Sets tab.

To configure the display logic for a dynamic component:

1. From a dynamic table or form's Properties pane, click **Go to Rule Set** to open the component's rule set in the Layouts' **Rule Sets** tab.



2. In the Rule Set editor, create a rule by clicking **+ Rule** to and giving it a name.

   The rule set for a dynamic component always contains a default rule. You can choose to edit it, copy it and use it as the basis for your own rules, or you can create a rule from scratch.

> **Tip:**
>
> It's helpful to give your rules meaningful names. For example, to show a particular layout only when the user is in Canada, you might call the rule `inCanada`. To edit a rule's name, hover near the name, then click , and click when you are done.

a. In your new or default rule, click **Click to add condition**.

b. Select an Attribute and Operator from the drop-down lists, and select or enter a Value.

   The Attributes drop-down list contains the fields and variables that you can use in your layout, and the Operators list contains the operators (for example, '=' and '<=') that are valid for the attribute you select. The Values list shows values already defined for the attribute (for example, 'true' and 'false'), if any, but you can also enter your own value.

   You can select built-in *context* variables that provide a way to access various pieces of information when building conditions for a rule. For example, you can check the size of the device accessing your app, or information about the user using the app such as their role or email. Built-in context variables include:

   - *$fields* variables determined by the fields displayed in the Fields editor. For example, the `$fields.firstName.value` lets you access the value of the First Name field in your data source. Look for these variables under the **Fields** group in the condition builder.

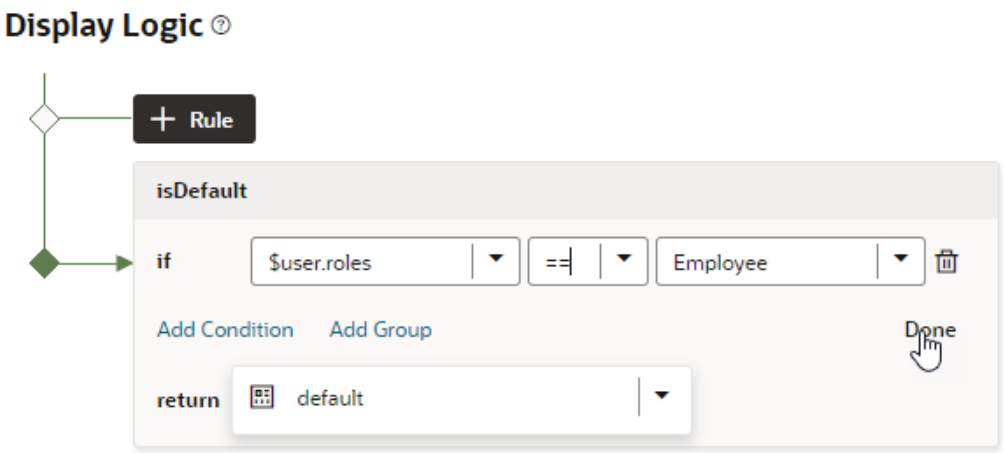
     > **Note:**
     >
     > For each field, regardless of type, you can choose `$numberValue` (for example, `$fields.ConflictId.numberValue()`) or `$value` (`$fields.ConflictId.value()`). You should use `$numberValue` when you know the field's value should contain a number. For example, if the ConflictId field's type is a string and you choose `$numberValue`, the field's value will be converted to a number, if possible. If the value can't be converted, the `$numberValue` will be `NaN` (Not a Number).
     > The only limitation is that `$numberValue` is limited by the maximum precision allowed by the Number type in Javascript.

   - *$responsive* variables determined by the screen size of the device the app is currently displayed on. For example, the `responsive.mdUp` variable's value is True if the current user is using a device where the screen width is 768 pixels or more, such as a tablet. Look for these variables under the **Responsive** group in the condition builder.
   - *$user* variables determined by the current user. For example, the `user.isAuthenticated` variable's value is True if the current user is an authenticated user. You can use the `user.roles` variable to check the role of the user using the app. Look for these variables under the **User** group in the condition builder.

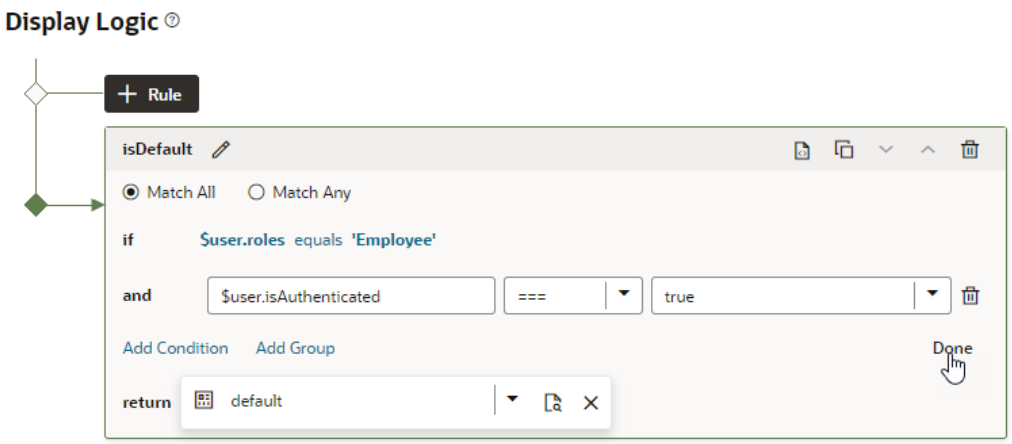
> **Note:**
>
> When using `user.roles`, the Value drop-down lists the available Oracle Cloud Applications job and abstract roles. (The drop-down will not list any duty roles. If you want to specify a duty role, you can manually type the duty role name in the Value field.)

You can also define your own context to augment your rules with values provided by your application, as described in Define Custom Contexts for Components in a Layout. These contexts will then become available through the *$componentContext* system variable. Look for these under the **Contexts** group in the condition builder.

Here's the condition to check whether the current user has the Employee role:



c. You can add more conditions and group conditions if you want to use more attributes to make the rule more precise, for example, you can make sure the user has the Employee role AND is authenticated. You would then create a rule with two conditions, and select **Match All** to require that both conditions be true.



d. Click **Done**.

e. In the return field, select the layout you want to apply when the rule is true.

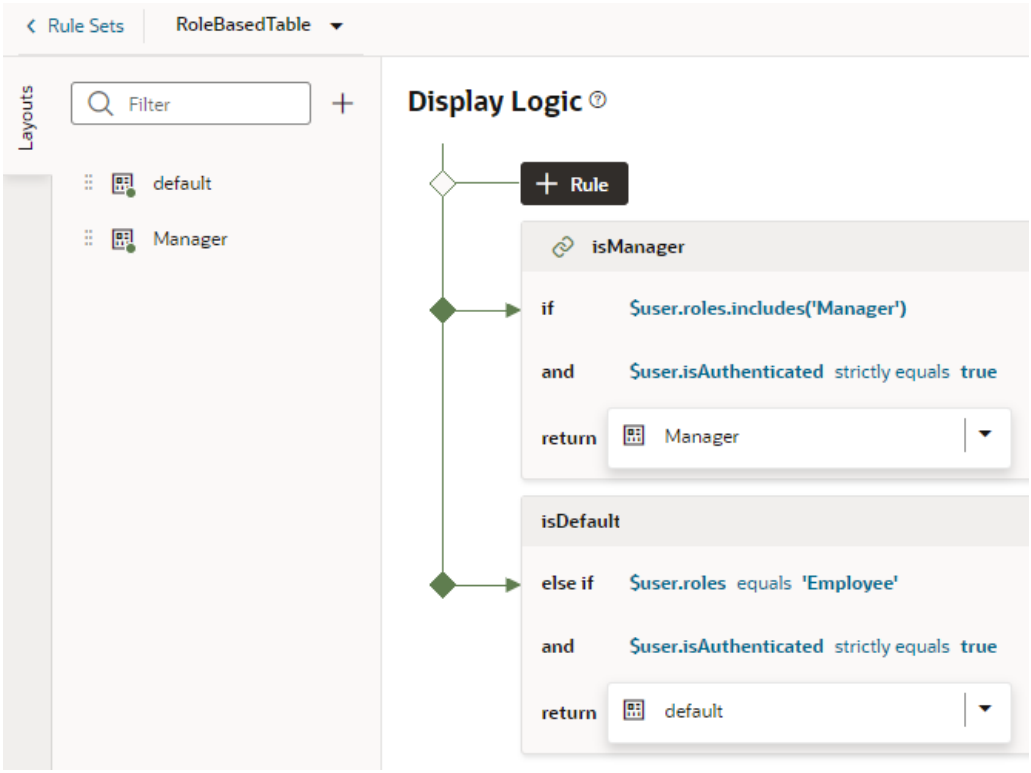
If you created a copy of a layout when you created the rule, it is selected by default in the return field. You can use the same layout with multiple rules.

3. Create more rules as required, for example, to display a Manager layout only to authenticated users who have the Manager role:

   a. Click the Duplicate icon ( ), then enter a name for the new rule in the Duplicate Rule dialog box.

      To also create a copy of the layout to use as a starting point, make sure that checkbox is selected. Click **Duplicate**.

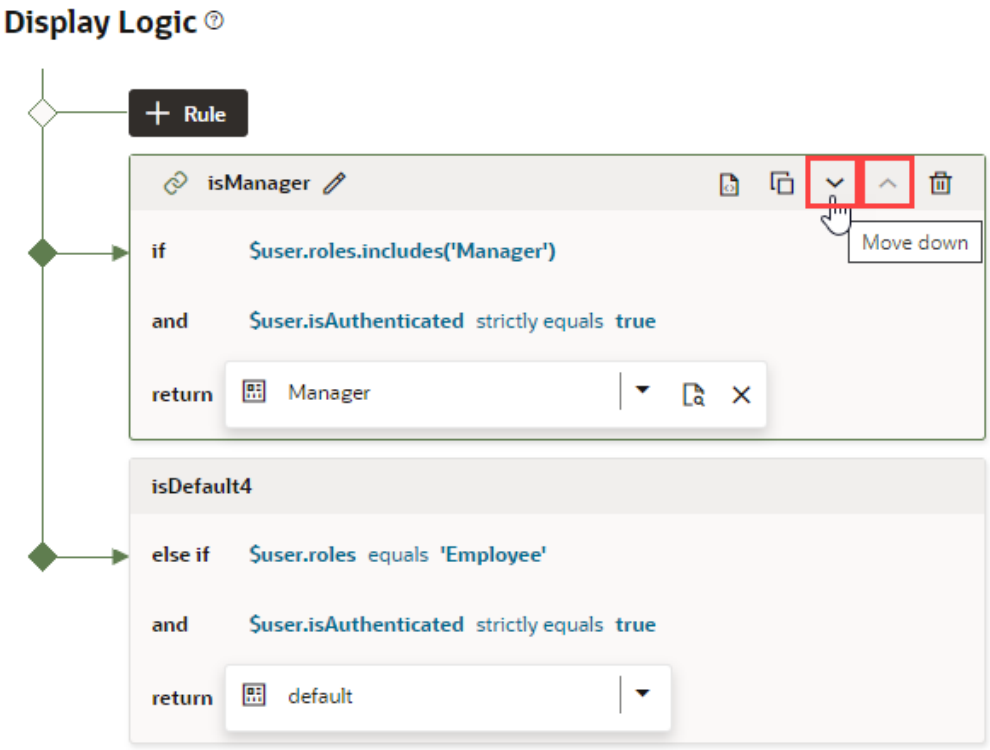
   b. Edit the new rule and define its conditions. To continue our example, you might set the rule to show the `Manager` layout when the current user's role is Manager and extend it to show only to authenticated users:



   If you click , you can see and edit the rule's expression. For the rule above, you'd see the following expression:

```
$user.roles.includes('Manager') && $user.isAuthenticated === true ?
'Manager' : null
```

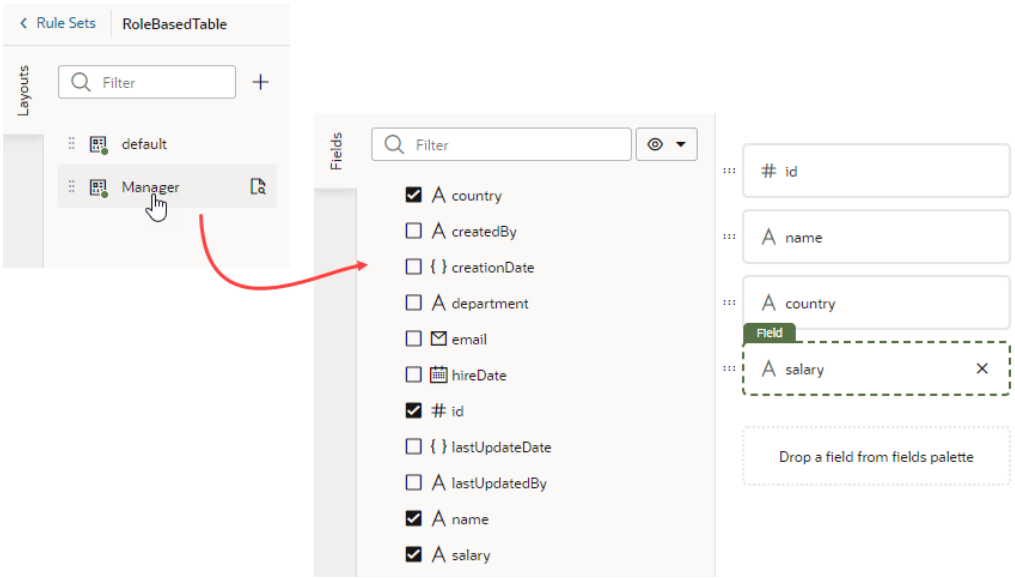
   c. Use the **Move Up** and **Move Down** buttons to make sure you have the rules in the order you want them evaluated.



The order and precision of your rules is important. The rules are evaluated from the top down, so the first rule evaluated as true will determine the layout that is used. When configuring the display logic, it's not a problem if there are rules that will never be used or evaluated.

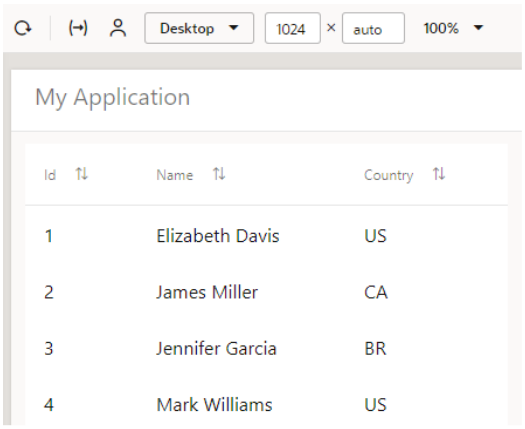
The default layout is usually the last rule in the display logic tree and is displayed if none of the conditions you've defined are met.

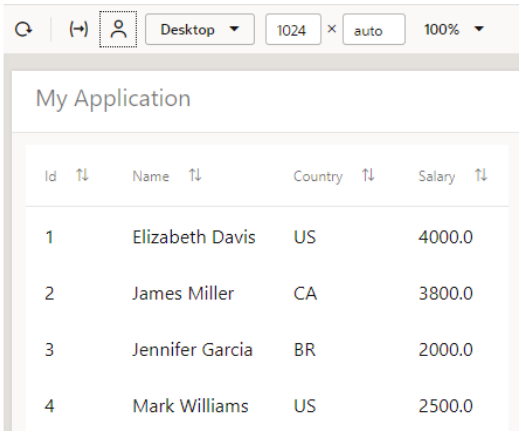
d. As part of configuring the new rule, click the newly created layout in the **Layouts** tab (`Manager`, in our example), then select the fields you want to show when the user is a manager (for example, the fields you included in the default layout plus salary):



4. Test your application to preview it in different roles (for example, as Employee and Manager). See Test Role-Based Access.

   Here's an example of a dynamic table with role-based logic enabled. On the left is what a user with the Employee role sees; on the right is what's shown to a user with the Manager role:





Create a Layout for a Dynamic Table or Form

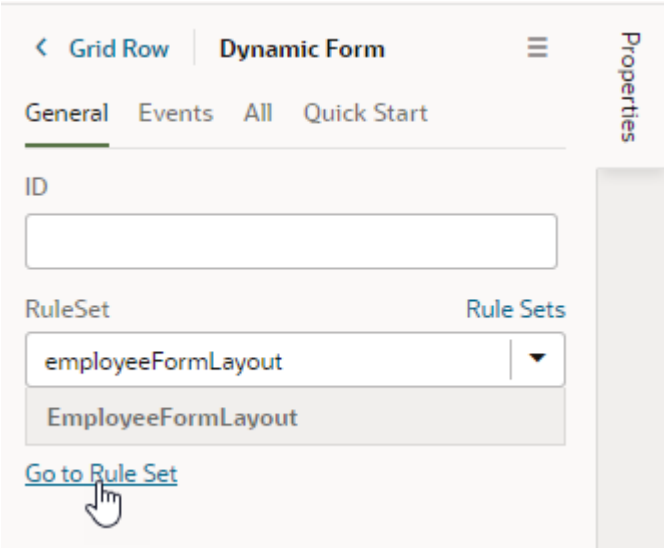
A *layout* defines the fields that are displayed in a dynamic component at runtime. You create and configure the layouts for a component's rule set in the Layouts' **Rule Sets** tab.

You can create multiple layouts for a single component, but only the layout associated with the rule that is found to be true **first** is the one applied to the component. For example, you might have three layouts that show different fields in a dynamic form based on a device's screen size. At runtime, the rules associated with the component are evaluated in the order they appear to see if the conditions set in that rule are met. If the condition is true—say, the current device's screen size is small—then the layout you selected for that rule is applied to the component and the user will only see the fields he needs in the form.

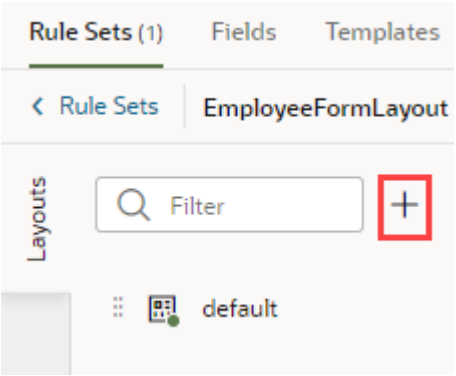
The fields you can display in a layout are determined by the fields available in the artifact's data source, say, a business object that has five fields. You can choose which of these five fields that you want to display in the dynamic component—and the order in which they should appear—but you can't include fields from other data sources.

To create a new layout:

1. When your page is open in the Page Designer, click the dynamic form or table you want to work with in the canvas area, or select it in the Properties pane.
2. Click **Go to Rule Set** in the Properties pane for the dynamic form or table:



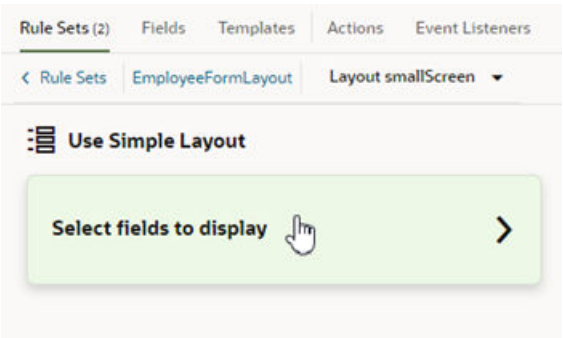
3. Click + in the rule set's Layouts pane, then enter a name for the new layout.



To illustrate, consider a dynamic form for employee data that displays the following fields in the default layout: Id, Name, Department, Email, and Hire Date. Now, say we want the form to show data based on screen size. To do this, we'll create two other layouts:

- A `SmallScreen` layout configured to show only Name and Email
- A `MediumScreen` layout configured to show Name, Department, and Email

4. Click the new layout name, then click **Select fields to display** to open the layout editor.



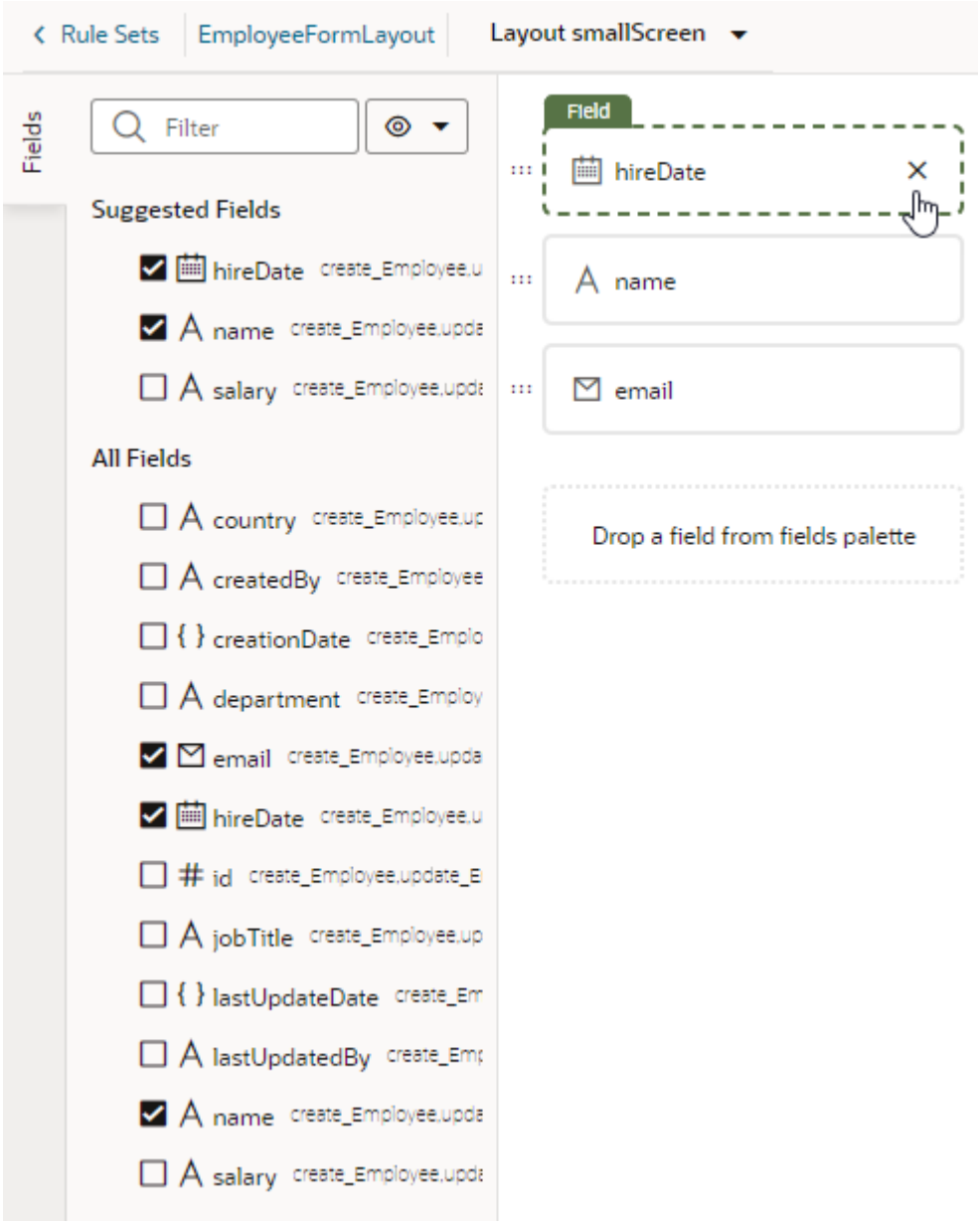
When you create a layout and haven't selected any fields for it yet, you'll see the **Select fields to display** option when you open the layout. (You won't see this option if the layout is a duplicate.) You'll also see the templates that already exist in the rule set listed as layout options. Click a template name if you want apply the template to the layout, otherwise, click **Select fields to display**.

5. Add fields from the Fields palette to the layout.

   The Fields palette lists all the fields and objects you can add to your layout. Required fields (those that have the Required property set in the Fields tab) are added to the layout by default. They also show as Suggested Fields in the Fields palette, emphasizing that they might be important or relevant to include in your layout.

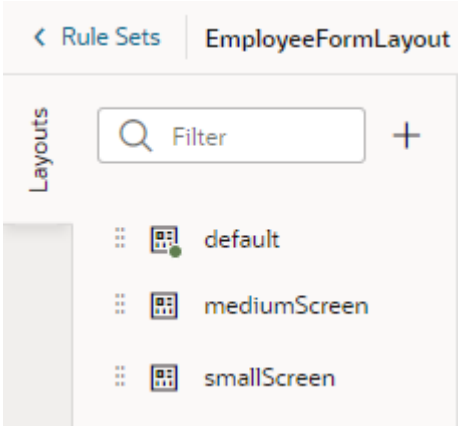
   To add a field or object, select its checkbox in the Fields palette or drag it from the palette onto the drop target area on the right. The columns appear in the order selected; use the handles to the left of each field if you want to re-order them. You can also remove a field by clicking its Delete icon.

   For example, here's what our SmallScreen layout might look like:



Because Hire Date and Name are required fields, they've been added automatically to the layout. For the purposes of this layout, we'll keep the Name field and remove Hire Date. (Salary, another required field, has already been removed).

6. Return to the component's rule set and repeat the steps as required to create other layouts, in our case, the `MediumScreen` layout.



After a layout is created, you can include it in a display logic rule. You can use the same layout in multiple logic rules.

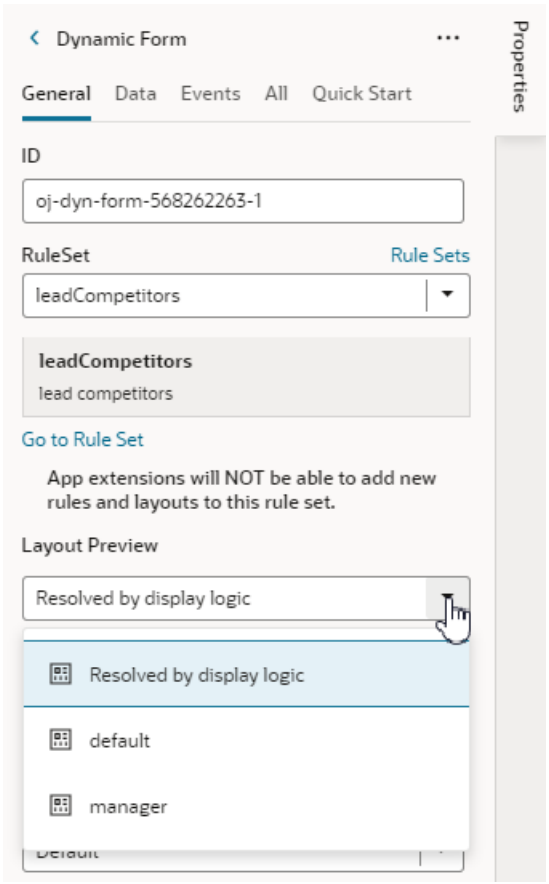
Preview Different Layouts

When you define multiple layouts for a dynamic table or form, you might want to preview how different layouts look when applied to your page. You can do this using Layout Preview in a dynamic component's Properties pane.

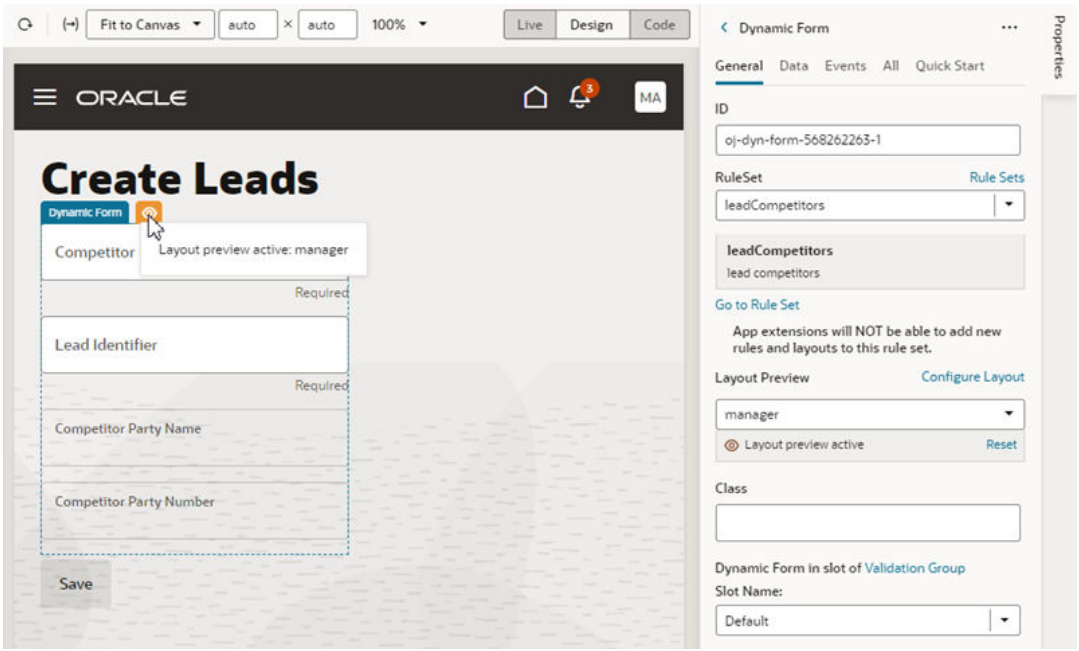
Layout Preview forces the Page Designer to render the layout you select and ignore the rules in the rule set. For example, if you created a layout that only managers can see, you won't see it in the Page Designer if you're not logged in as a manager. But you can use Layout Preview to override the display logic and render the page as it would to managers.

To preview a dynamic component's layout:

1. Open the page in the Page Designer and select the dynamic table or form you want to work with.
2. In the component's Properties pane, select the layout you want to preview in the Layout Preview list.



Selecting a layout will render it on the page. For example, when the `manager` layout is selected, the page shows the Salary field, meant only for managers:



A preview icon ( ) also appears next to the dynamic component, indicating that a layout preview is currently active. Click the icon to see which layout is being previewed.

3. If you want to make changes to the layout, click **Configure Layout** in the Properties pane and update the layout in the editor. Then return to this page to preview the layout again.
4. When you are done, click **Reset** to return to the default `Resolved by display logic` option.

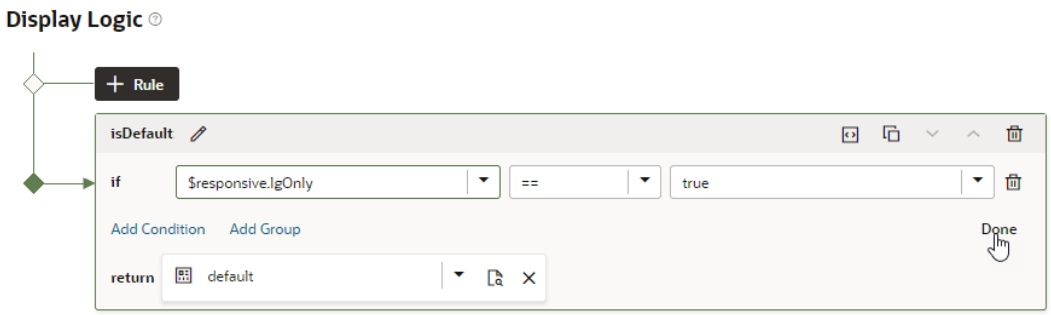
Responsive App Display Logic Example

The following example shows how to configure display logic for responsive apps. Suppose you want a dynamic component that shows different fields based on the device's screen size, say, small, medium, and large screens. You'd then create a rule that checks the current user's device screen size and applies the layout that contains the desired fields for that screen size.

To illustrate, consider a dynamic form that displays the following employee fields in the default layout: Id, Name, Department, Email, and Hire Date. If the user's device screen is small, you might want the page to show a particular layout (say, the `SmallScreen` layout) with only the Name and Email fields. If the user's device screen is medium, you might want the page to show another layout (for example, the `MediumScreen` layout) with the Name, Department, and Email fields. If the user's device screen is large, you might show the `default` layout.

To configure a rule set for responsive logic:

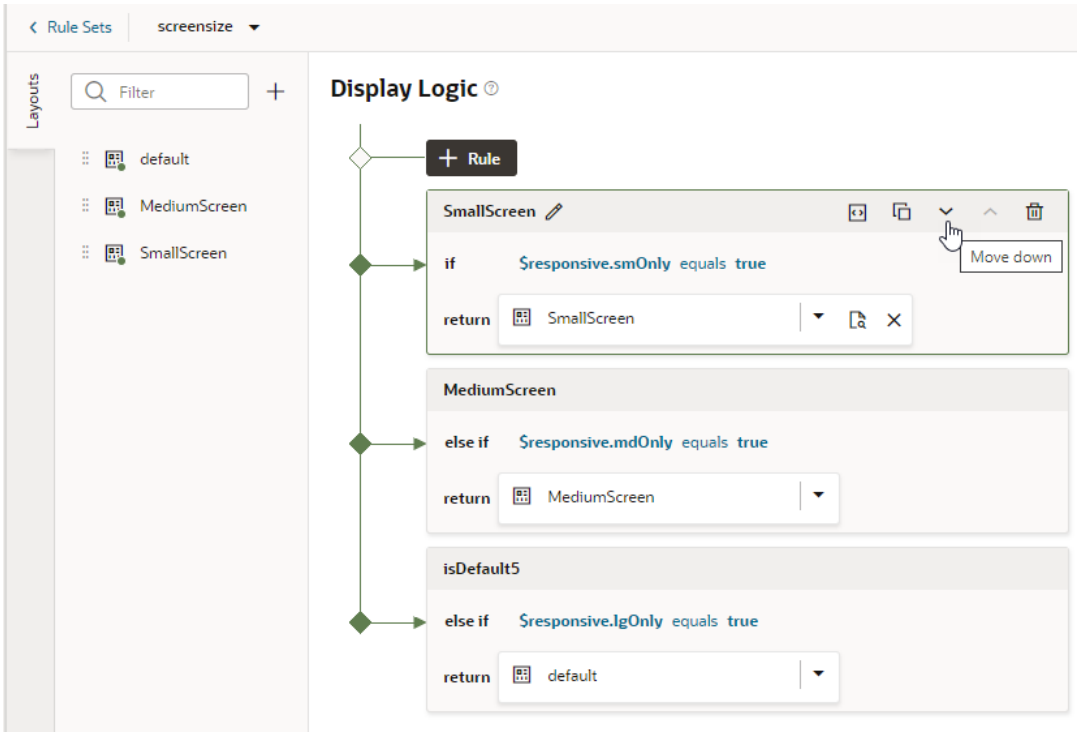
1. Update the default rule to show the default layout when the device's screen size is large:
   a. In the Rule Set's Display Logic section, click **Click to add condition**.
   b. Choose `lgOnly` under **Responsive** in the Attributes drop-down list, select `===` from the Operators list, then remove one of the equal signs, and select `true` as the Value.
   c. Click **Done**.



2. Duplicate the existing rule as required and use it as the basis to create more rules, in our case, the `MediumScreen` and `SmallScreen` rules. During this step, you have the option of creating copies of a particular layout which you can then update to show the fields you want when the device screen is small and when it is medium.
   a. Click the Duplicate icon ( ), then enter a name for the new rule in the Duplicate Rule dialog box.

      To also create a copy of the layout to use as a starting point, make sure that checkbox is selected. Click **Duplicate**.
   b. Edit the new rule and define its conditions. To continue our example, you might use the `mdOnly` attribute to show the `MediumScreen` layout when the current user's screen size is medium and the `smOnly` attribute to show the `SmallScreen` layout when the current user's screen size is small.

c. As part of configuring the new rules, click the newly created layouts in the **Layouts** tab (`MediumScreen and SmallScreen`), then select the fields you want to show when the device screen is small (Name and Email) and when it is medium (Name, Department, and Email).

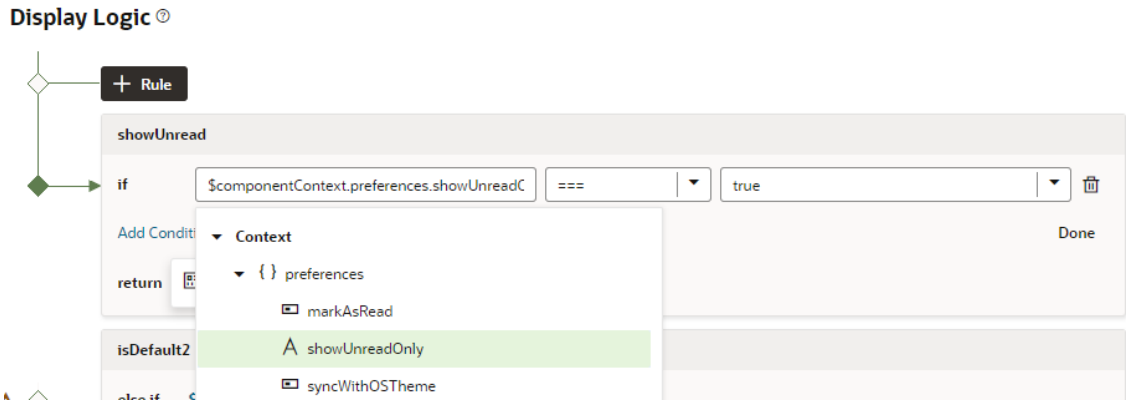


3. Use the **Move Up** and **Move Down** buttons to make sure you have the rules in the order you want them evaluated.
4. Test your application using different screen sizes in the Page Designer toolbar. For example, use iPhone to test the display logic on a small screen, iPad to test a medium-sized screen's display, and desktop to test a large screen's display.

Define Custom Contexts for Components in a Layout

In addition to built-in system variables that can be used within a dynamic component (for example, device size or user role), you can define a context to pass information that might otherwise be inaccessible to a dynamic component. This information can be values produced by your application that come from outside your layout, such as page variables, or details from other parts of your application.

For example, your application might have some preferences that could be useful when you're using a dynamic table or form to build a layout. These preferences can be any arbitrary value, but as a layout developer, you don't want your layout to consume values that might not be valid. So Visual Builder provides a *componentContextType*, a contract that helps you define the shape—or parameters—that the component context will accept. Once the componentContextType is defined, these parameters become part of the *$componentContext* system variable and can be used anywhere in the layout. Here's an example of *$componentContext* parameters in the rule set condition builder:



Visual Builder also provides the *baseComponentContextType* to define parameters at the service level. Much like componentContextType, parameters defined by the baseComponentContextType contract can be used anywhere in the layout where the *$componentContext* variable is available. But unlike component context parameters, base component context parameters can also be used in OpenAPI schema.

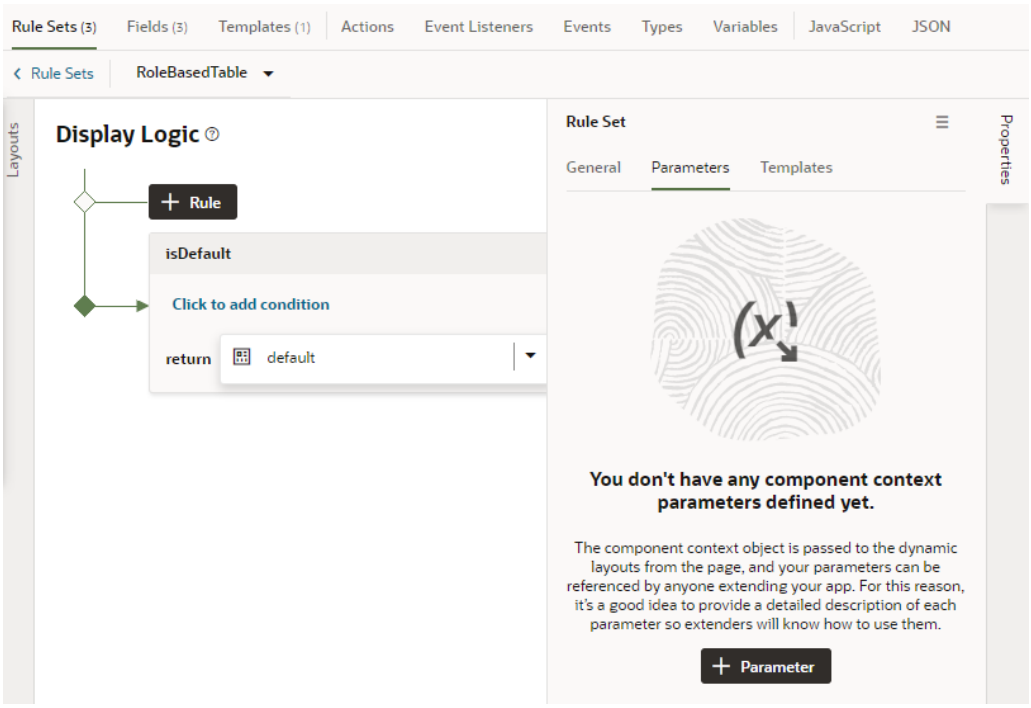
Let's say you want to control access to your layout based on whether OAuth 2 authentication is enabled for the user. You could use *$componentContext* to pass external information through the `isOAuth2enabled` field, then bind that to the query parameter of the URL in the `openapi.json` file:

```
"employees": {
  "operationRef": "http://example.com/hrApi/resources/1.0/employees/describe#/
paths/~1employees/get",
  "parameters": {
    "token":
"AUTH_TOKEN=[[$componentContext.securityInfoMap.isOAuth2Enabled ? 'foo' :
'bar' ]]"
  }
}
```

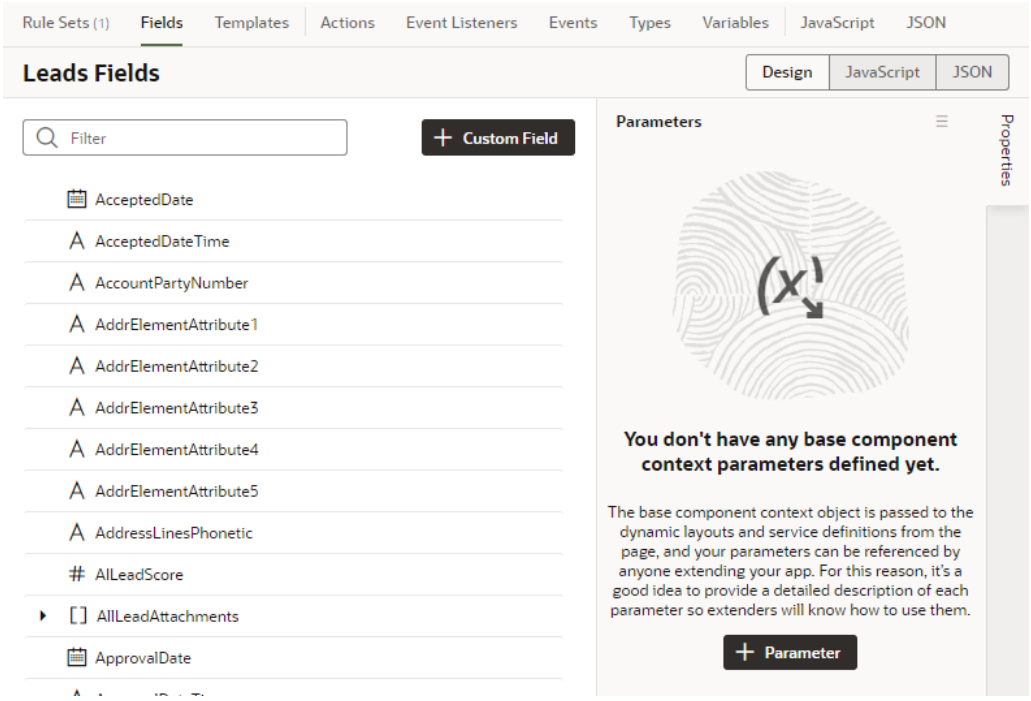
The baseComponentContextType is the contract that makes it valid to use `$componentContext.securityInfoMap.isOAuth2Enabled` in `openapi.json`.

To define parameters that can be passed to a layout:

1. Choose the type of parameters you want to define: component context or base component context.
   - To define component context parameters, go to your layout's Rule Set editor. Then in the Properties pane, click the **Parameters** tab.

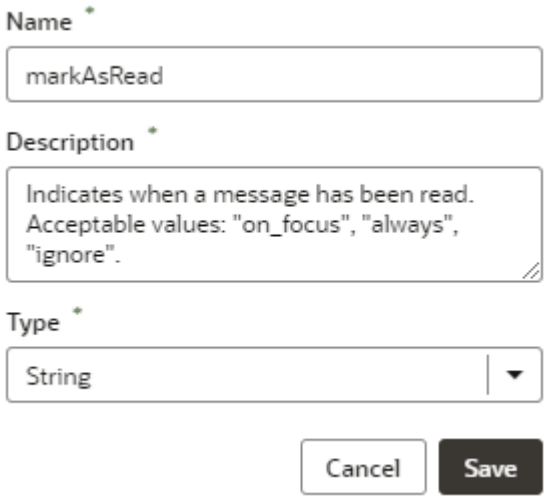


- To define base component context parameters, go to your layout's Fields editor and view the **Parameters** tab in the Properties pane.

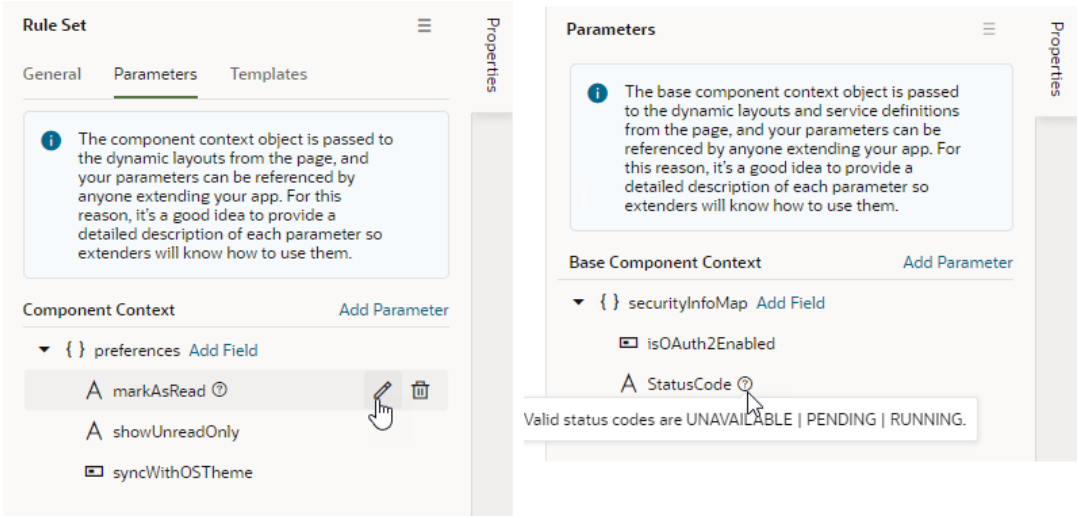


2. Click **+ Parameter** (or **Add Parameter** if other parameters exist).
3. Enter a name and description, then select the type (for example, select object if you want the parameter to have sub-fields). Because your parameters can be referenced by anyone extending your app, it's important to provide a description of the parameter and values it accepts.

4. If you chose array or object, click **Add Field**, then enter the name and description and select a type.



5. Repeat the steps to add as many parameters and fields as you need. Here's an example showing component context parameters (set in the Rule Sets tab) on the left and base component context parameters (set in the Fields tab) on the right:



The parameters, now available to all rule sets in a layout, will be accessible in the expression editor and the condition builder.

Use Field and Form Templates

You can customize how a dynamic component is rendered on the page by editing layouts to group fields together and to apply templates to the layout and fields.

Control How a Field is Rendered with Field Templates

You can customize how a field is rendered in a layout for a dynamic form or table by applying a *field template*. A field template contains UI components, for example, text fields or images, and defines their properties, such as styling details. Components in a template can access the variables, constants, action chains, and event listeners defined in the layout.

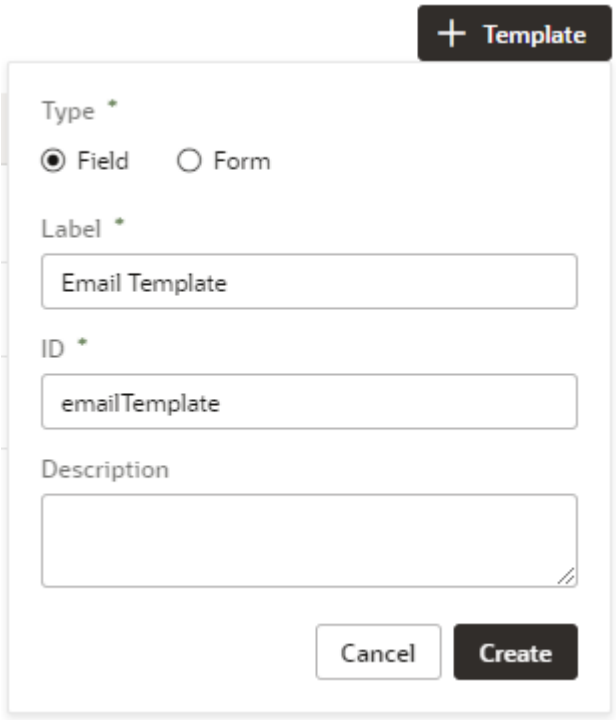
You might define a default template for a field, which is then applied to the field in every layout. You can override the default template if you want to apply your own template. Suppose the visual app has applied a template called BoldType to the Update field. The Update field will have the BoldType template applied in every layout where it appears. However, you can create a field template called Italics and override the BoldType template, either in specific layouts or across all the layouts that you create. You can apply your Italics template to multiple fields, as long as they are part of the same layout.

To create a field template for a field in a dynamic form or table:

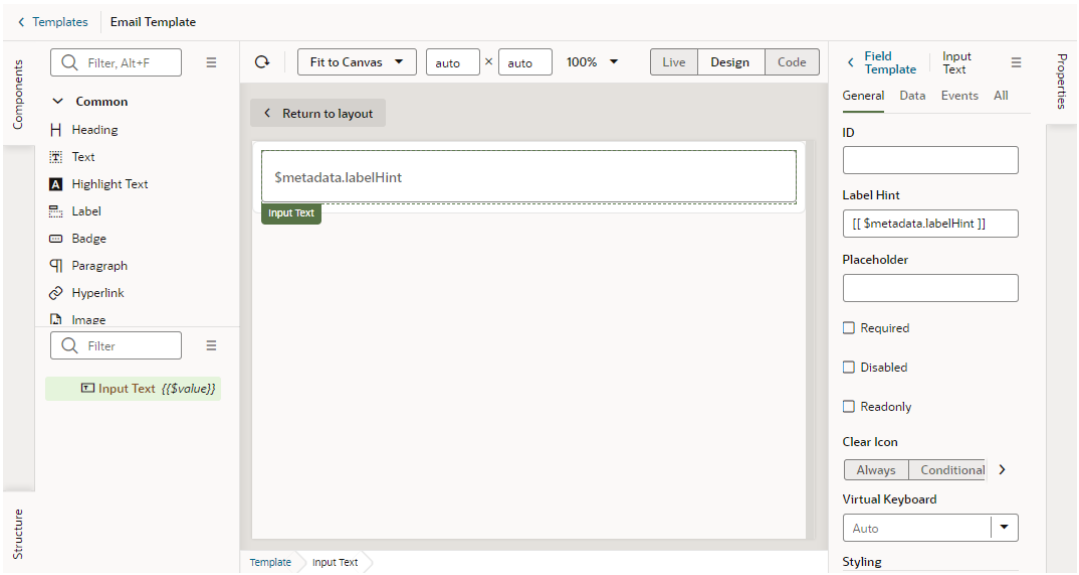
1. Open the layout's **Templates** tab.

   The Templates tab displays a list of field and form templates that are already defined for the artifact.

2. Click **+ Template**. Select **Field**, specify the Label (the ID is generated for you), and click **Create**.



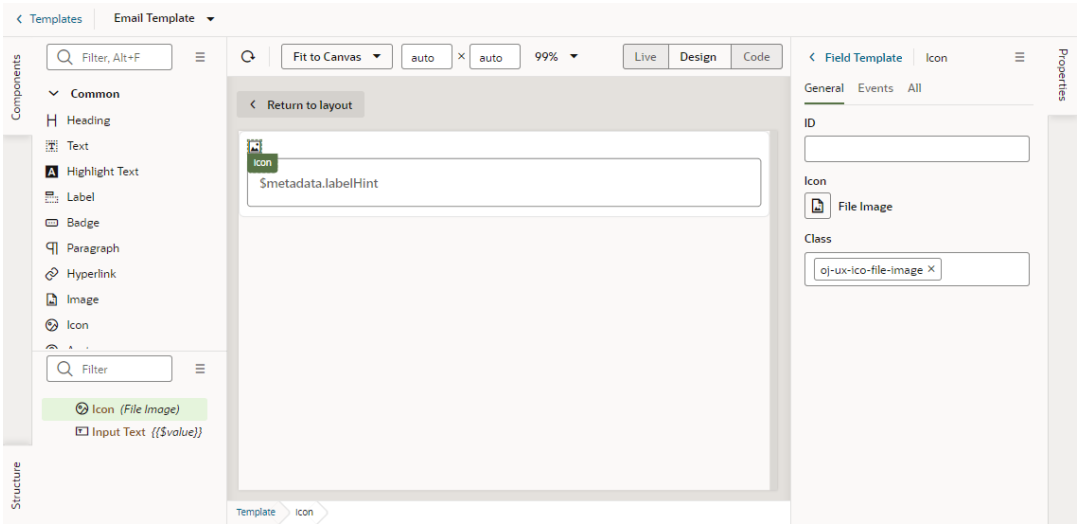
The new field template opens in the template editor, which contains a Components palette, Structure view, canvas, and a Properties pane. In the Structure view, you'll see that your new field template includes an automatically generated Input Text component. This is used to display the data and display name when you apply the template to a field in the layout.



3. In the Templates editor, add any other UI components you want to display in the template by dragging them from the Components palette onto the canvas or the Structure view.

   You can add more UI components above or below the Input Text component, or replace the Input Text component with a different one, for example, to render a field using a Rating Gauge component instead of an Input Text component.

   In this image, you can see in the Structure view that the template contains an Icon component and an Input Text component:



4. Select a component on the canvas or in the Structure view, then edit its properties in the Properties pane.

   Just like when you are working in the Page Designer, the Properties pane might contain several tabs for editing the component's properties. For example, if you added an icon component to your template, you might decide to also create an event in the Events tab. If you did this, an event listener and action chain would be created for you, and you would then need to edit the action chain to define the behavior.

Alternatively, you can edit the field template's code directly in the Code editor, and use the editor's code completion to help you. For example:

```
<!-- Contains Dynamic UI layout templates -->
<template id="emailTemplate">
  <span class="vb-icon vb-icon-envelope"></span>
  <oj-input-text value="{{ $value }}" label-
hint="[[ $metadata.labelHint ]]"></oj-input-text>
</template>
```

After you've created the template, click **< Templates** to view your template added to the list of field templates in the Templates tab. From here, you can open and duplicate the templates you've created.

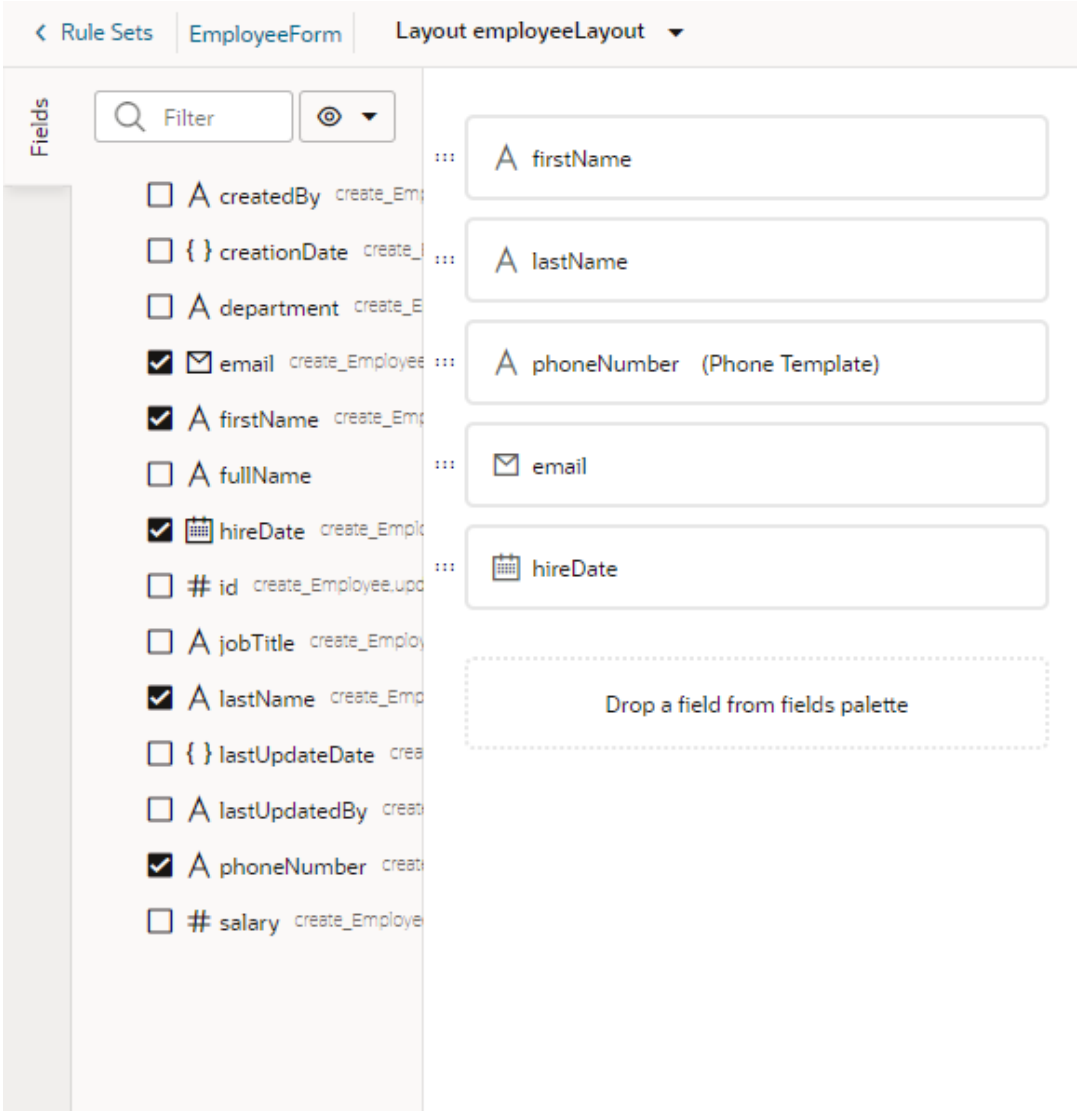
Apply a Template to a Field

Once you've defined a field template, you can apply it to a field in a dynamic form or table's layout, making it the default template applied to that field in every layout in that rule set.

To apply a template to a field in a layout:

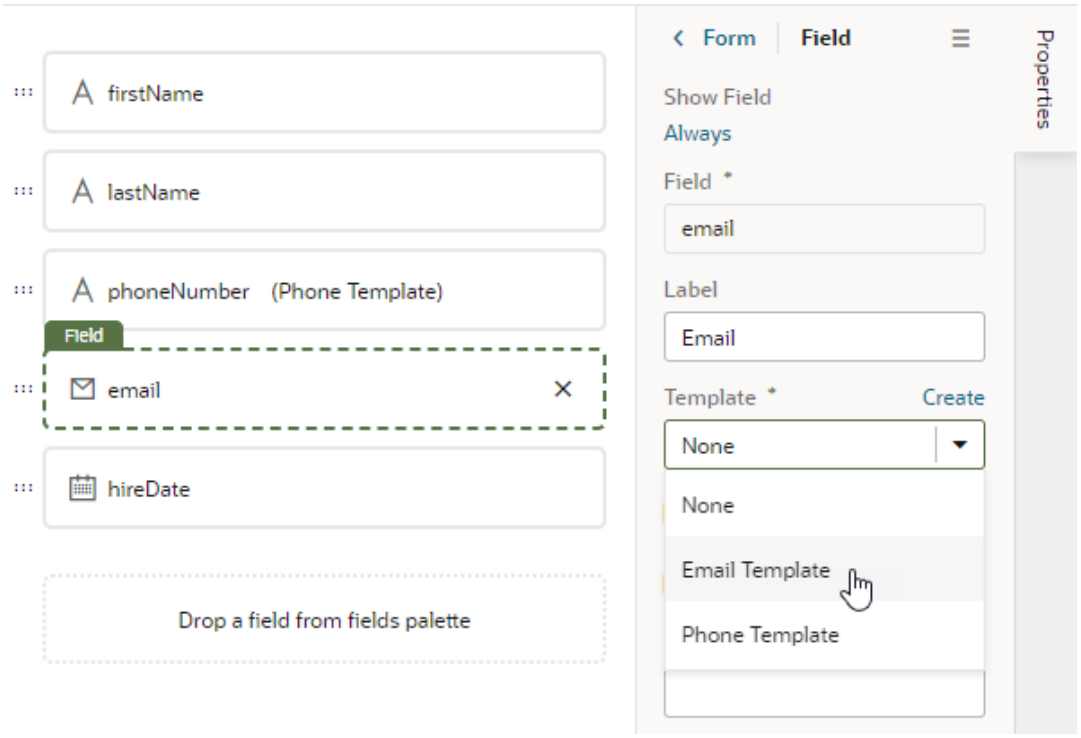
1. In the rule set editor, click the name of the layout name you want to work on.

   The center pane of the layout editor lists the fields that will be displayed in the layout and any templates that are applied to them (as shown in the image for the `phoneNumber` field). If you duplicated an existing layout, your new layout might already list some fields, or have templates already applied to fields.

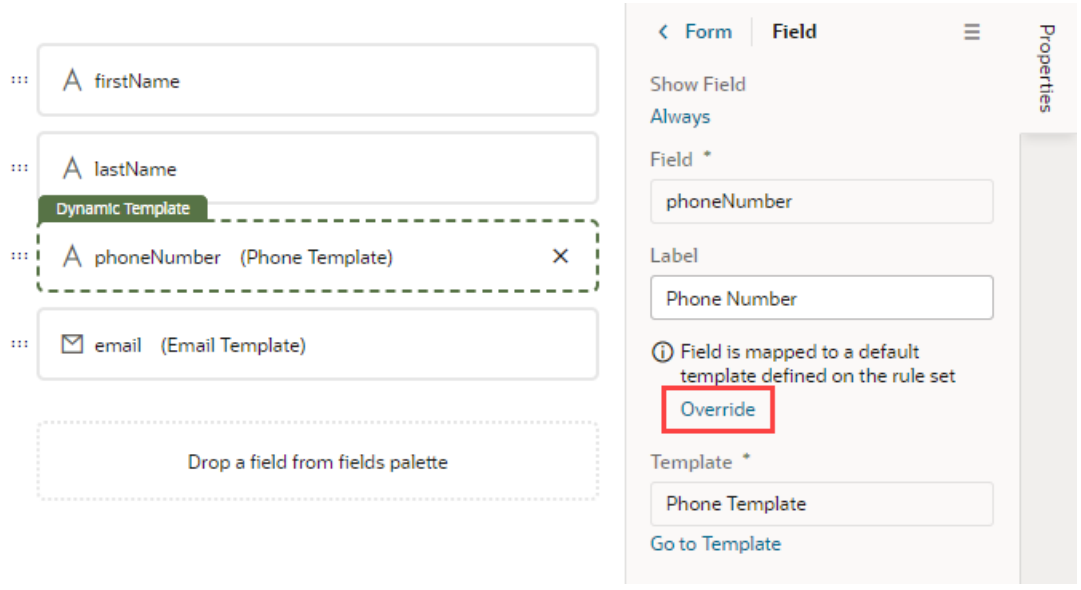


2. Select the field you want to apply a template to.
3. In the field's Properties pane, select a template from the Template drop-down list.

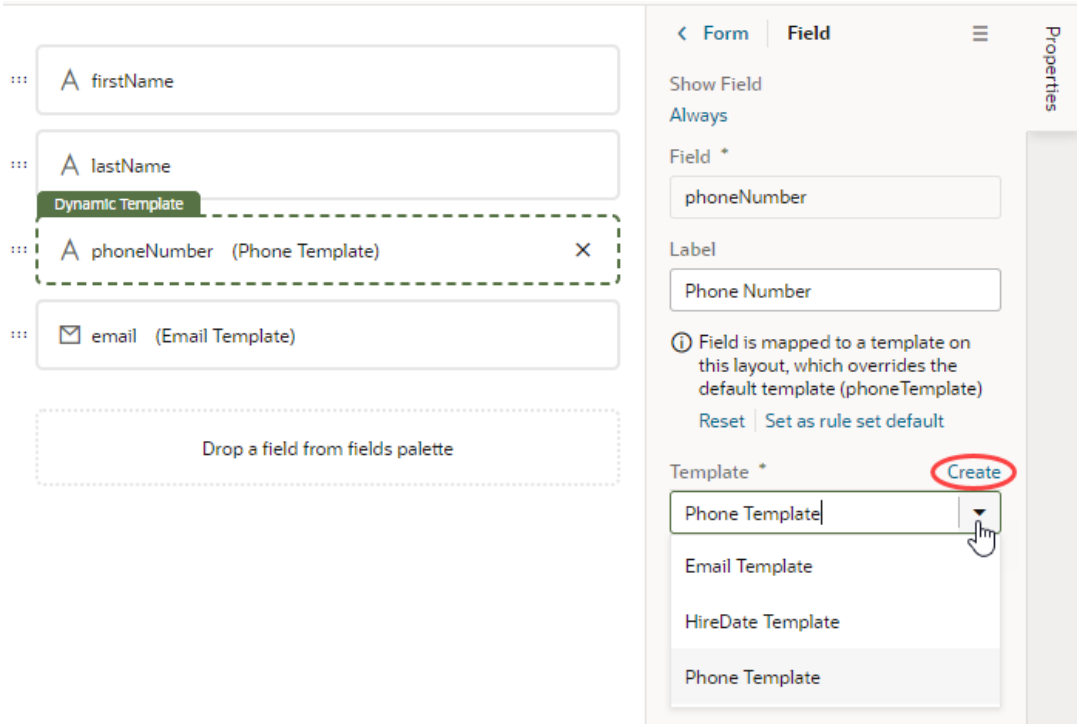
   If no template has been applied to the field, you can select a template in the list (as in this image):



If the field already has a field template applied to it (you'll see the template name next to the field name), you'll see a notification in the Properties pane that a default template has been defined for the field:



If you want to change the field's template only in the current layout, click **Override** in the Properties pane, then select another template from the drop-down list. Or, click **Create** to define a new template. If you don't want a template applied to the field, select None in the drop-down list.



If you want the changed template applied to the field in every layout in the rule set, click **Set as rule set default** in the Properties pane. You can click **Reset** at any time to re-apply the default template.

Start an Action Chain from a Field

You can start an action chain when an event occurs in a field by adding a component event to the field in a field template.

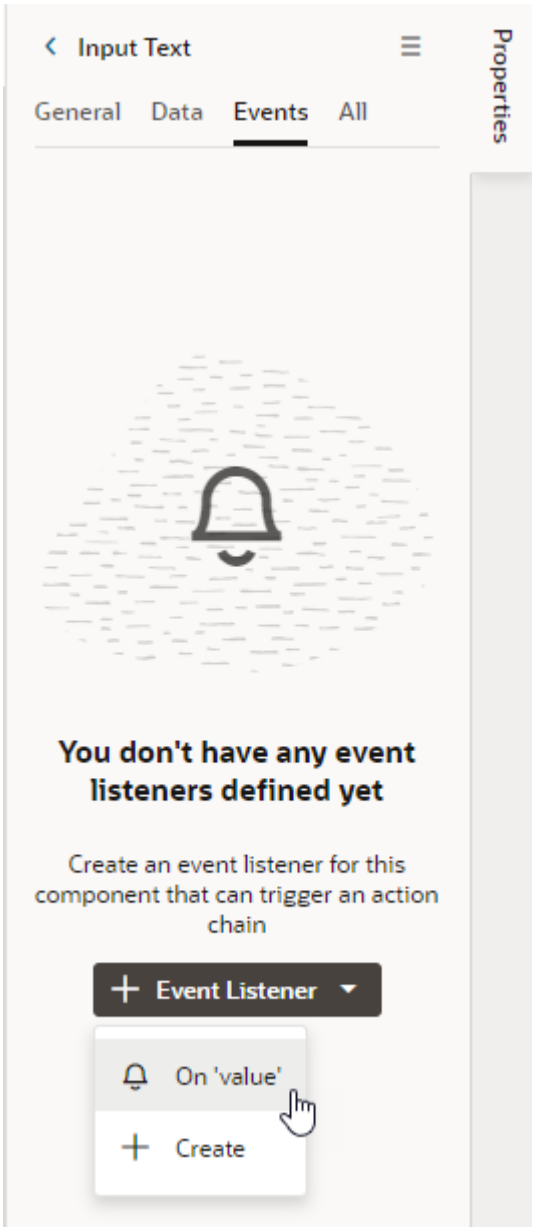
For example, you might want to display some additional details or options when someone changes the value in one of your form fields. You can add an event that's triggered when the value changes, and start an action chain that retrieves the data and displays it in your page. The Quick Start option in the component's Properties pane can help you quickly create the event, event listener, and action chain. You can also use the event to start action chains that are already defined in the layout.

When creating an action chain, you can use variables and constants defined in your layout, and create new ones if you need them.

To start an action chain from a field:

1. Open the **Templates** editor for your layout and click the field template you want to edit.

   The field template opens in the Template editor.
2. Select the text field, then open the **Events** tab in the Properties pane.
3. Click **+ Event Listener** and select the **On 'value'** option.

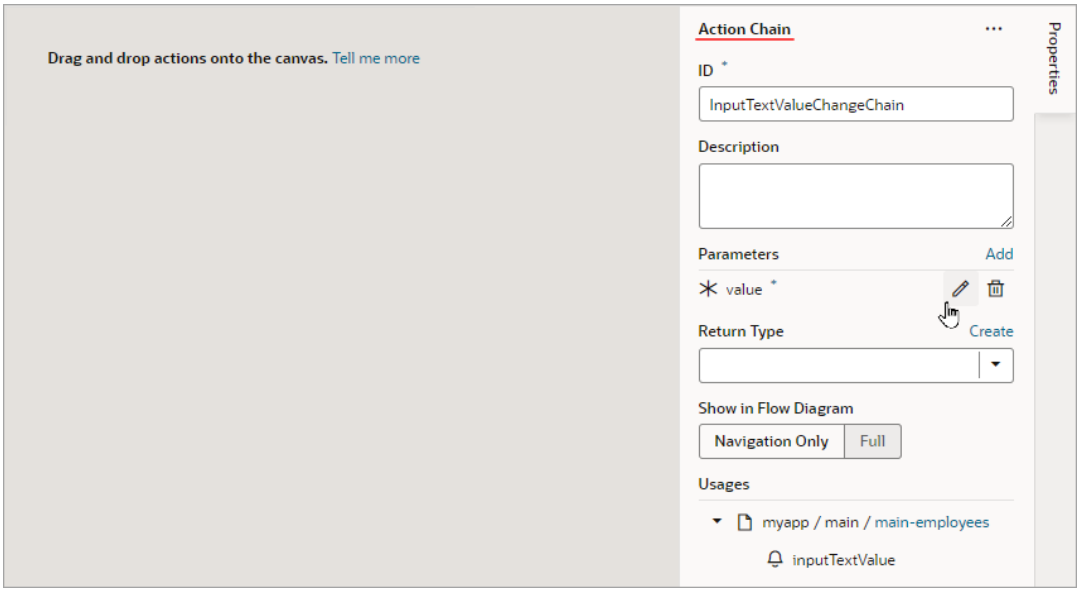


The event listener for the `On 'value'` event is a quick start that suggests an event for your component. In the case of a text field, the suggested event is `value`, which is triggered when the text field's value changes, for example, when someone types in the field. If you don't want to use the suggested event, you can select **+ Create** in the drop-down list and select a different event. Make sure the event name starts with a lowercase letter, though camel case is allowed. Hyphens are not supported.

When you select the Quick Start option:

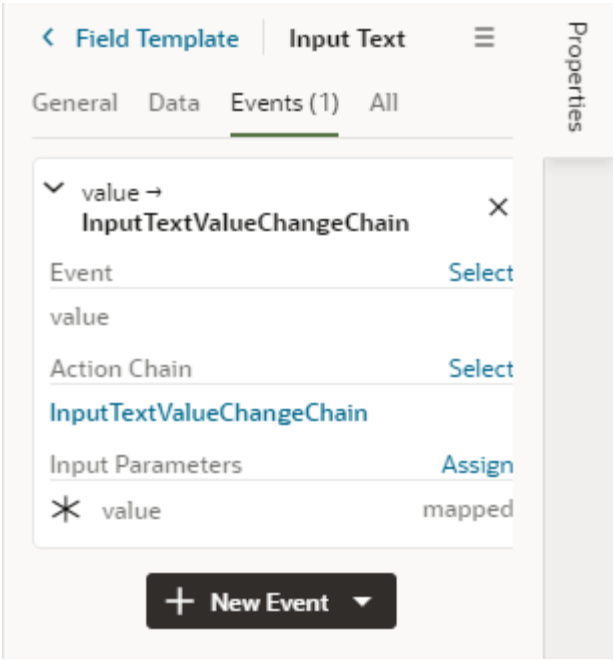
- an event is defined for the text field
- a new action chain is created
- an event listener is created that will trigger the new action chain when the event occurs
- you are navigated to the new action chain in the Action Chain editor.

4. In the Action Chain editor, define the action chain's properties in the Properties pane. You can edit the default ID, add a description, and configure the action chain's input parameters and return type.



5. Create the action chain by adding actions from the palette. Depending on the actions you add, you might also need to create variables used in the action chain or layout and define other properties for the actions.

If you navigate back to the template editor, you'll see the event details in the field's Events tab in the Properties pane. You can add more action chains that will be triggered by the same event, or you can add different events to the same component.



Control How a Form Layout is Rendered

You can apply a form template to layouts to control how it's rendered, including which fields you want the layout to contain and how they are displayed in the layout.

For example, you might have a page that uses a dynamic form (not table) to display a detail view that includes sales figures, and you want the form to always display a Rating Gauge component, regardless of which fields are defined in the layout. You could create a 'Sales' form template that includes the Rating Gauge component, and then apply the template to the form. You can re-use the template in other dynamic forms in the layout's rule sets, but templates can't be shared between rule sets in different layouts.

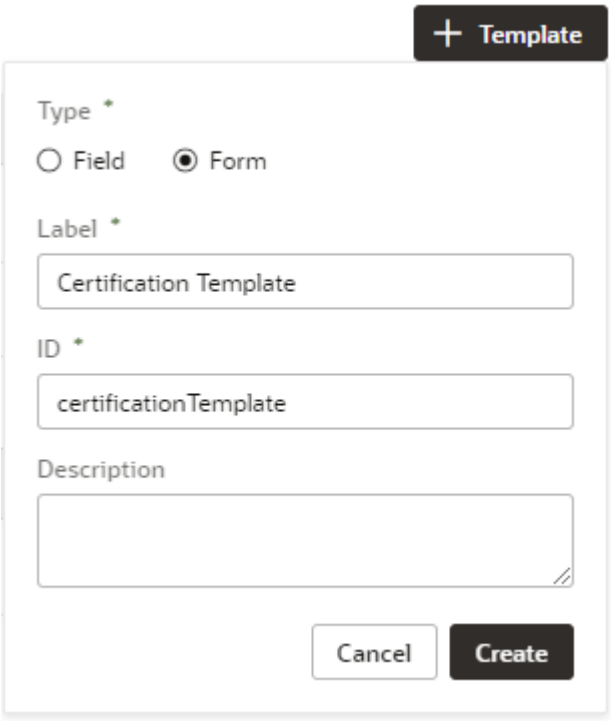
To create a form template for a dynamic form:

1. Open the layout's **Templates** tab.

   The Templates tab displays a list of field and form templates that are already defined for the artifact.

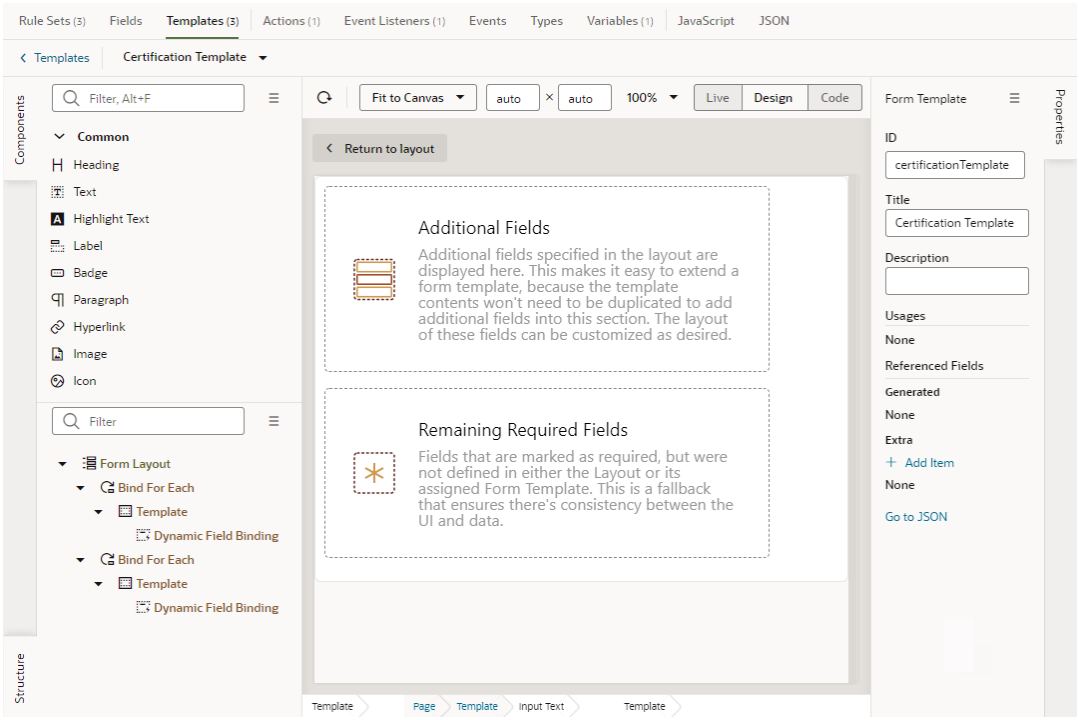
   > **Tip:**
   >
   > If you've already created a form layout and want to create a template for it, you can open the layout in the rule set editor, click Use Template, and then select Create a New Template in the Use Layout Template window.

2. Click **+ Template**. Select **Form**, specify the Label (the ID is generated for you), and click **Create**.



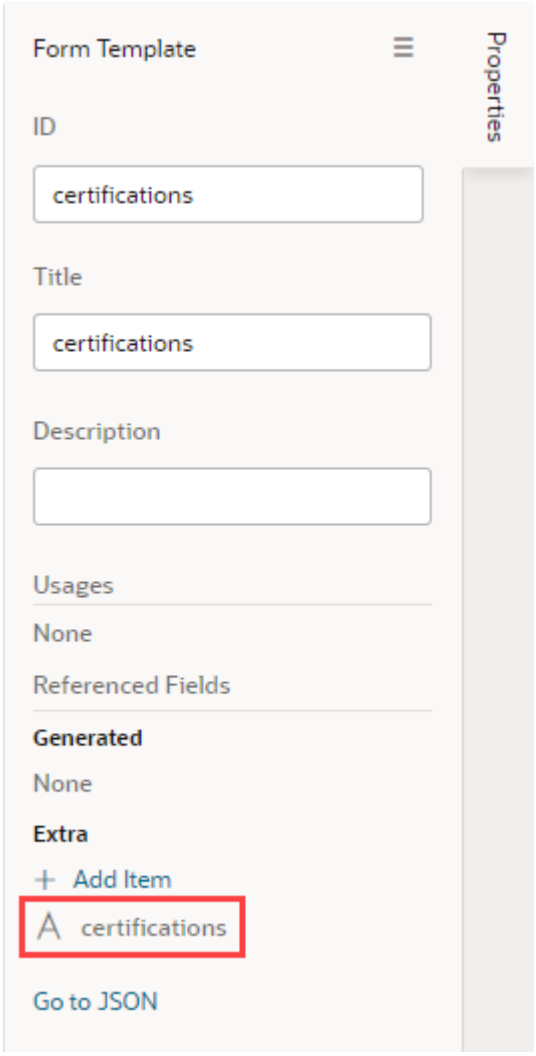
The form template opens in the template editor, which contains a Components palette, Structure view, canvas, and a Properties pane.

In this image, you can see that the canvas has two read-only template sections that are generated automatically: `Additional Fields` and `Remaining Required Fields`. These fields are used to display the data and display names for the fields defined in the layout. These template fields render all the fields in the layout, so you don't need to modify the template each time you change a layout.

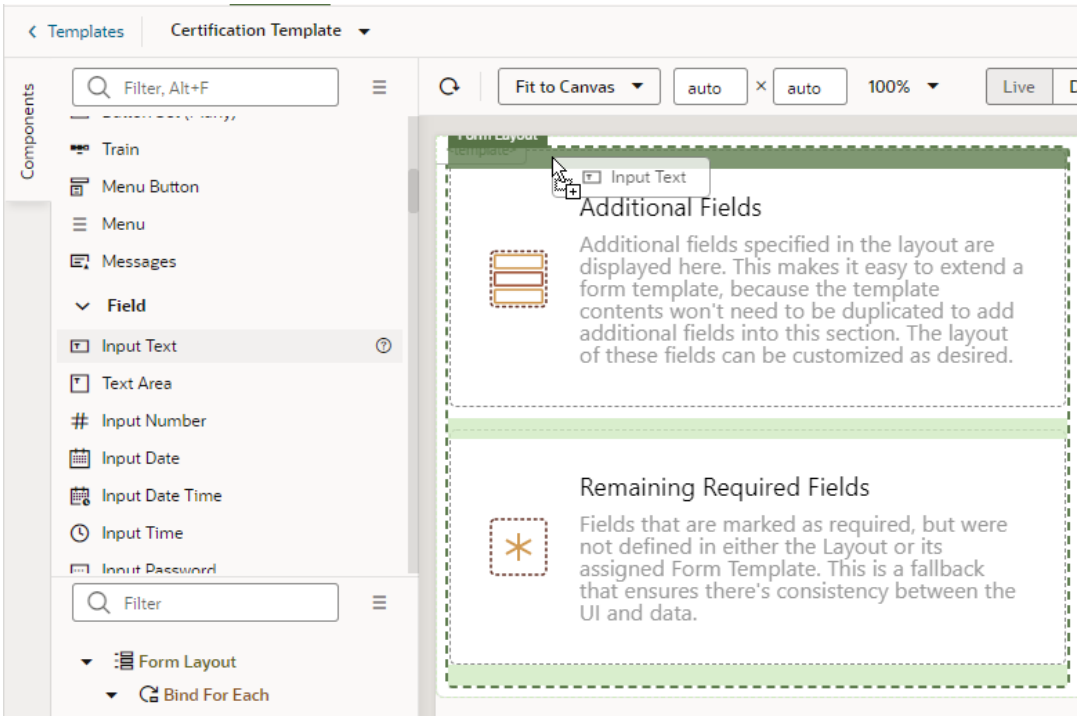


3. In the Form Template's Properties pane, click **+ Add item** under Extra and select a field. Extra fields are defined in the template, not in the layout. You will want to add a field as an Extra field when you know it will be needed by the layout. Each field you add in the Extra section can be used in the form and will always be available when the template is applied.

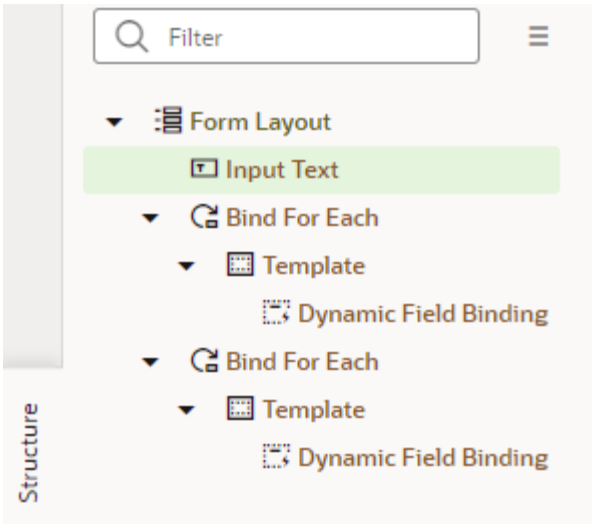
   Each Extra field must be mapped to a component if you want it to appear in the form. This image shows the Properties pane after the `certifications` field has been added to the template as an Extra.



4. Drag the component you want to add from the Components palette and position it in the Structure view or on the canvas.

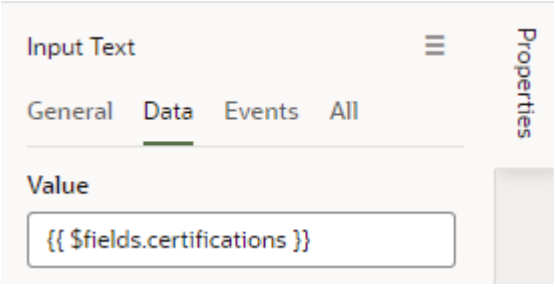


You can add components above and below the read-only template fields, but not within them. In the Structure view of this template, you can see an Input Text component that was positioned above the Additional Fields template in the Form Layout.



5. While the component is selected on the canvas or in the Structure view, open the component's Data tab in the Properties pane and bind the component to the Extra reference field.

   To help you select the reference field, you can click fx to open the Expression Editor, or to open the variables picker.



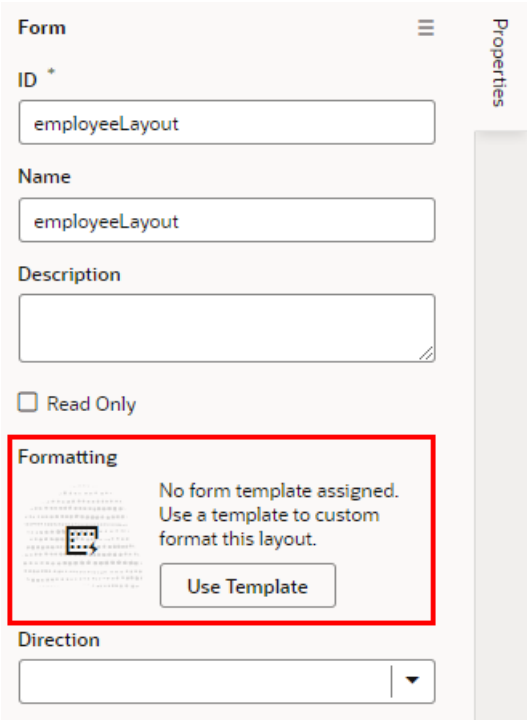
To write efficient expressions that handle situations where a referenced field might not be available or the field's value could be null, see How Do I Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null?

After you've added the components and fields to your form template, you can apply the template when you edit a layout in the Rule Sets editor.

Apply a Template to a Form

To apply a form template to a dynamic form:

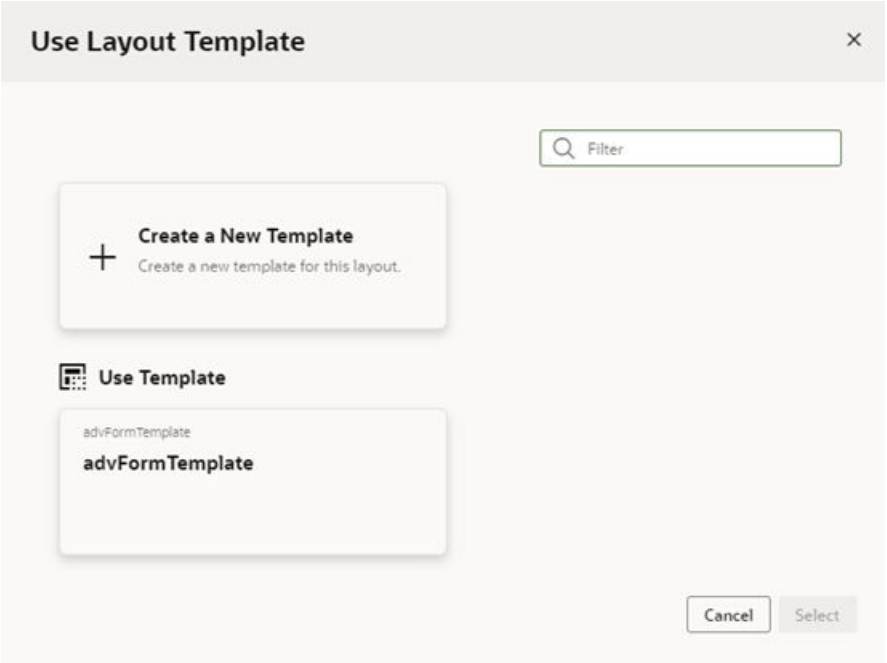
1. In the rule set editor, click the name of the layout you want to work on.
2. While the form is selected, click **Use Template** in the Properties pane.



If a template has already been applied to the form and you want to switch to a different one (or remove it), click **Select** in the Properties pane.

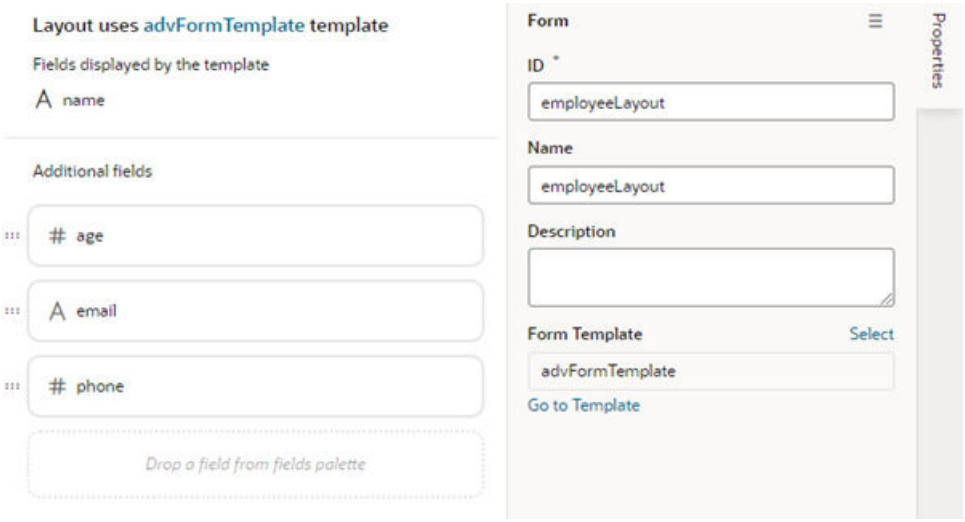
3. Select the template you want to apply in the Use Layout Template window. Click **Select**.

   The Use Layout Template window lists the available templates you can apply to your form layout.



You can select **Create a New Template** to create a new form template.

When a template is applied to a form layout, the template name and the fields defined in the template are displayed above the list of fields in the layout. In this image of the layout editor, you can see the header displays the name of the template applied to the form layout (`advFormTemplate`) and the fields defined by the template (`name`).



If the template displays a field you don't want to appear in your form, you'll need to select a different template, or click **Select** in the Properties pane and select **No Template** in the Use Layout Template window to remove the template.

Add and Group Fields in Dynamic Form Layouts

When creating a layout for a dynamic form, you can group the form's fields so that they are displayed together as a single entity in the layout.

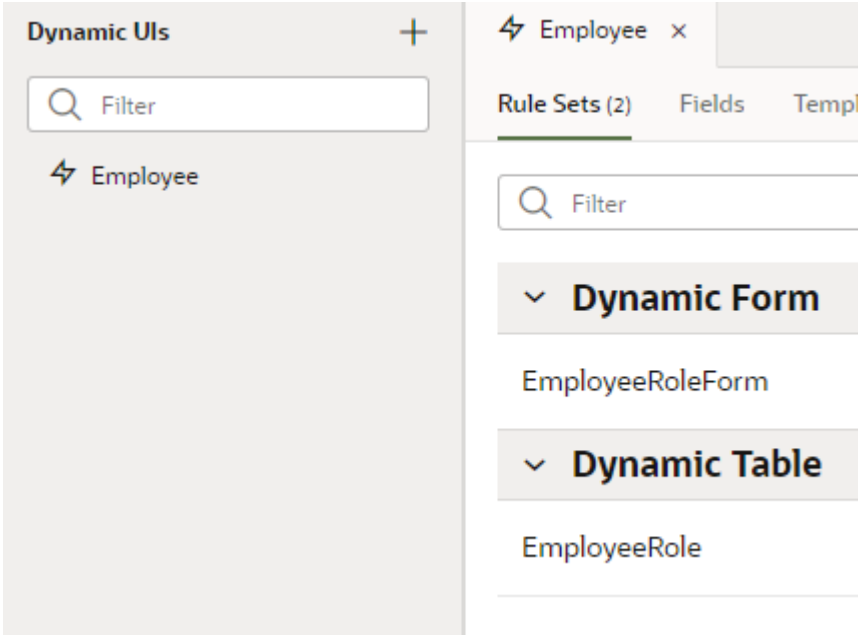
For example, you might create an address group that contains the name, address, city, state, country, and postal code fields. You can then apply conditions to the group that control when the group is displayed. A group also makes it easy to add several fields to a different layout in one step, rather than adding them individually.

You can define properties for a group (for example, a group label) and for individual fields in a group (for example, to specify column spans for fields to create complex dynamic form layouts).

To group fields in a dynamic form layout:

1. Open the Rule Sets tab for the dynamic form you want to work with. You can select a standalone dynamic form or one that's part of a dynamic container.

   To do this, select a layout in the Navigator's **Layouts** tab, then find the dynamic form in the **Rule Sets** tab (as shown here); click the form on a rendered page in the canvas area; or select the form from the Properties pane.

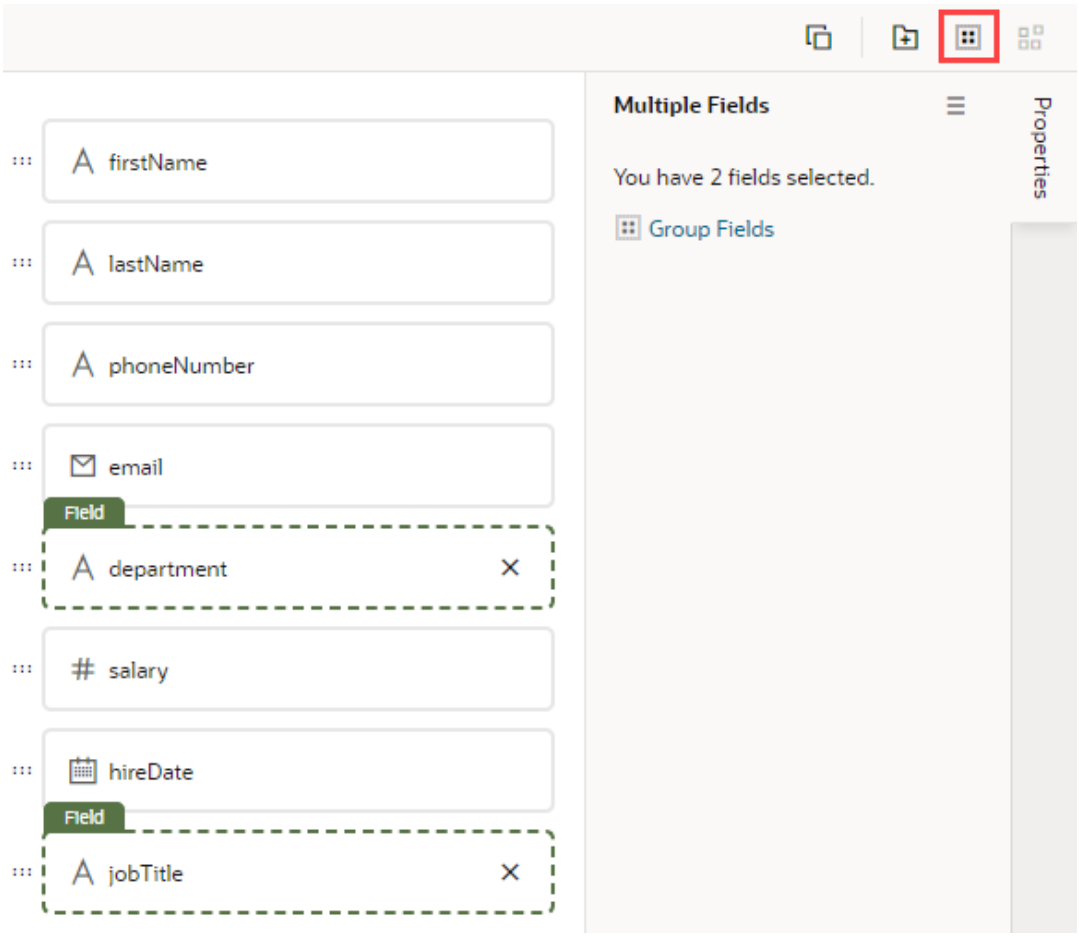


2. Open the form you want to edit.

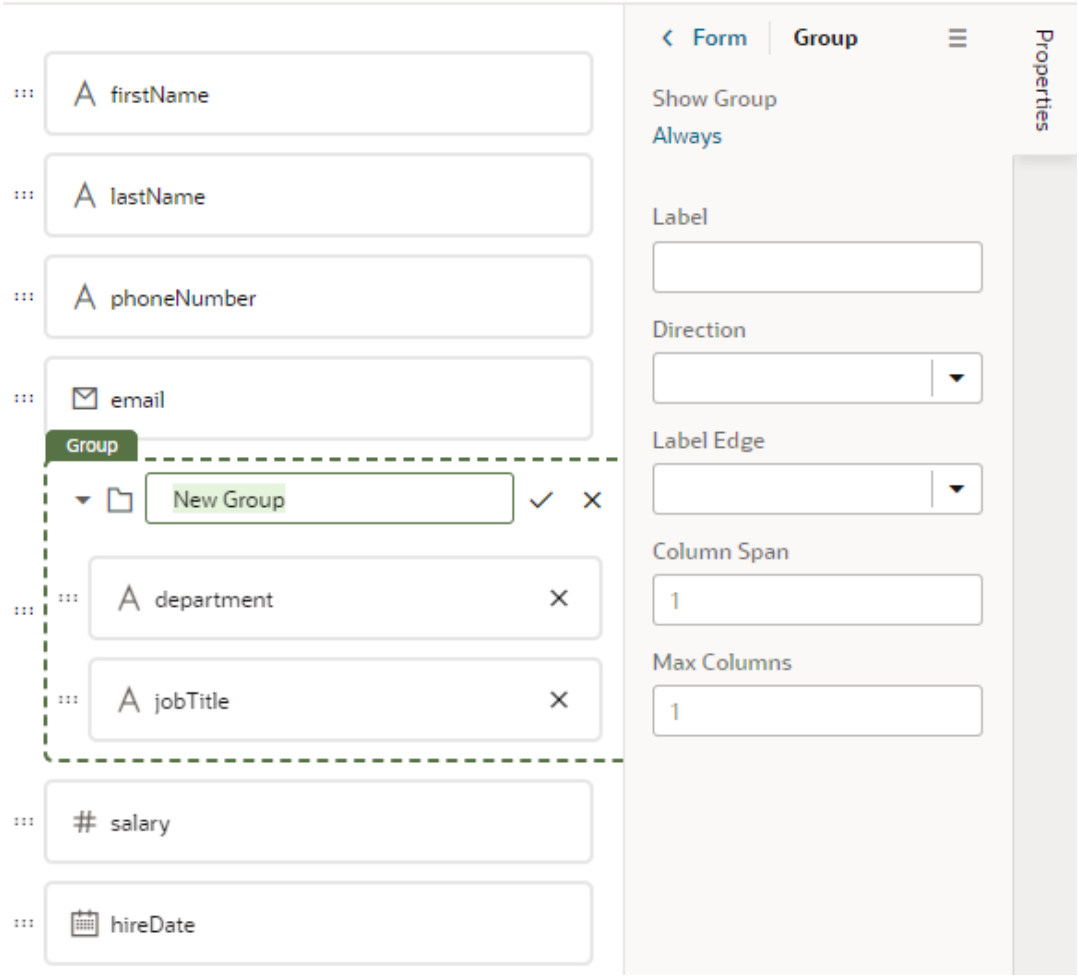
   Use the icon in the toolbar if you want to duplicate the current layout.

3. In the layout diagram, select the fields that you want to group together, either by holding down the CMD key (on macOS) or the Ctrl key (on Windows).

4. Click **Group Fields** in the Properties pane, or in the toolbar.



The selected fields are grouped under a new folder in the layout diagram:



5. Enter a name for the new group. Click ✓ to save the group name.
6. Optionally, use the Properties pane to set properties for the group. You might even click the Always link to set conditions that determine when the group is displayed in a layout. The default setting is to always display the group.

After a group is created, you can still use the handles for fields to drag them into and out of a group.

Use Business Rules With Your Rule Sets

After you create a rule set for a dynamic table or form, you can use business rules to change the value of specific field properties, across multiple dynamic form and tables, when the rule's conditions are met.

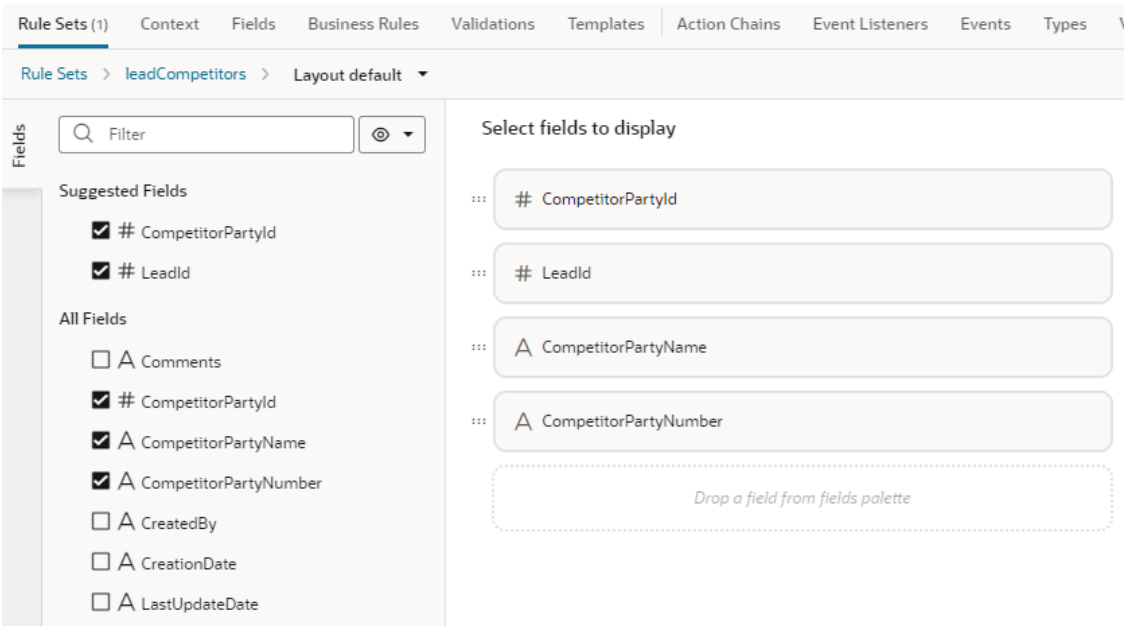
When you create a dynamic form or table, a rule set is created automatically, and by default the fields in the rule set's default layout are included in business rules. When a field is included in business rules, you can override some of its properties using rules and conditions. It is up to you if you want to use business rules to override field properties. You can choose to use rule sets or business rules, or use both. You can also choose to exclude rule set layouts from business rules.

Compared to layouts in rule sets, business rules don't provide the fine grain control over all the details of a form or table layout, but they do allow you to quickly create rules that can override some field properties. Like rule sets, the overrides in business rules are applied at runtime, but

they take precedence over field properties defined in rule set layouts. For more details, see What Are Business Rules?

> **Note:**
>
> Like rule sets, business rules cannot override properties set in the Layout's Fields tab.

Let's take a look at an example of what happens when the fields in a rule set are included in business rules. When you create a form to display an object's fields (the leadCompetitors object, in this example), the default layout for the form's rule set might look something like this:



When you switch to the Business Rules tab for leadCompetitors, the four fields in the default layout are displayed in the list. The "Show only fields used by layouts" filter is selected, so only the four fields are shown:

If you create another form, with a different rule set, or modify the default layout to add more fields, the list of fields is updated to include the new fields. In this example, two more fields (Last Update Date and Threat Level Code), added in the new form, are now included in the list:

You could now write one business rule that would affect both forms, for example, to hide a field, even though the forms use different rule sets.

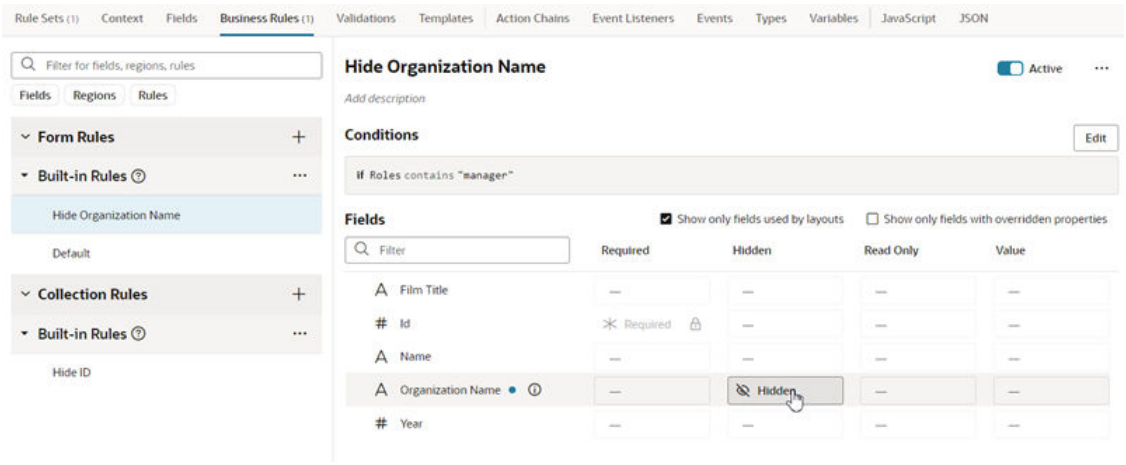
What Are Business Rules?

Business rules allow you to override the appearance and behavior of fields in dynamic tables and forms, provided that the specified conditions in a rule are met at runtime.

> **Note:**
>
> If you're looking for information on business rules as they pertain to business objects, see Create Rules for Business Objects.

> **Note:**
>
> In some applications, where you see fields grouped into *regions* in the business rules editor, a region is simply a dynamic table or form.

For example, suppose you had a data object called Person, that was used in two dynamic forms. Now suppose you want to hide the **Organization Name** field when the user is a manager, as only HR specialists should see this data. With business rules, you can create a single rule that defines the condition, then overrides the setting for the Hidden property to ensure that **Organization Name** is hidden when the user is a manager. With this one action, the **Organization Name** field will be hidden for every form that includes that field. When you set a property in a business rule, a form rule in this case, you are setting it at the object level, so it can be applied to every occurrence of that field, in each form using that data object.



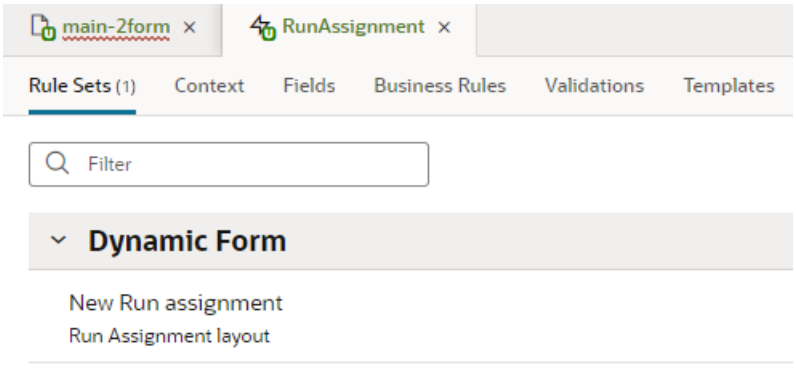
There are two types of rules: *form rules*, which control the properties of fields in dynamic forms, and *collection rules*, which determine the columns displayed in tables. When you create a rule, it's listed under *built-in rules*.

Include and Exclude Rule Sets in Business Rules

When you create a dynamic form or table, you can choose if you want to allow it to be governed by business rules. Rule sets are included in business rules by default, so you need to explicitly exclude a rule set if you don't want business rules to affect the component.

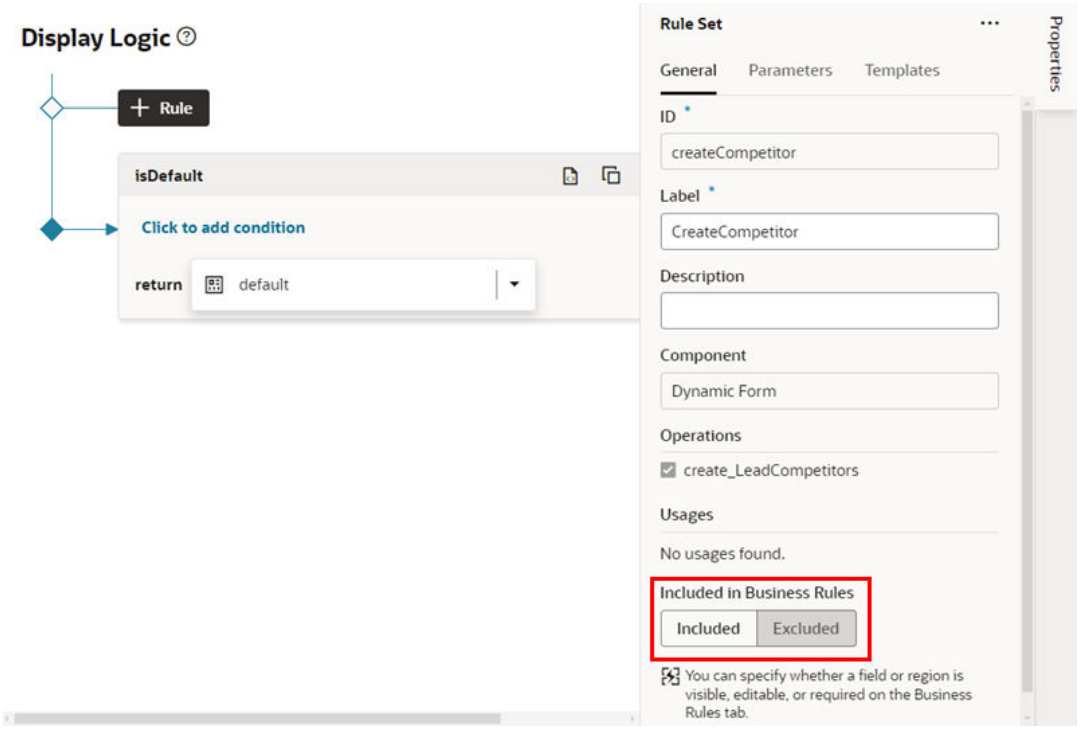
To exclude a rule set from business rules:

1. Open the component's rule set.
   In the Rule Sets tab, click the rule set to open it in the editor.



You can also open the rule set from the Page Designer by clicking Go to Rule Set in the component's Properties pane when the component is selected on the canvas.

2. Select **Excluded** in the rule set's Properties pane if you don't want it included in business rules.



Work with Business Rules

When configuring regions and fields, the main components of a business rule are *conditions* and *properties.* Conditions determine the circumstances under which the rule is applied, while properties override the value set for a given field or region by Oracle (or by your functional administrator).

When validating fields with business rules, you configure *messages* that are displayed on the page when the rule's conditions are met.

Create a Rule for Forms

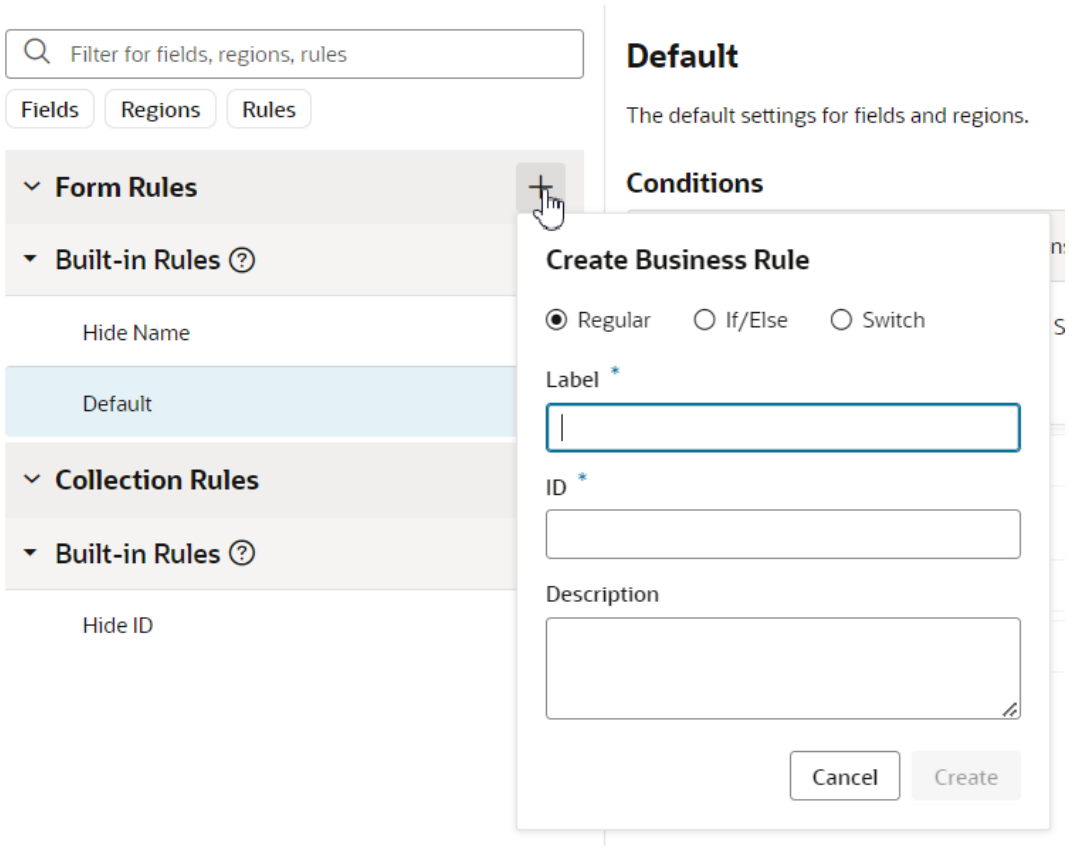
Use a form rule to override properties for fields in a dynamic form, assuming that certain conditions are met at runtime.

To create a form rule:

1. Open the Business Rules tab in the Layout.

   If you are in the Page Designer, you can select the dynamic form on the page, and then click **Go to Rule Set** in the Properties pane to open the Rule Sets tab, then open the Business Rules tab.

2. In the business rules editor, click + next to Form Rules to open the Create Business Rule dialog box.



   Rather than starting from scratch, you can duplicate an existing collection rule and use it as the basis for a new rule. To do this, right-click the rule, and then click **Duplicate**.

3. Select **Regular** in the dialog box.

   You can choose If/else or Switch in the dialog box to add conditions to when rules are evaluated. See Add an If/Else Rule and Add a Switch Rule

4. Enter a label, id, and description for the rule.

   The id is generated automatically based on the label you enter, but you can modify the id if you wish. The description field is not required, but it can be helpful when you later try to understand what a rule is doing, especially when there are many rules.

5. Click **Create**

Your new rule is added to the top of the list of built-in rules under Form Rules.

6. Create the rule's condition.

   See Set Conditions for a Rule.

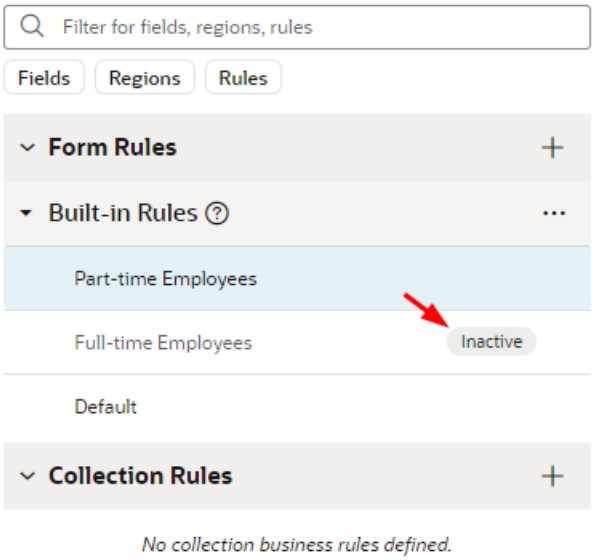
7. Specify the overrides for the field properties.

   See Override Field Properties in a Form.

To delete a rule, right-click the rule in the list to open the popup menu, and then click **Delete**.

Rules are evaluated in order, from bottom to top. As you create more rules, make sure you position each one in the order you want them evaluated, using the grab handles (six dots) beside the rules to drag-and-drop them to new positions. You can find out more about how rules are evaluated in Understand What Will Be Shown at Runtime.

If you decide you don't want to include a rule in the evaluation order, select the rule, then use the **Active** toggle switch in the upper right corner to deactivate it. (You can also right-click a rule and deactivate and activate it in the popup menu.) This enables you to still keep the rule so you can re-activate it later. You can tell at a glance if a rule is inactive because a little badge appears next to it, like this:



Inactive rules are not included in the rule evaluation process.

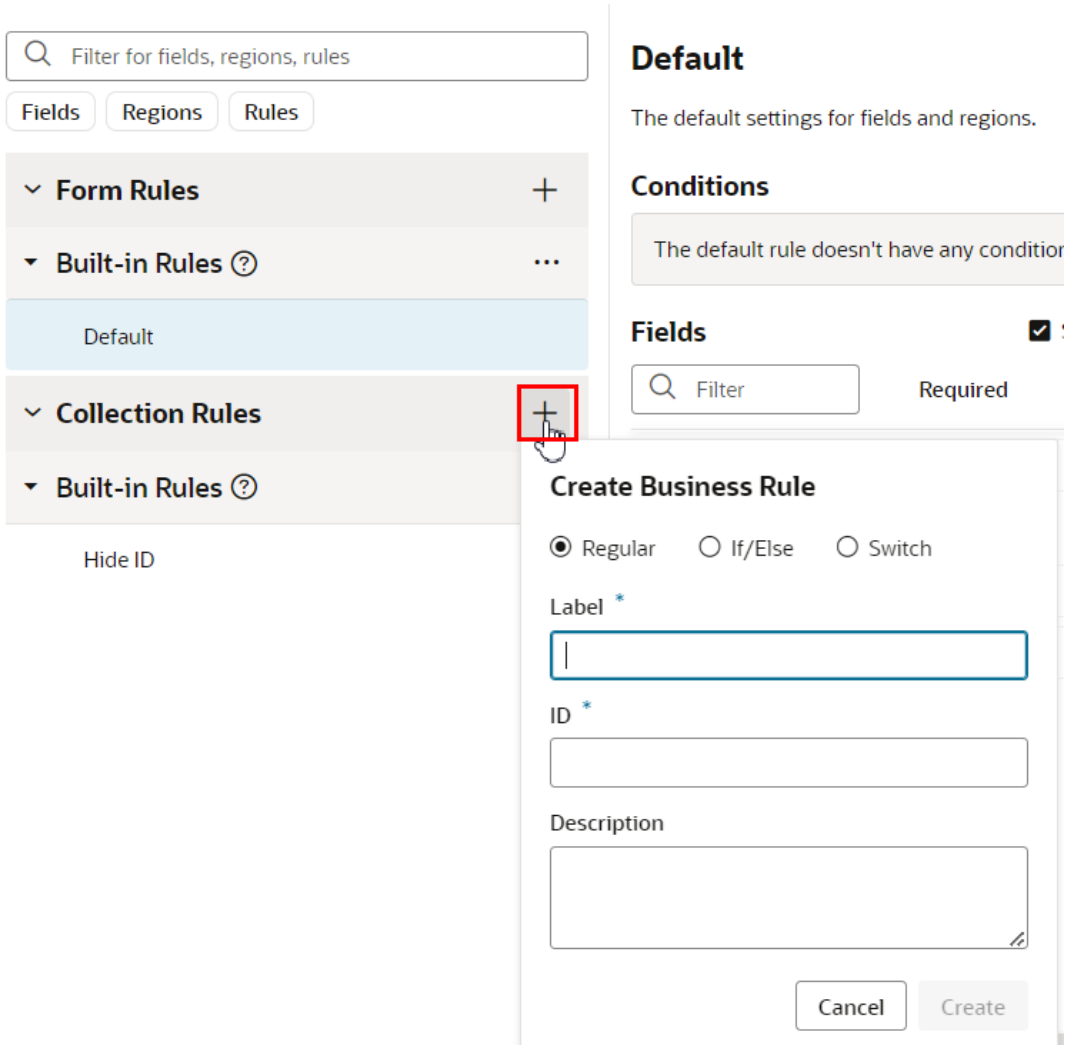
> **Note:**
>
> You can deactivate all the rules at once by clicking the three dots next to the Built-in Rules heading, then clicking **Deactivate All**. This can be useful when debugging a page, allowing you to see the page with no rules applied. Use **Activate All** to reinstate all the rules at once, or use the **Active** toggle to selectively activate them as you work through your debugging process.

Create a Rule for Tables

Use a collection rule to hide or display columns in a dynamic table, assuming that certain conditions are met at runtime.

To create a collection rule:

1. Open the page you want to configure and select the table in the Properties pane.
2. In the Properties pane, click **Configure Business Rule** to open the Business Rules tab.
3. In the Business Rules tab, click + next to Collection Rules to open the Create Business Rule dialog box.



Rather than starting from scratch, you can duplicate an existing collection rule and use it as the basis for a new rule. To do this, right-click the rule, and then click **Duplicate**.

4. Select **Regular** in the dialog box.

   You can choose If/else or Switch in the dialog box to add conditions to when rules are evaluated. See Add an If/Else Rule and Add a Switch Rule

5. Enter a label, id, and description for the rule.

The id is generated automatically based on the label you enter, but you can modify the id if you wish. The description field is not required, but it can be helpful when you later try to understand what a rule is doing, especially when there are many rules.

6. Click **Create**

   Your new rule is added to the top of the list of built-in rules under Collection Rules.

7. Click **Edit**, and then create a condition for the rule.

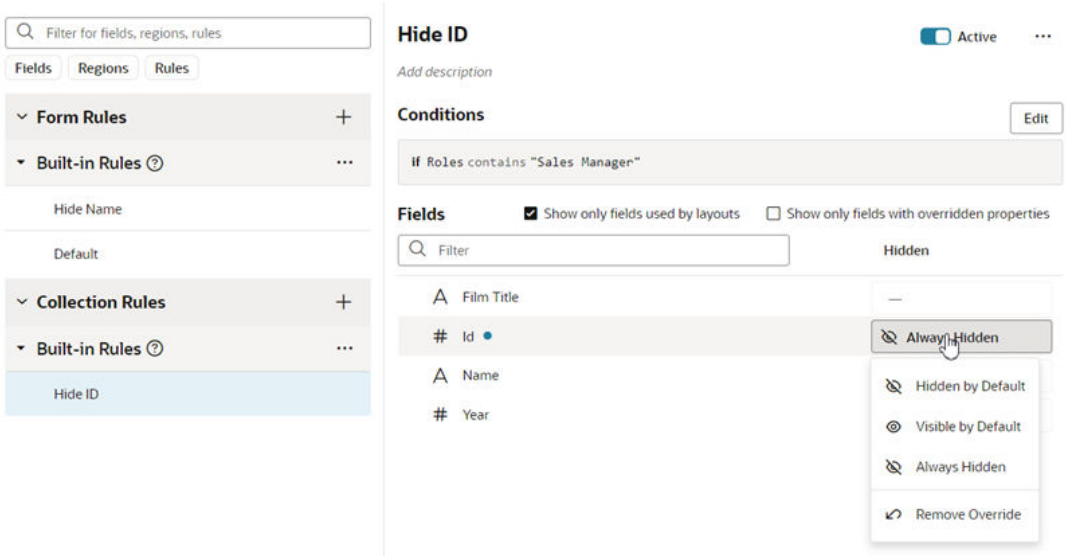
   See Set Conditions for a Rule.

8. In the Fields area, locate the field corresponding to the column you want to configure.

   In a table, each field corresponds to a table column. This means that if a dynamic table is configured to display four fields, the table will have four columns.

9. Click the dash—or an existing value—for the field's Hidden property, and select a new value for the property in the dropdown list.

   In a collection rule, the only property you can override is the Hidden property. You can set the property to Hidden by Default, Visible by Default, or Always Hidden:



   In a collection rule, you can set the Hidden property for a field even if it's not a table. However, the Hidden property in a collection rule is only applied to table columns. (The property does not affect fields in forms.)

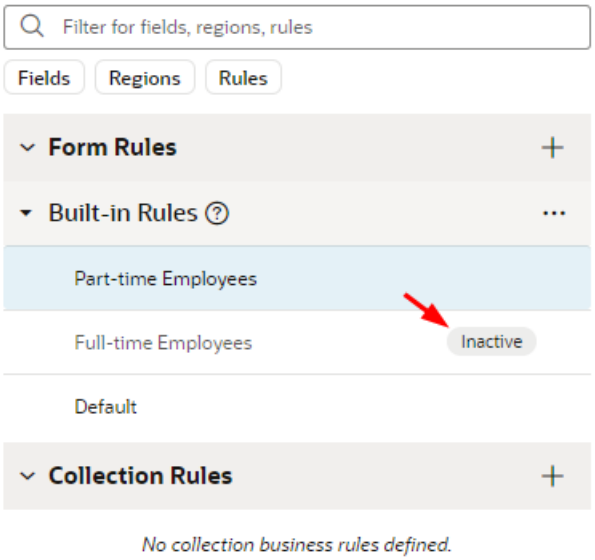
   If you only want to see the fields used in your table, select **Show only fields used by layouts**. If you don't select this, the list contains all the object's fields. You can select **Show only fields with overridden properties** to limit the list of fields to those that have been modified by a rule.

To delete a rule, right-click the rule in the list to open the popup menu, and then click **Delete**.

Rules are evaluated in order, from bottom to top. As you create more rules, make sure you position each one in the order you want them evaluated, using the grab handles (six dots) beside the rules to drag-and-drop them to new positions. You can find out more about how rules are evaluated in Understand What Will Be Shown at Runtime.

If you decide you don’t want to include a rule in the evaluation order, select the rule, then use the **Active** toggle switch in the upper right corner to deactivate it. (You can also right-click a rule and deactivate and activate it in the popup menu.) This enables you to still keep the rule

so you can re-activate it later. You can tell at a glance if a rule is inactive because a little badge appears next to it, like this:



Inactive rules are not included in the rule evaluation process.

> **Note:**
>
> You can deactivate all the rules at once by clicking the three dots next to the Built-in Rules heading, then clicking **Deactivate All**. This can be useful when debugging a page, allowing you to see the page with no rules applied. Use **Activate All** to reinstate all the rules at once, or use the **Active** toggle to selectively activate them as you work through your debugging process.

Set Conditions for a Rule

You determine when a rule is applied by defining a *condition*. For example, you might create a rule that is applied only when the user is in Canada and has the Manager role.

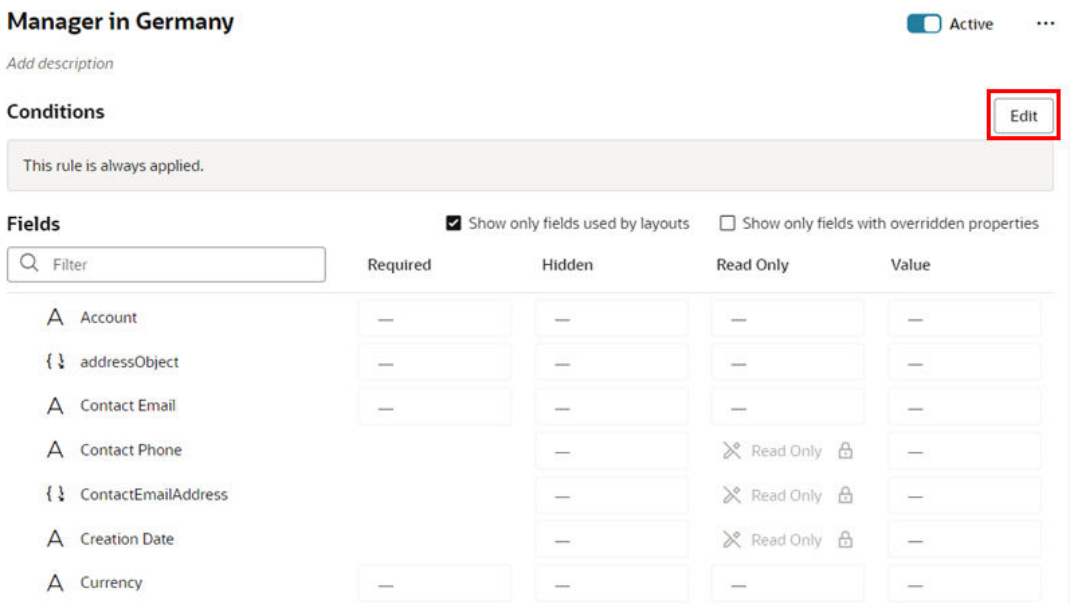
There are two ways to define the rule's conditions. The first way is to use the basic condition builder to create conditions by selecting criteria and values. This way should be enough to define most conditions. However, if you need to create more complex conditions, and you are comfortable working with expressions, you can click **Use Advanced Expression** to open the visual expression editor. For more about using the expression editor, see Build Advanced Expressions.

Rules define overrides that are applied to properties only when the rule's conditions are satisfied at runtime. For conditions that use criteria in the User context, like User Authenticated (`$user.isAuthenticated`) or Roles (`$user.roles`), the condition is met if the logged-in user satisfies the condition.

> **Note:**
>
> When using the Roles criterion (`$user.role`) in a condition, the Value drop-down lists the available Oracle Applications Cloud job and abstract roles. (The drop-down will not list any duty roles. If you want to specify a duty role, you can manually type the duty role name in the Value field.)

To create a condition for a rule:

1. Select the rule you want to edit, then click **Edit** to open the condition builder:



2. In the condition builder, define one or more conditions.

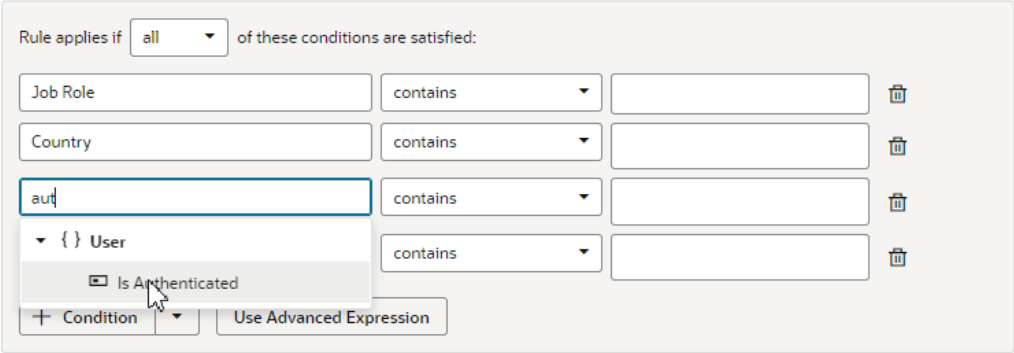
   When possible, Visual Builder starts you off by pre-populating the conditions with criteria used in the page, but you can use different criteria in your condition. The criteria, and their available values, depend on the object and what has been set for the page.

   To define a condition:

   a. Select a criterion from the dropdown menu.

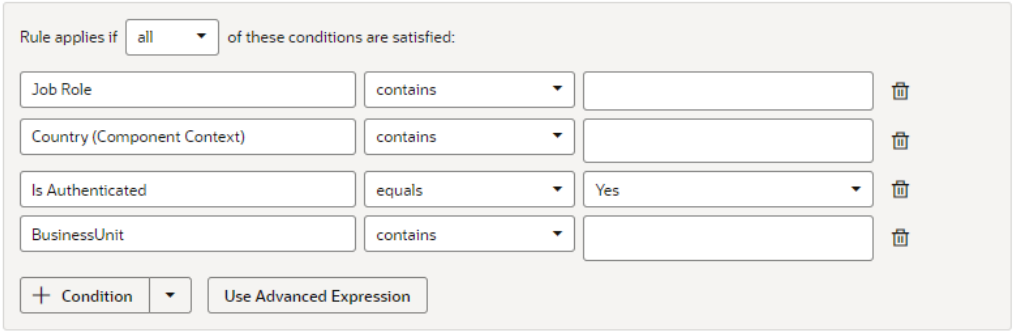
      You can choose any of the listed criterion in your conditions, but you cannot change the list of available criteria to add your own.

      If you know what criteria and values you're looking for, you can try typing in the field to filter the list:



b. Select an operator.

The options available in the operator dropdown menu are pre-defined based upon the criterion's type. So for a criterion like User Authenticated, the operator menu only has `equals`, and the only available values in the value menu are `yes` and `no`:

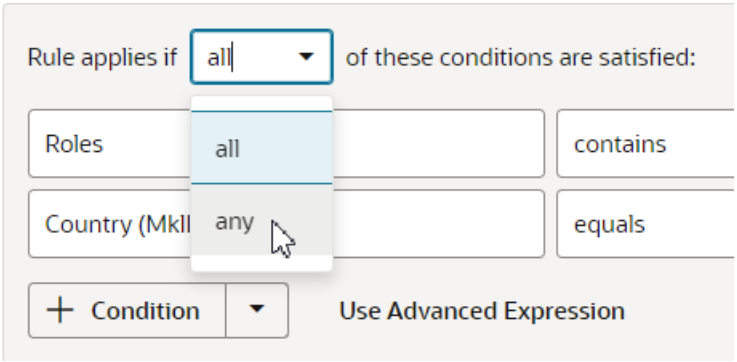


c. Specify the values.

Select a value from the dropdown list, or type a value in the field. Depending on the criterion, you might be able to choose multiple values.

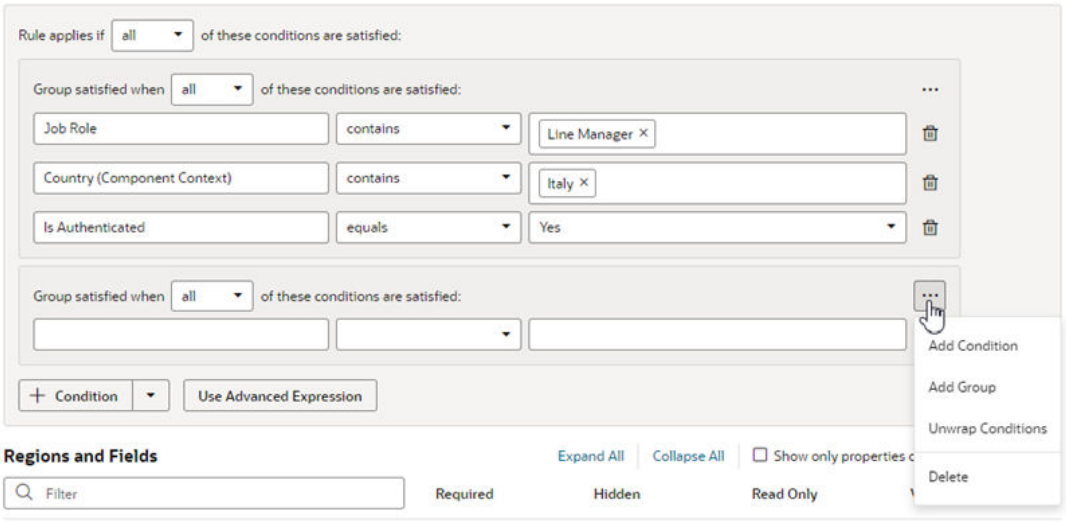
3. To add another condition, click **+ Condition**, and then define the new condition. To remove a condition, click .

When you have more than one condition, the rule is applied only when ALL the conditions are true (by default). However, you can change this to ANY, which means only one of the conditions has to be true in order for the rule's overrides to be applied:



4. To create a group, click the arrow next to **+ Condition**, and then select **Add Group** in the dropdown menu. Grouping conditions lets you create more complex conditions for the business rule.

   When you click Add Group, your existing conditions and groups are combined into one group, and a new group containing a condition is added.



   Each group has a menu with options for managing the group:

   - **Add Condition**. Adds a new empty condition to the group.
   - **Add Group**. Creates a new group within the group.
   - **Unwrap Conditions**. Ungroups the conditions, and the empty group is deleted.
   - **Delete**. Deletes the group and all the conditions in the group.

   After creating a group, you can add and define conditions in the group, and set the logic (`any`, `all`) for the groups.

5. Click **Done** to close the condition builder.

   Conditions are always saved automatically, so you don't need to worry about explicitly saving your changes when editing a condition.

To change the conditions for an existing rule (not a new one), click the rule in the list, then click **Edit** above the Conditions pane.

If you're creating a new rule, the next step is to set *properties* for the fields.

Create Condition Using a Field's Initial Value

When choosing a field in the condition builder, you might see fields listed under both **Field Values** and under **Initial Field Values**.

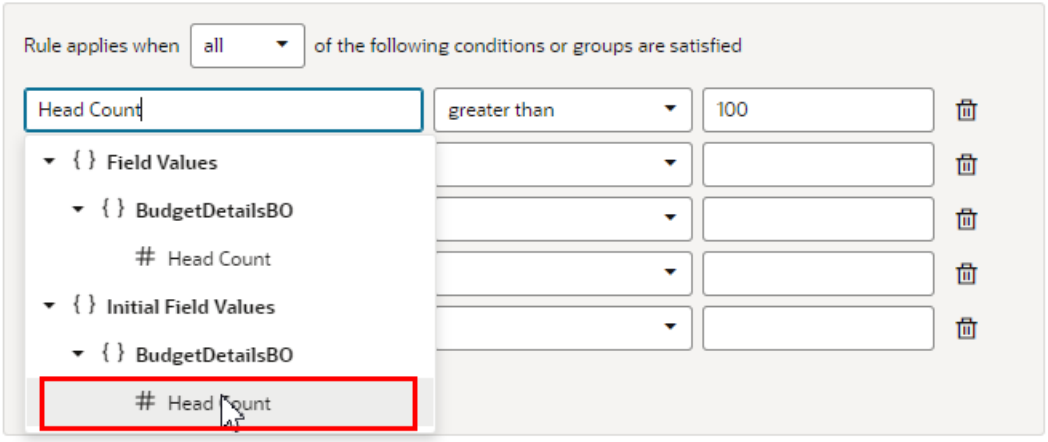
When creating a condition, you might want to use the value retrieved from the data source when the page loads—that is, the Initial Field Value. Once the page loads, the Initial Field Value doesn't change. The Field Value, on the other hand, is the value displayed or cached in a page, which may already have been modified by a rule or user. For example, say the value for the Head Count field retrieved from the data source (the Initial Field Value) is 50. There might be some rule that sets the field's value (the Field Value) to 60. The Field Value is displayed in the Head Count field in the form. The user may then change the Head Count field

to 70 in the form, so the Field Value for Head Count is now 70. The Head Count Initial Field Value, however, is unaffected by changes made by rules or users, so it is still 50.

Let's look at how to add a rule so that users cannot edit the Head Count value in a form if the field's initial value is greater than 100.

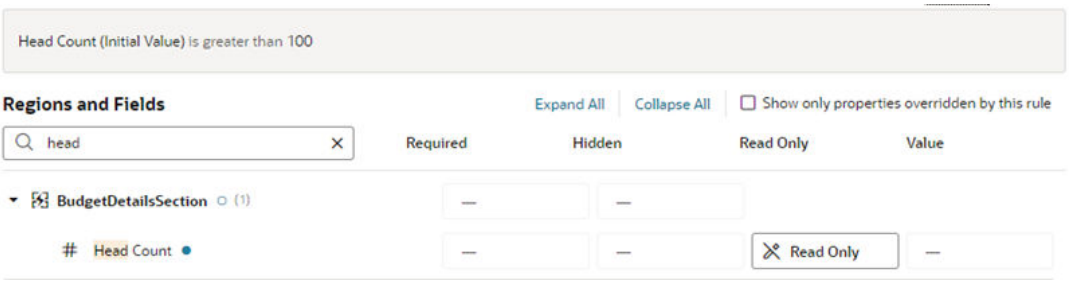
1. In the business rules editor, create a new form rule.
2. Create a condition where the initial value of Head Count is greater than 100.

   In the criterion dropdown list, be sure to select Head Count under **Initial Field Values** (as opposed to under **Field Values**):



3. Set the Read Only property of the Head Count field to **Read Only**.

   To help you locate where the Head Count field is used, you can type 'head' in the filter field to filter the list of fields:



In this example, the Head Count field in the form might display a value greater than 100 and *still* be editable, because a rule is modifying the Head Count value. If you wish, you could create a validation rule that displays a message when the Head Count is over 100, telling the user the Head Count field value must be 100 or less. In the condition for the validation rule, make sure the rule is evaluating the Field Value (not the Initial Field Value) for the Head Count field.

Build Advanced Expressions

When creating conditions for business rules, you may find your conditions are more complex than you can achieve using the basic condition builder. If this occurs, you can build your own custom expressions to suit your needs.

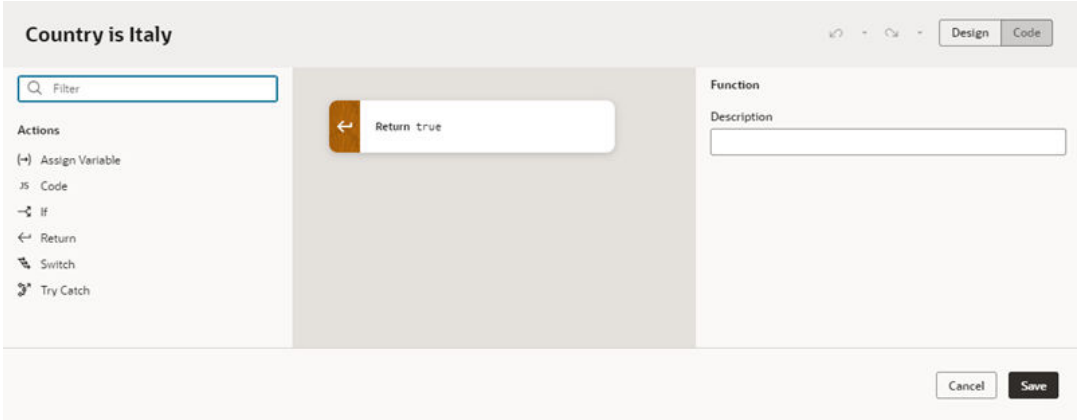
> **Note:**
>
> If you use advanced expressions in the condition builder, you will no longer be able to use the basic condition builder. If you have already defined some conditions in the basic condition builder, they will be displayed as actions in the Advanced Expression editor.

To create an advanced expression:

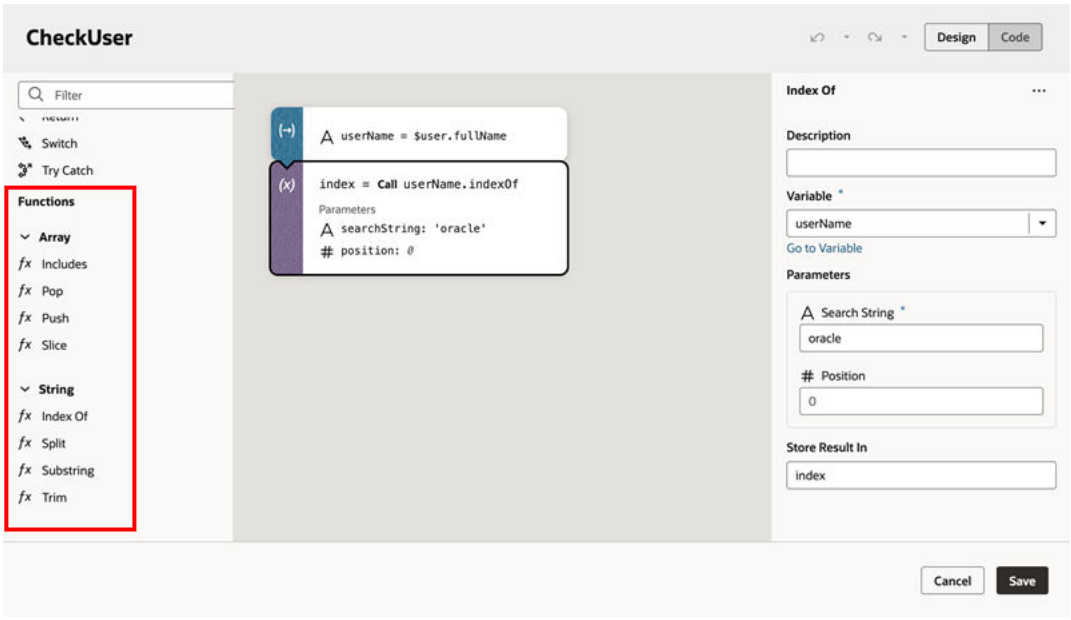
1. Click **Use Advanced Expression** in the condition builder.

   The Advanced Expression editor has a Design mode, which you use to visually create an expression, and a Code mode where you can type the expression. You toggle between modes using the **Design** and **Code** buttons in the header.

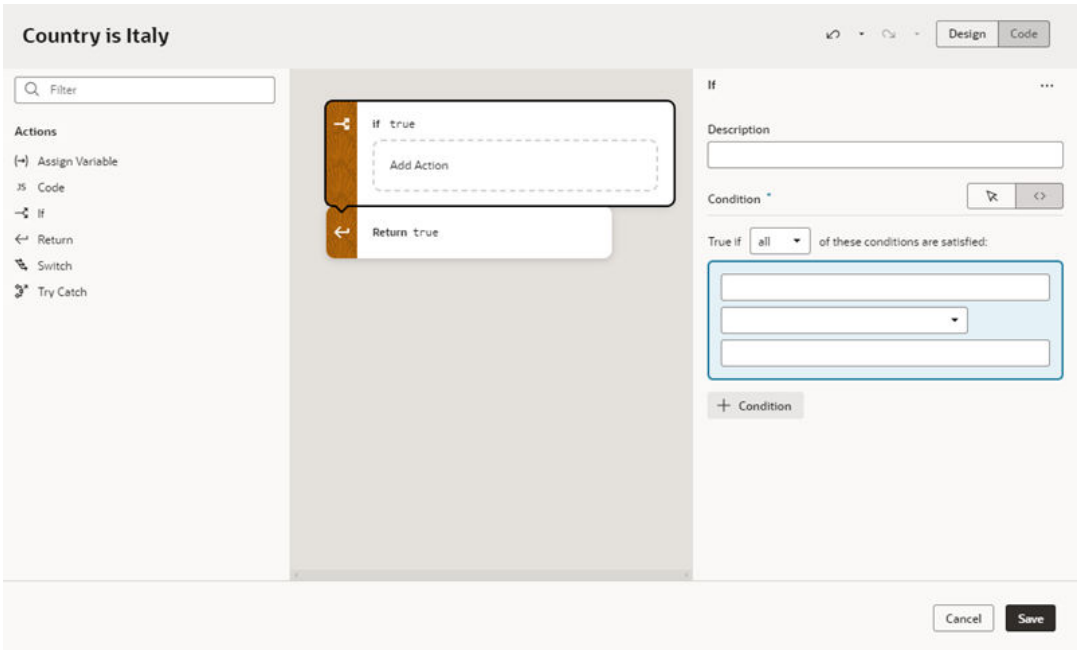
   This image below shows Design mode, which has an Actions palette on the left, a canvas in the middle, and a Properties pane on the right:



2. Drag an action from the palette and position it on the canvas.



Let's take a look at what happens when we drag an If action onto the canvas:



When the If action element is selected on the canvas, the Properties pane contains a Description field and a condition builder.

3. Define the action in the Properties pane and on the canvas.

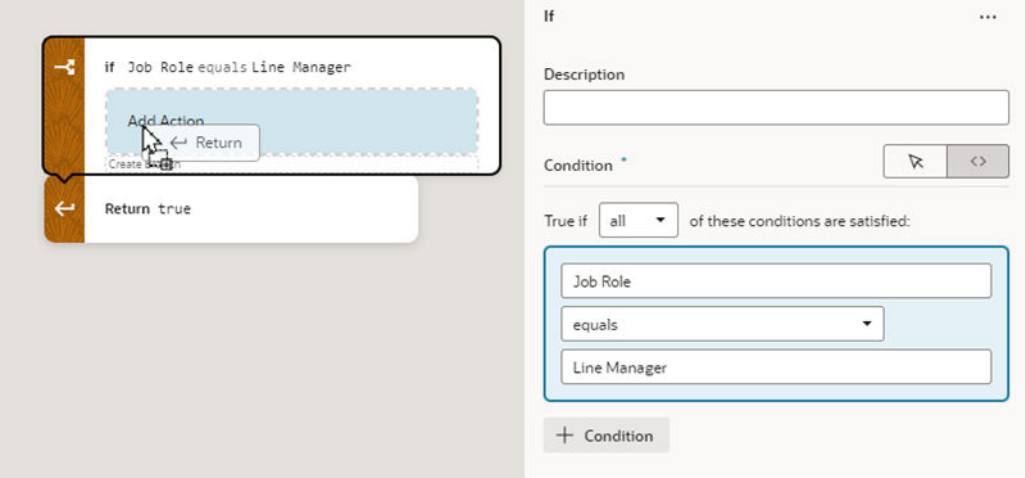
   a. Edit the action's properties in the Properties pane. The properties displayed in the Properties pane vary according to the action.

      For an If action, the Properties pane has a condition builder for selecting the condition's criterion, operator, and value. Click **+ Condition** in the Properties pane to add a condition. Right-clicking a condition opens an options menu for adding, moving, and deleting the condition. You can also reposition a condition using drag-and-drop in the condition builder.

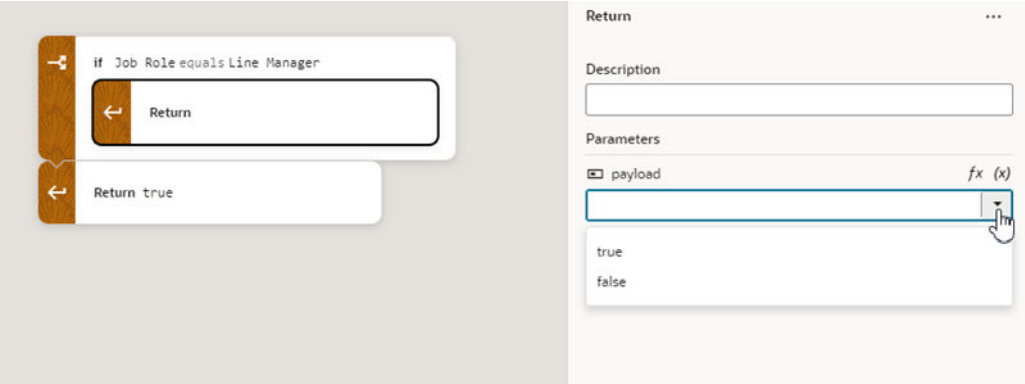
You can use the condition builder's toggle buttons to switch between the Design and Code views:

b. On the canvas, drag actions from the palette (for example, Return, Assign Variable, Try Catch) into the action's Add Action drop target.

For an If action, you need to define what happens when the condition is true. If the condition is true, and you want to return from the advanced expression, drag a Return action into the action's drop target:



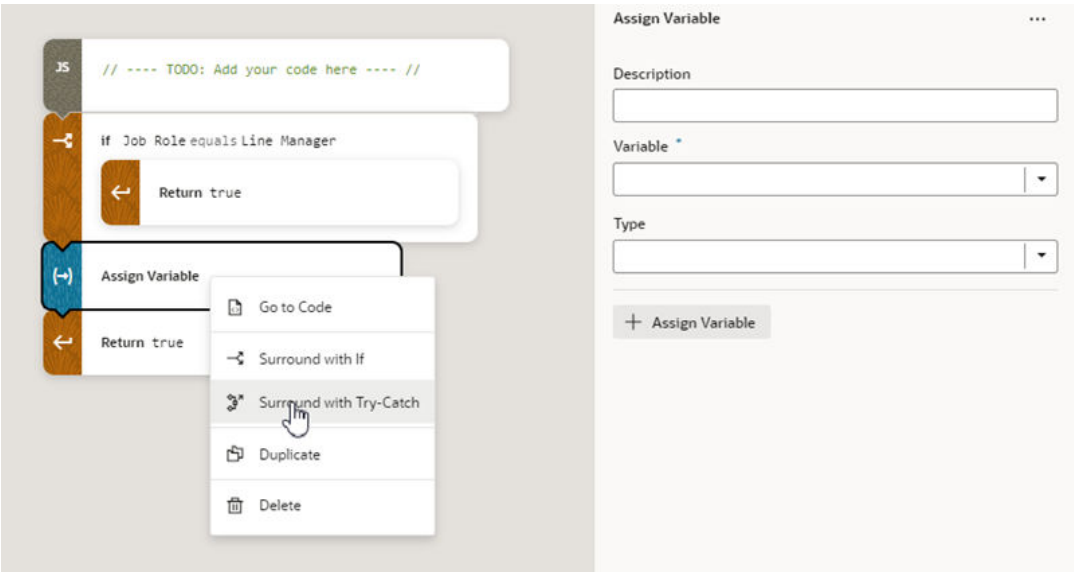
Now select the Return action on the canvas, and then select 'true' in the Payload dropdown list in the Properties pane:



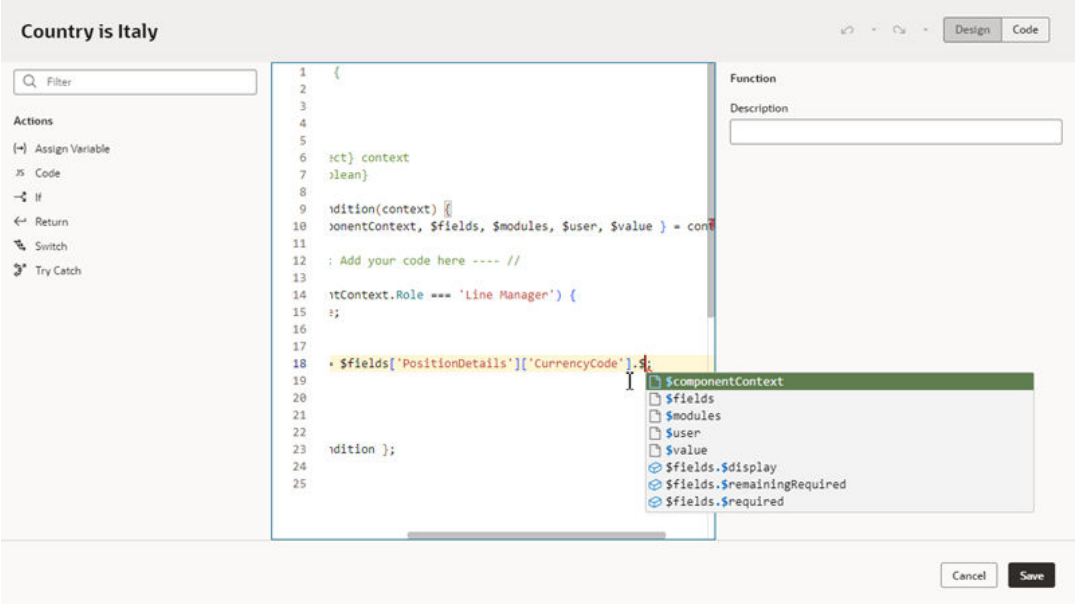
4. Repeat steps 2 and 3 to add more actions to the expression.

   As you add more actions, you can reorganize the order by grabbing the front of the action element on the canvas, and then moving it into a new position:

   Right-clicking an action on the canvas opens a popup menu with some convenient shortcuts, including deleting the action and switching to Code view:



At any point, if you're comfortable typing expressions you can click **Code** in the header to open the editor's Code mode. In Code mode you can type your expression directly, and take advantage of the code completion in the editor:



5. Confirm that the payloads for your Return actions are correct.

   When a Return action is selected on the canvas, you can choose a payload in the dropdown list in the Properties pane.

6. When you're finished building your expression, click **Save**.

When you look at your rule, the condition looks like this if it's created using the expression editor:

Override Field Properties in a Form

For each field on the page, you can set some *properties* to override the values set by lower-level rules, .

These properties include:

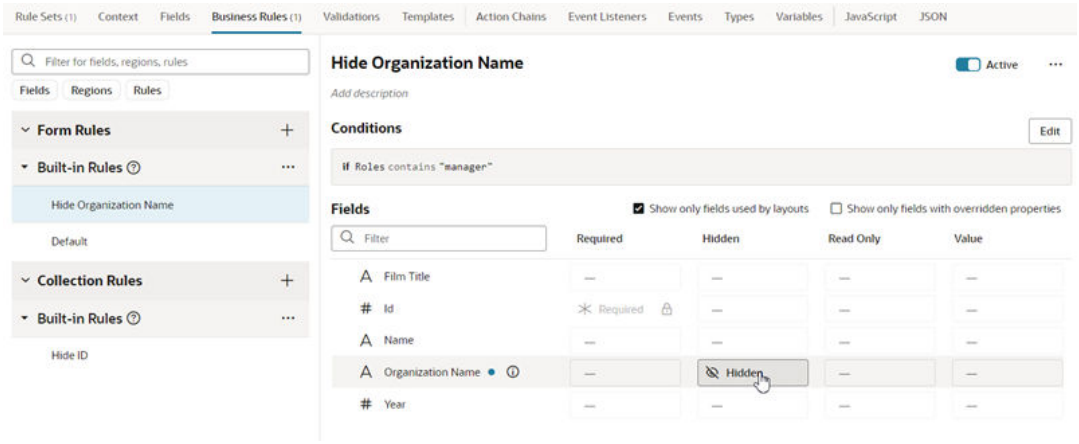
- Required – Make required or optional
- Hidden – Visible or hidden
- Read Only – Editable or read only
- Value – Static or expression

If more than one rule impacts a given field, it can be tricky to sort out what is finally displayed at runtime to each user group; Understand What Will Be Shown at Runtime can help you.

To set a property on a field:

1. In the business rules editor, select the form rule you want to configure.
2. Locate the field you want to modify.
3. Click the dash—or an existing value—in the appropriate column, and select a new value for the property in the dropdown list.

   For example, you can modify a field's Hidden property by selecting Visible or Hidden in the dropdown:



   VB automatically saves your work for you, so there's no need to do so explicitly.

   If you change your mind after setting a property, use the **Remove Override** option to remove your setting and restore the property to its original value.

   Descriptive Flexfields (DFF) and Extensible Flexfield (EFF) sections (or contexts), when shown, are treated like any other fields; that is, you can set the Required, Hidden, Read Only, and Value properties for them as needed.

   Remember that the Default rule is always active, which establishes the out-of-the-box behavior. All other built-in are essentially *overriding* what is specified in the Default rule. If none of the other rules evaluate to true, then the only overrides applied are those defined in the Default rule.

> **Note:**
>
> A field marked as hidden can still be rendered as visible at runtime. For example, suppose when a page's rules are evaluated, a given field is both required, which means it must have a value, but also hidden. In addition, the field does not have a default value, which means that the user must supply a value explicitly. But how can the user supply a value if the field is hidden? To protect users from encountering this quandary, VB will show the field even though it is marked as hidden, thus allowing users to enter a required value and move on from the page.

Set a Default Value for a Field

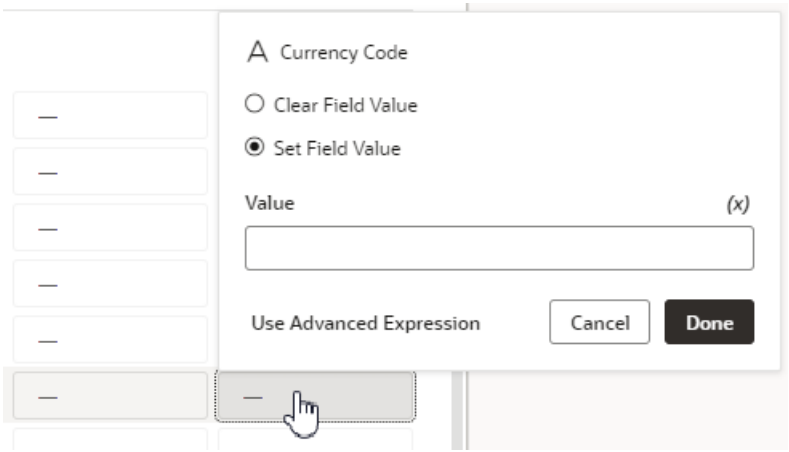
Use the *Value* property in a business rule to set a default value for a field.

Suppose you have a form in which the user can enter a currency in the Currency Code field. If the user doesn't enter a value—that is, if the field is empty at runtime—you can populate the field with a default value. You can accomplish this by using the Value property to set the field to, say, "euro", if the user is in Italy. When the user updates the form and clicks Save, the value "euro" is saved in the field, unless the user changes it to something else.

If the business rule sets the field to Read Only, of course, the user won't be able to change it. But if the field is editable, the user can change the value simply by updating the field.

To set a value for a field:

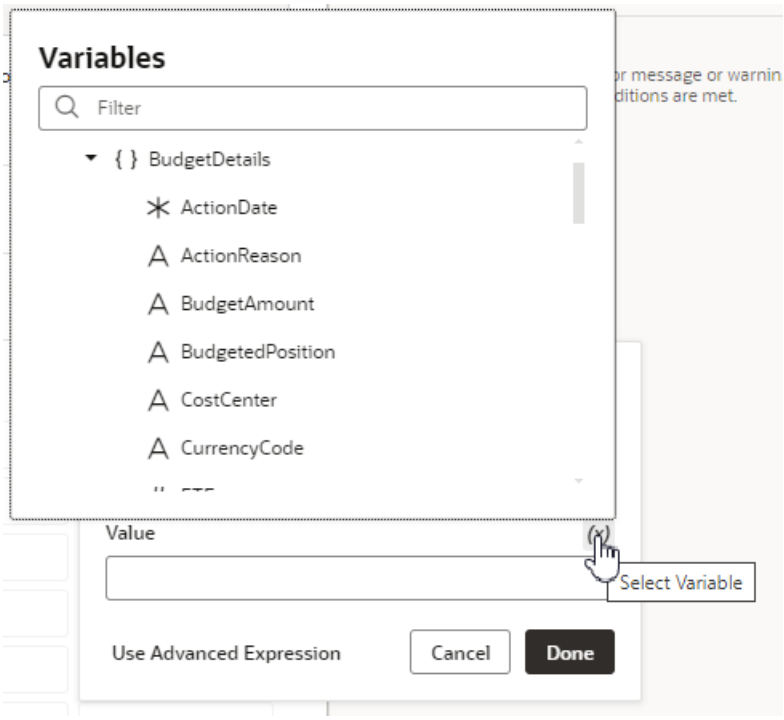
1. In the Fields and Regions editor, find the field you want to modify.
2. Click the dash in the Value column to open a popup dialog for setting the value.



If there's already a value set for the field, click the value and select **Edit Value** in the popup menu. You can also select **Remove Override** in the menu to remove the value.

3. In the popup dialog, do one of the following:
   - Select **Clear Field Value** to remove any default value already set for the field by other rules. For example, you can use this option to remove the value set by the Default built-in rule. When the rule is applied, this option makes the field's value empty (the value is set to `null`).
   - Select **Set Field Value** to enter a default value for the field, which can be:
     - A static value (like "euro"), or

– An expression which typically uses one or more different variables (like `$fields.BudgetDetails.CurrencyCode.$value()`) to calculate the actual value shown at runtime. When you use an expression, the value is recalculated if a variable referenced in the expression changes.
To help you create your expression, you can click **(x)** and select variables from the list:



If you want to create a more complex expression, click **Use Advanced Expression** to open the expression builder. See Build Advanced Expressions for more on how to use the expression builder.

> **Note:**
>
> If you see a lock, this means the field has been locked by the Default built-in rule, and you can't override it. Values that should not be overridden might be locked by Oracle.

At runtime, if the business rule's conditions are met, the field will show the value set in the popup.

What Do the Blue Indicators Mean?

Indicators in the **Regions and Fields** area help you see at a glance what has been changed.

In this example:

1. An empty blue circle next to a region (**When & Why Section**) indicates that while the region itself does not have a property overridden, at least one field in the region does.
2. The number in parentheses next to the region indicates the number of fields in this region that has overridden properties.
3. A blue dot next to a field or region indicates that there is at least one overridden property at this level.

Locks, Blanks, and Dashes

Let's take a look at what the lock icon, empty spaces, dashes, and grayed text mean in the context of a given rule.

The extent to which these attributes are displayed depends on the type of rule you're looking at. The default rule, for example, always shows the property values as set by Oracle. If nothing has been overridden, the values for the Required, Hidden, and Read-Only properties are always Optional, Visible, and Editable, respectively.

For all other rules:

- A dash indicates that a value has not yet been set for the property by this rule. At runtime VB evaluates all the rules from bottom to top, so a property setting for one rule can be overridden—and then overridden again—by rules that are higher up in the list.
- A lock icon means that a value has been locked by Oracle and cannot be overridden. (Any property value that appears in light gray is an Oracle-seeded value.)
- Blank fields represent system fields that are not available for modification by anyone.

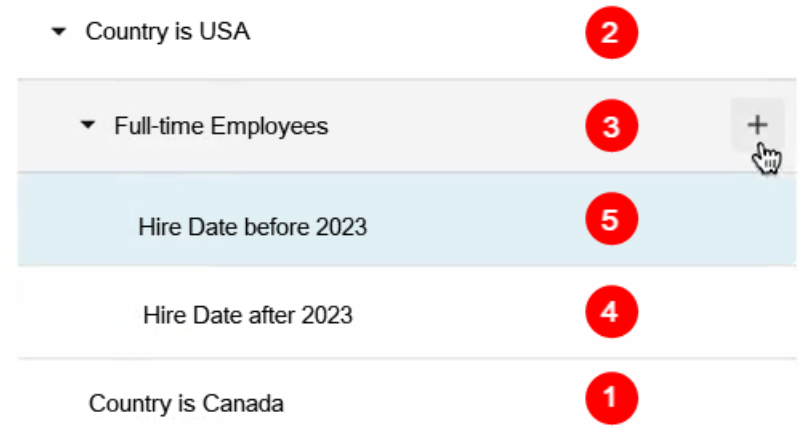
Use Nested Rules

Nested rules help you avoid having to write complicated conditions and can improve rule organization, especially if you have lots of rules.

A nested rule is simply a parent rule with one or more child rules, indicated in the UI by indentation. When you use nested rules, the children of a rule can override the properties set by its parent(s).

In addition to the typical nested rules, there are two special types of nested rules (If/Else and Switch) that behave slightly differently. These special types are described later, but first let's look at typical nested rules.

Here's a simple example of a nested rule:



This example shows two sets of nested rules:

- **Country is USA** and **Country is Canada** are at the same level
- **Full Time Employees** is nested beneath **Country is USA**
- **Hire Date before 2023** and **Hire Date after 2023** are both children of **Full-time Employees**.

The red numbers indicate the order in which these rules are evaluated at runtime. Let's take a closer look at what that means:

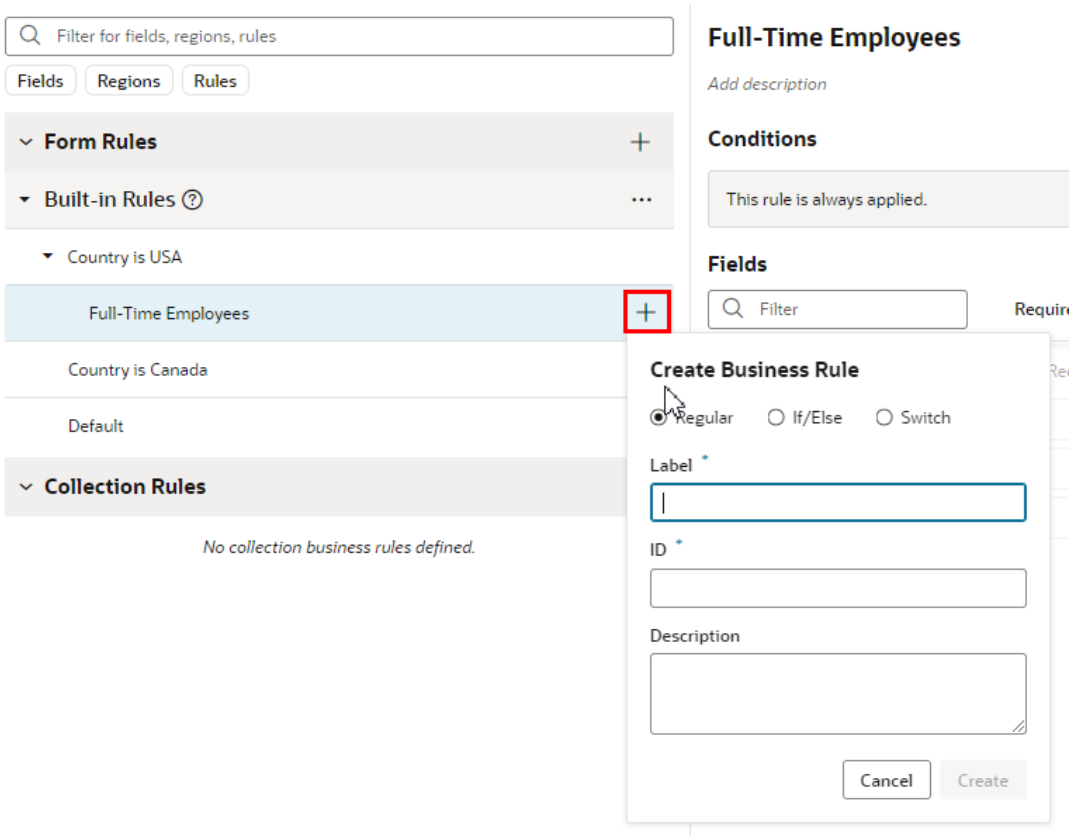
1. Evaluation begins at the bottom with **Country is Canada**. If the conditions for this rule are met, the rule's property overrides are applied. Next:
2. **Country is USA** is evaluated.
   - If the rule's conditions are NOT met, the rule doesn't override any properties. In addition, its child rules (**Full-time Employees**) are never evaluated.
   - If the conditions for **Country is USA** ARE met, the property overrides defined in the rule are applied, and **Full-time Employees** is evaluated.
3. If the conditions for **Full-time Employees** are NOT met, the rule does not apply any properties, and its children (**Hire Date after 2023** and **Hire Date before 2023**) are never evaluated.
4. If the conditions for **Full-time Employees** ARE met, the property overrides defined in the rule are applied, and its children are evaluated.
   - **Hire Date after 2023** is evaluated first, because it is lowest, followed by **Hire Date before 2023**.

Although you could write a more complex set of conditions to achieve the same outcome, nested rules make it much simpler to apply multiple property overrides at runtime.

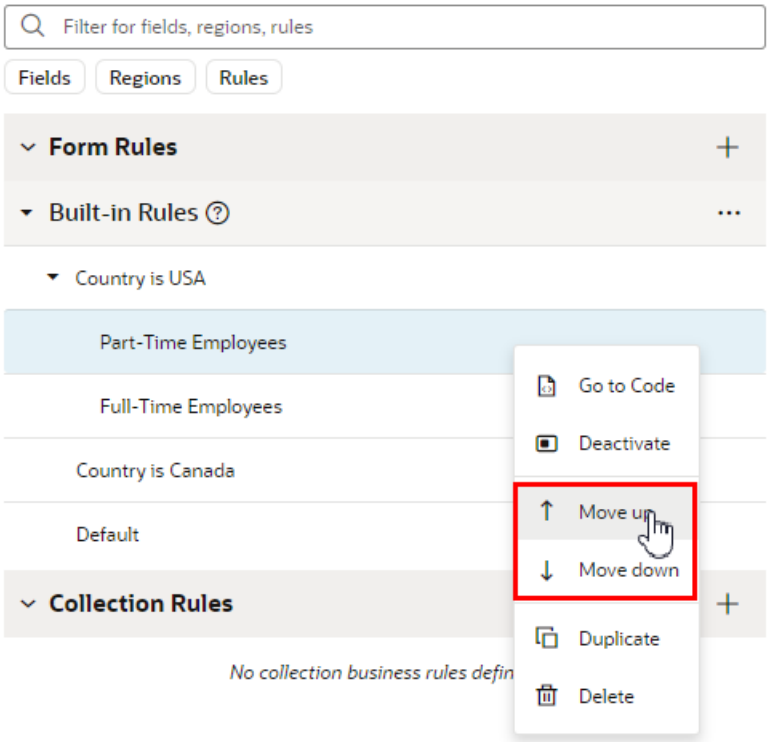
Add a child rule

You can add a child rule to any of your extension rules:

1. Click + next to the rule that you want to be the parent rule.



2. Type the Label of the child rule. (The ID field is automatically populated based on the Label, but you can type a different ID if you want.) Click **Create**.
3. Right-click the child rule, and then use **Move up** and **Move down** in the options menu to move the rule into the position you want.
   Remember, child rules that are peers are evaluated from the bottom up, so the order is important.



When you duplicate a rule, the rule's children are also duplicated. In the example above, if you duplicated **Full-time Employees**, a new rule (**Full-time Employees copy**) will appear under the parent (**Country is USA**), along with its child rules **Hire Date not set**, **Hire Date before 2023**, and **Hire Date after 2023**.

Add an If/Else Rule

An If/Else rule is a special type of nested rule where only the first child rule meeting the conditions is applied, and none of the remaining child rules are evaluated.

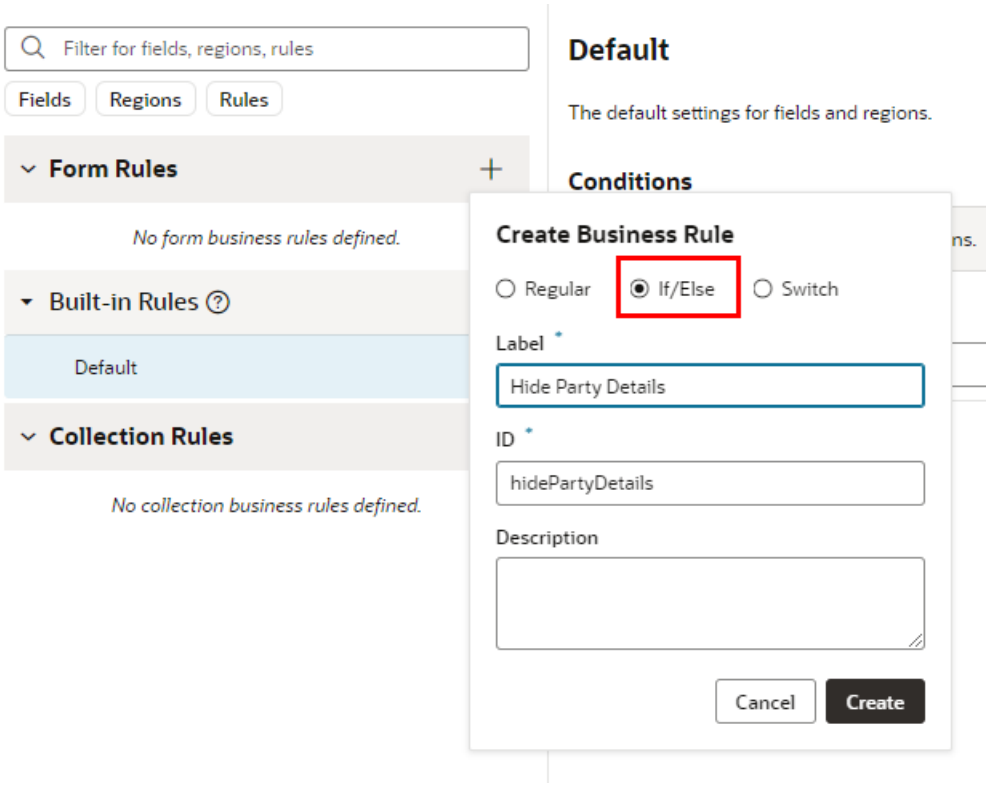
You use an If/Else nested rule when only one of the rules in a list of rules needs to be applied. Unlike regular business rules, where every rule is evaluated, the child rules in an If/Else rule are evaluated in order until one is applied; the unevaluated child rules are skipped. This means that rules won't be evaluated if they don't need to be.

For example, you might have a rule that should be applied when the user's role is Sales Manager, and some rules that should only be applied when the user is a sales manager in the UK, or in Canada. Handling this type of situation, where only one rule should be applied, is very simple if you use nested If/Else rules. Let's take a look at how you could do this.

1. In the business rules editor, click + to open the Create Business Rule dialog box.

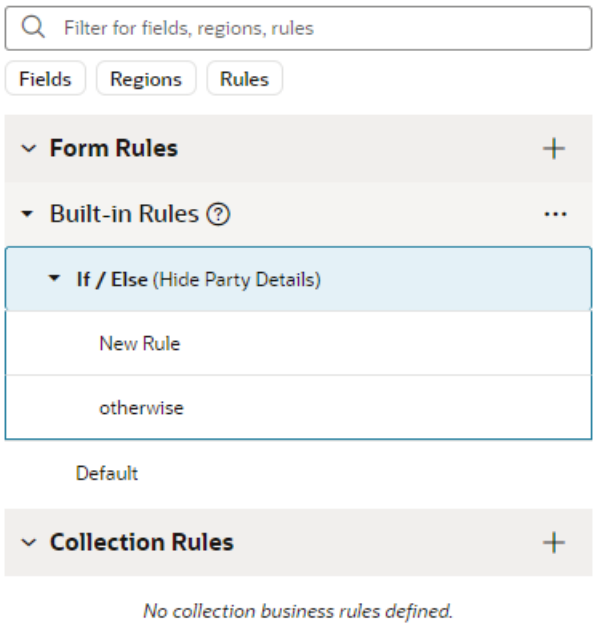
   You can create If/Else rules for form and collection rules.

2. Select **If/Else** in the dialog box.



3. Enter a label, id, and description for the rule. Click **Create**.

   An If/Else rule containing two child rules is created by default. The default names for the two child rules are "New Rule" and "otherwise", but you can rename them to something more meaningful. You can add as many child rules as you want, but remember that only one of them will be applied.



4. Select New Rule.

   The first child rule is the first "If" part of the "If/Else" nested rule. In If/Else nested rules, rules are evaluated from the top to the bottom. New Rule is the first rule in the list, so it's the first to be evaluated.

   a. Click Edit, and then define the rule's conditions, and give the rule a meaningful name.

      Let's define this rule so that it's applied to users in the UK with the role Sales Manager.

   b. Set the property overrides you want applied when the conditions are met.

5. Select the otherwise rule.

   The otherwise child rule is the "Else" part of "If/Else", and is only evaluated if none of the preceding rules were applied.

   a. Click Edit, and then define the rule's conditions, if you wish, and give the rule a meaningful name.

      If you don't set any conditions, the rule is always applied if it's evaluated. If a rule has no conditions you should make sure it's the last rule in the list.

      Let's define this rule so that it's applied to all users with the role Sales Manager.

   b. Set the property overrides you want the rule to apply.

6. Add and configure additional child rules.

   You can add as many child rules as you want, and arrange the order they are listed.

   Let's add a child rule that is applied to users in Canada with the role Sales Manager, and move the rule below the Sales Managers in UK rule.

   Click **Edit** to define the conditions, and then set the property overrides you want the rule to apply.

At runtime, starting with the first rule, the rules are evaluated in order until a rule is applied. In this set of nested rules, the last rule is applied only if the user's role is Sales Manager and they are not in the UK or Canada.

Add a Switch Rule

A Switch rule is a special type of nested rule where instead of evaluating a list of rules, a switch parameter (switchOn) is evaluated, and the parameter's value determines which case is applied.

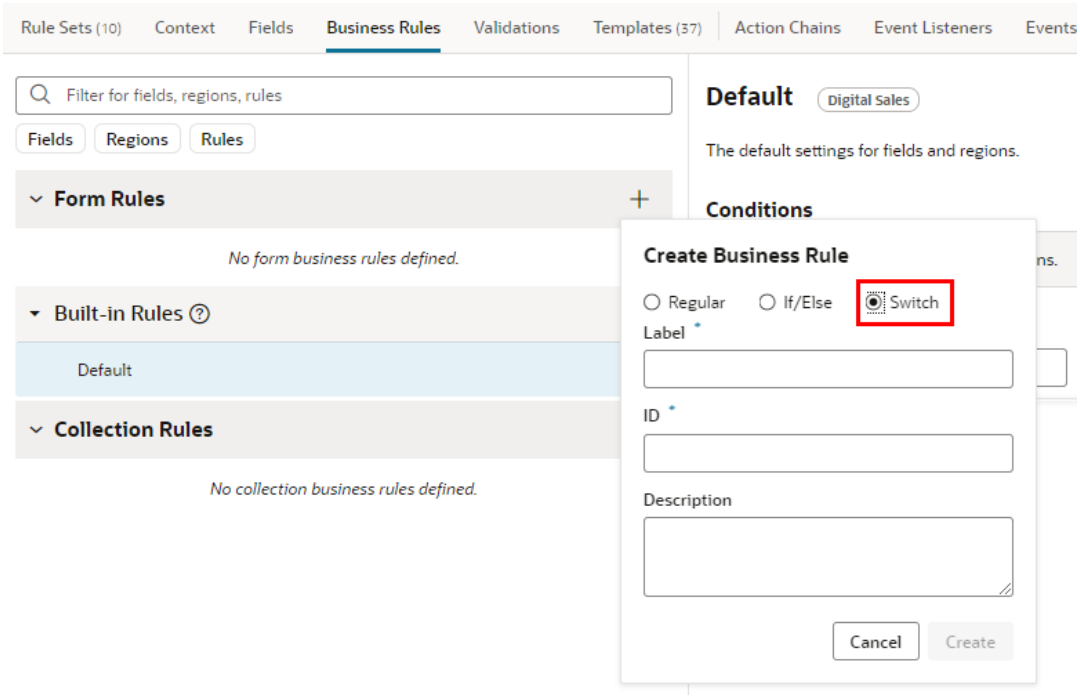
Let's say you want to apply a rule based on the role of the current user, and there are several different cases. For example, when the user's role is Sales Manager, you want the Sales Manager rule applied, when it's Administrator, the Administrator rule is applied, and so on. You can use the Switch nested rule to apply the appropriate rule based on the value of the user roles field, instead of evaluating a long list of rules at runtime.

Let's create a Switch rule that evaluates the role of a logged in user. You then create some different cases, for example, one case if the user role is Sales Manager, and another case if the role is Administrator. At runtime, the user role is evaluated, and based on the user role value, the appropriate case is applied.

1. In the Business Rules tab, click + to open the Create Business Rule dialog box.

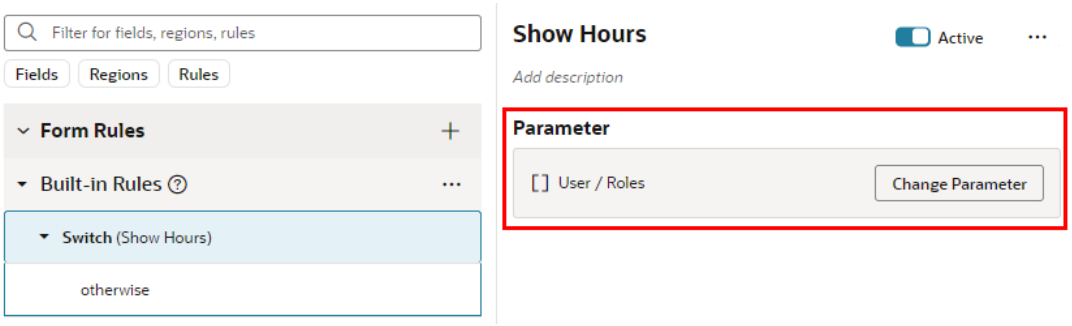
   You can create Switch rules for form and collection rules.

2. Select **Switch** in the dialog box.



3. Enter a label, id, and description for the rule. Click **Create**.
4. Select the Switch parameter.
   a. Select the **Switch** parent rule
   b. In the rule's Parameter panel, click **Change Parameter**, and then select a parameter in the dropdown list. Click **Change**.

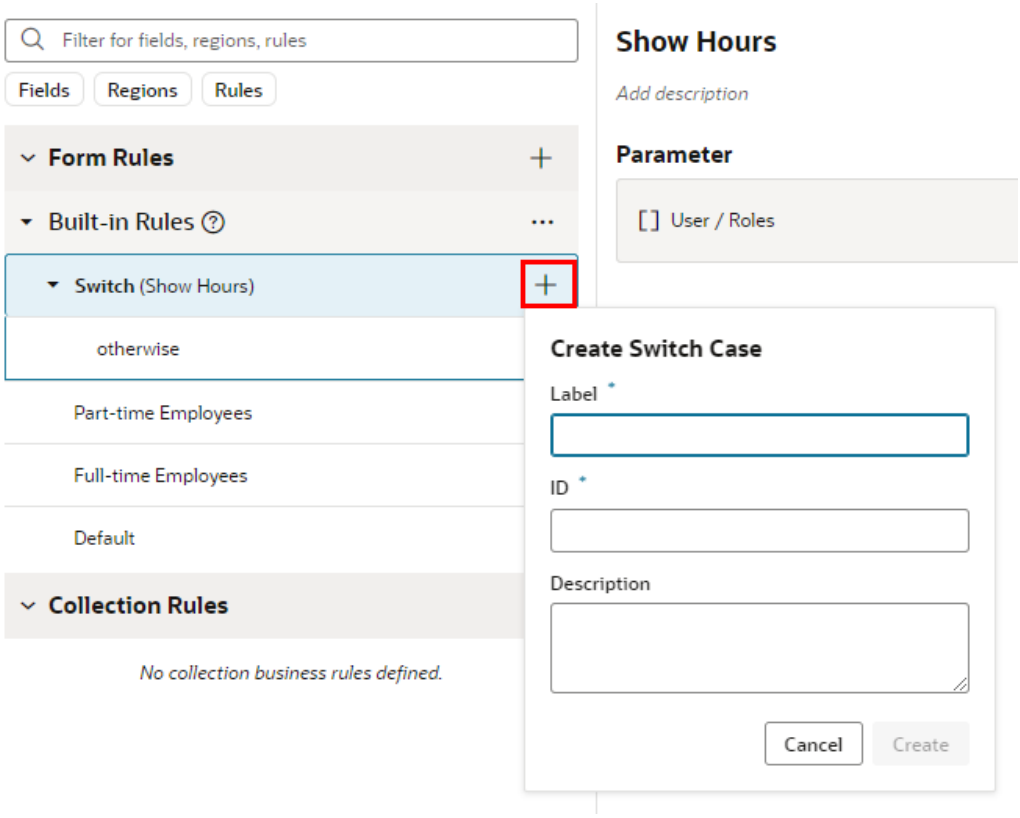
   The Parameter panel displays the parameter (the `[] User / Roles` field in this example).



5. Create cases.

Let's create a case that is applied when the user role is Sales Manager.

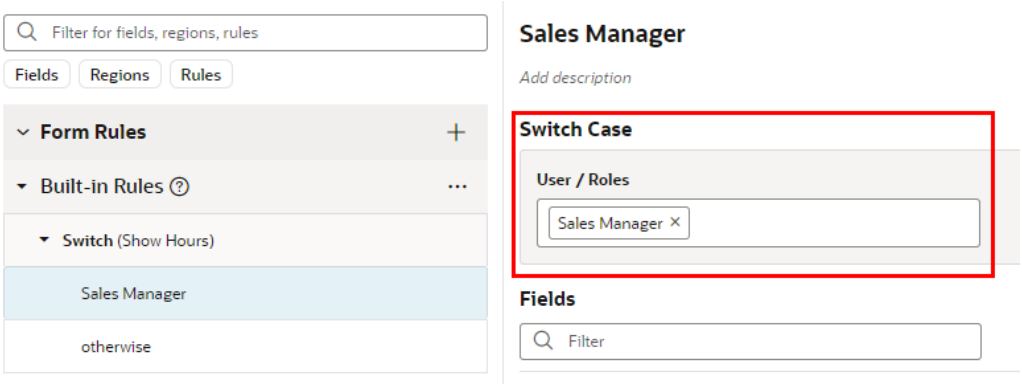
a. Click + next to Switch to open the Create Switch Case dialog box.



b. Enter a label, id, and description for the case. Click **Create**.

c. Select the case, and then enter a parameter value in the Switch Case text field.

You can enter multiple parameters in the Switch Case field. For this case, enter Sales Manager, so that the case is applied when the user role is Sales Manager.



d. Set the overrides you want the case to apply.

6. Create any additional cases.

For example, create a case called Administrator, and then set the case's parameter value to Administrator. You then set the overrides that should be applied when the user role is Administrator.

7. Select the last rule in the list of nested rules (the default name is otherwise), and set the property overrides you want the rule to apply.

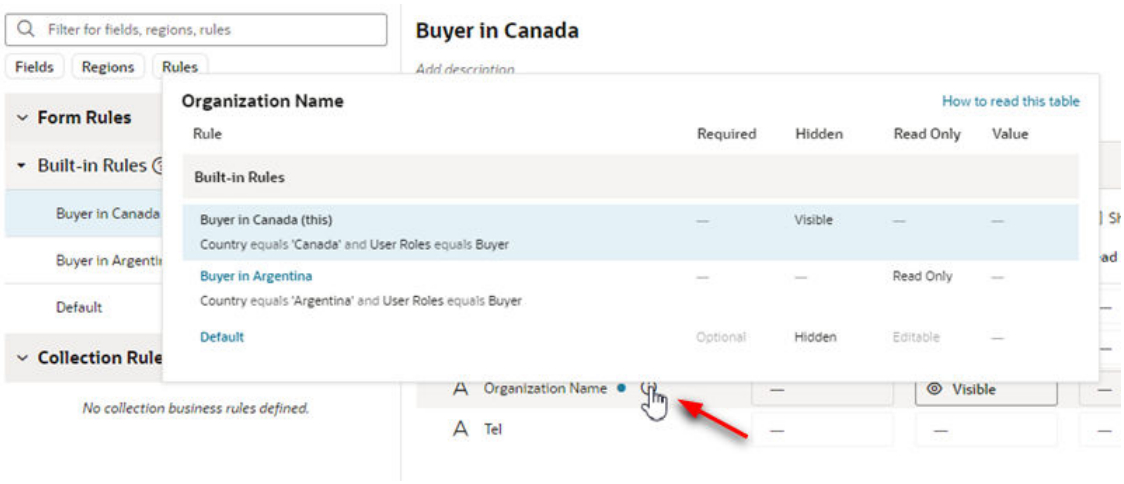
   The last rule is applied if none of the cases have been applied. You cannot create conditions for the otherwise rule.

Understand What Will Be Shown at Runtime

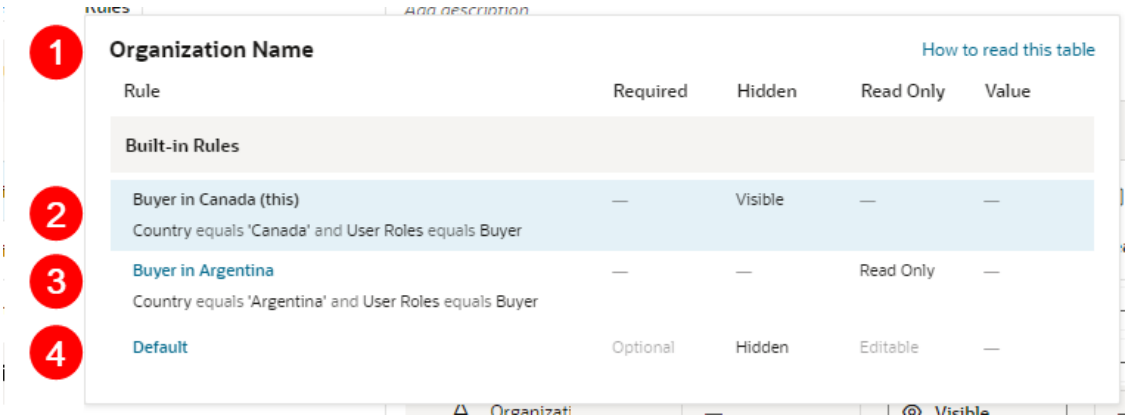
At runtime, VB evaluates all the rules from bottom to top, starting with the Default rule, then moving through the built-in rules.

As long as a rule's conditions are met at runtime, the rule's property values are applied to the display. However, since rules are evaluated one at a time (starting at the bottom), the topmost rule effectively takes precedence, as it can override what was previously set by lower rules.

To help you evaluate what will be shown for fields and regions at runtime, the editor provides a *pop-up viewer*. To display the pop-up viewer, hover over the field or region until you see an info label, then click it:



The pop-up viewer shows all the *active* rules that modify a region's properties, or a field's properties within the same region:



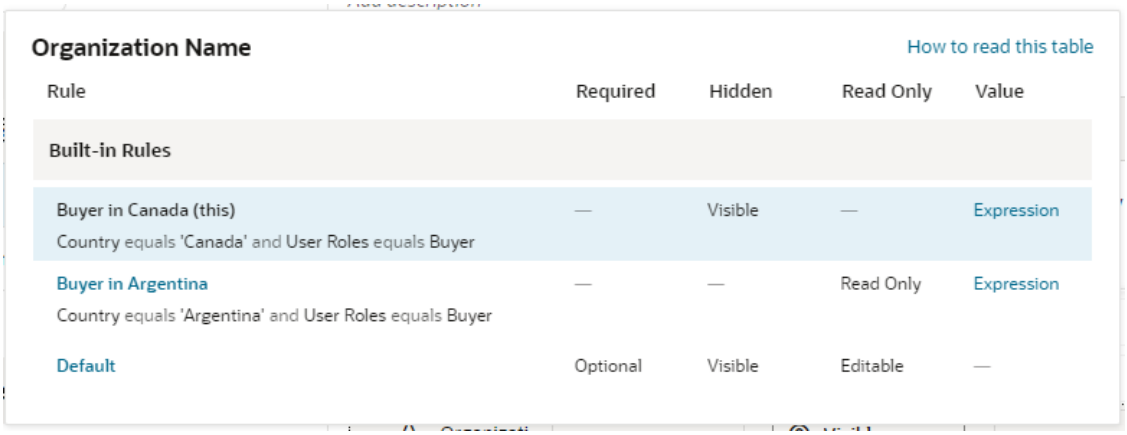
Let's take a closer look at the pop-up viewer:

1. Name of the field (Organization Name) that these rules pertain to.
2. Currently viewed rule, which is shown with a blue background. The conditions for each rule appears beneath the rule's name.
3. Another rule. If a rule name appears in blue, you can click it to see *all* the fields/regions modified by that rule (as opposed to just the selected field).
4. The Default rule, which specifies the out-of-the-box behavior. The Default rule is always active and cannot be modified. And since it doesn't have any conditions, it will always be considered "true".

Interpret the Pop-Up Viewer

Let's examine the rules for the **Organization Name** field to see how it would appear to two different users: first, a buyer in Canada, and then a buyer in Argentina.

The pop-up viewer shows that three rules have modified the **Organization Name**:



Let's suppose the current user satisfies the conditions for the **Buyer in Canada** rule, that is, they have the User Role "Buyer" and are located in Canada. What will he or she actually see on the page?

Let's begin by looking at the first rule that is evaluated, the **Default** rule, which is always applied. The properties for the **Organization Name** field are set to Optional, Visible, and Editable, and no value is set for the field. None of these properties are locked in the Default rule, so they can all be overridden by rules above it.

Moving up from the Default rule, the **Buyer in Argentina** rule is not enforced because it's conditions aren't met (the current user is in Canada.)

Moving up to the **Buyer in Canada** rule, our current user satisfies the rule's conditions, so this rule will be enforced, overriding the properties enforced by lower rules (in this case, just the Default rule.)

- The **Required** property does not have a value enforced by this rule, as indicated by the dash mark. Moving down the **Required** column, the next rule (Buyer in Argentina) doesn't enforce a value for this property because its conditions aren't met, so we look to the Default rule, which enforces a value of Optional. So, for the buyer with the "supporting role" located in Canada, the **Organization Name** field is optional because no rule overrides the value set in the Default rule; in other words, the user doesn't have to supply a value for this field before submitting the form.

- Now let's scan the **Hidden**, **Read Only** and **Value** columns in the rule. Two of these have values that are enforced by the rule, overriding values set in lower rules; that is, the Hidden property is set to Visible, and the Value property is set to an expression. The rule doesn't set a value for the Read Only property.

For users with a "Buyer" role in Canada, then, the **Organization Name** field will be visible and its value set to an expression (as the field is not Read Only, the user has the power to change the value.)

Now let's see what happens at runtime if the user is a buyer but located in Argentina, rather than Canada. Once again we start with the Default rule, which sets the properties described above.

Then we move up to the next rule, **Buyer in Argentina**. This time, both conditions are satisfied, so let's look at what this rule does:

- Once again, this rule does not enforce a value for the **Required** property. Continuing down the column we reach the Default rule, which states that the **Organization Name** field is Optional for these users.
- Likewise, the **Hidden** property does not have a value, so we take the value from the Default rule, Visible.
- The **Read Only** property is set to Read Only, overriding any value set in lower rules (just the Default rule, in this case).
- The **Value** property is set to an expression (this expression is different from the one set in the Buyer in Canada rule.)

Moving up to the **Buyer in Canada** rule, this time the conditions for this rule are NOT satisfied, because the user is not in Canada, so this rule is not enforced.

In summary, buyers with the "supporting role" in Argentina can see the **Organization Name** field, but, unlike their counterparts in Canada, it is read only, and its value might be calculated differently.

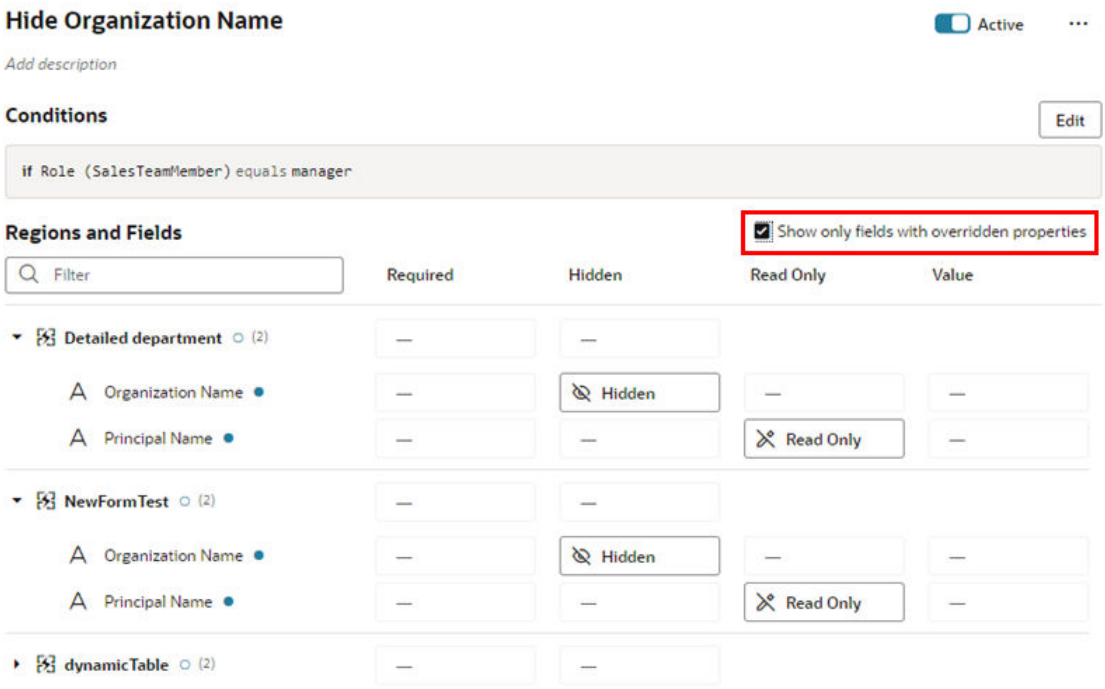
Identify the Regions and Fields Impacted By a Rule

Use **Show only fields with overridden properties** to understand which regions and fields are impacted by a rule, and how are they impacted.

To see the regions and fields that are affected by a rule:

- Select a rule, and then enable **Show only fields with overridden properties**.

The list is filtered to only display the regions and fields affected by the selected rule. This lets you focus just on the properties that have been overridden by the rule. In this example, the list only contains fields impacted by the **Hide Organization Name** rule.

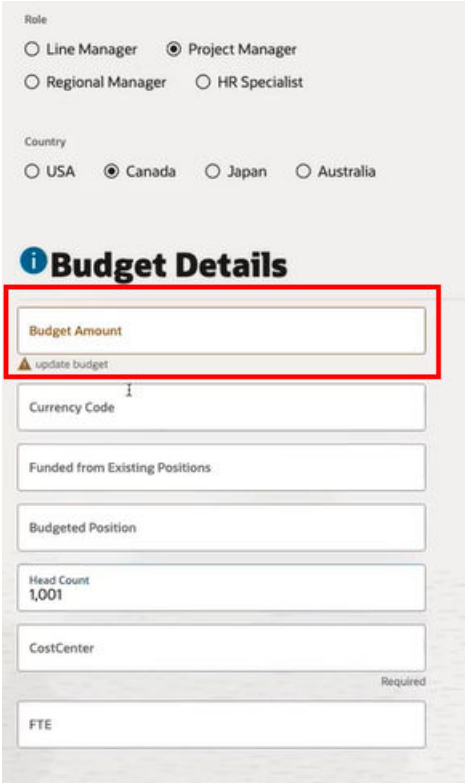


Display Messages When Conditions Are Met

You use a validation rule to display a message when certain conditions are met. When you create a rule, you define the conditions for when the rule should be applied, and define the message that should be displayed.

Validation rules are set at the object level, and not for a specific page. This means that when you define a validation rule for a form, for example, a form for editing the BudgetDetails business object, the rule is applied on every page where that form is used and the conditions are met.

Validation rules are particularly useful when you want to display some type of warning message based on data entered in a form. Suppose you want to display a message reminding the user to update the Budget Amount when the Head Count is more than 1000. You can create a rule that checks the value entered in the Head Count field, and display a message like this in the form:

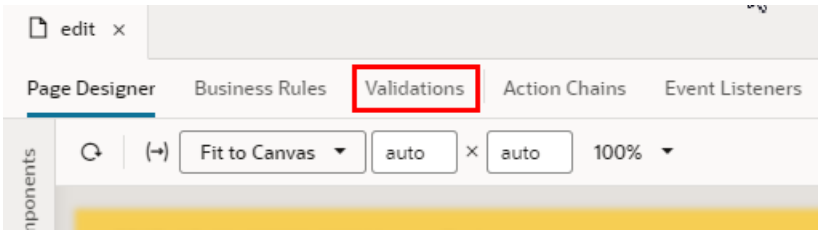


Create a Rule to Validate a Field

Let's take a look at how to create a rule that works like the example above, displaying a message in a form when the value for the Head Count field is over 1000.

To create the validation rule:

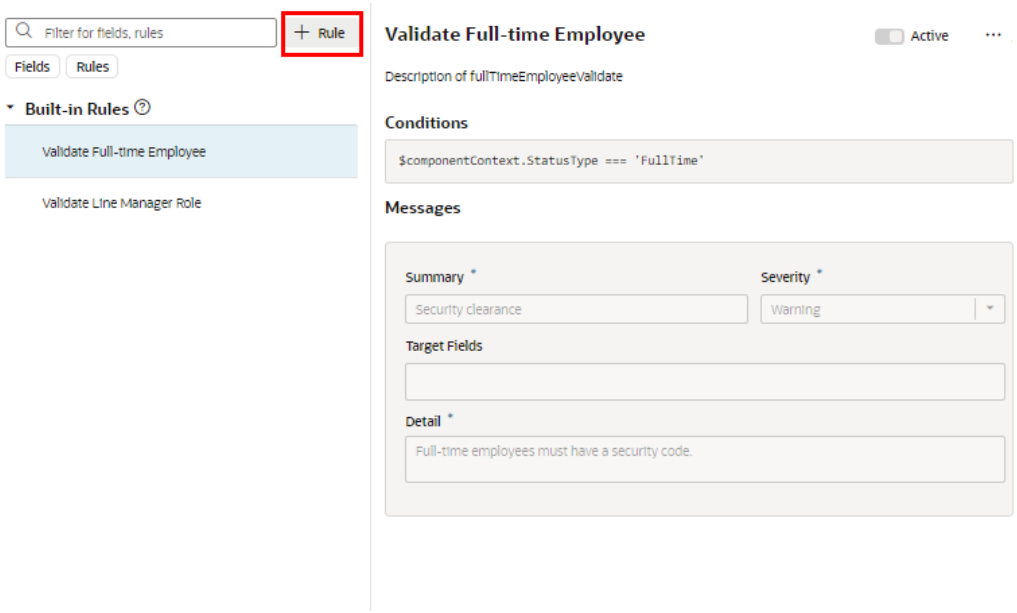
1. Open the page containing the form, and then open the **Validations** tab.



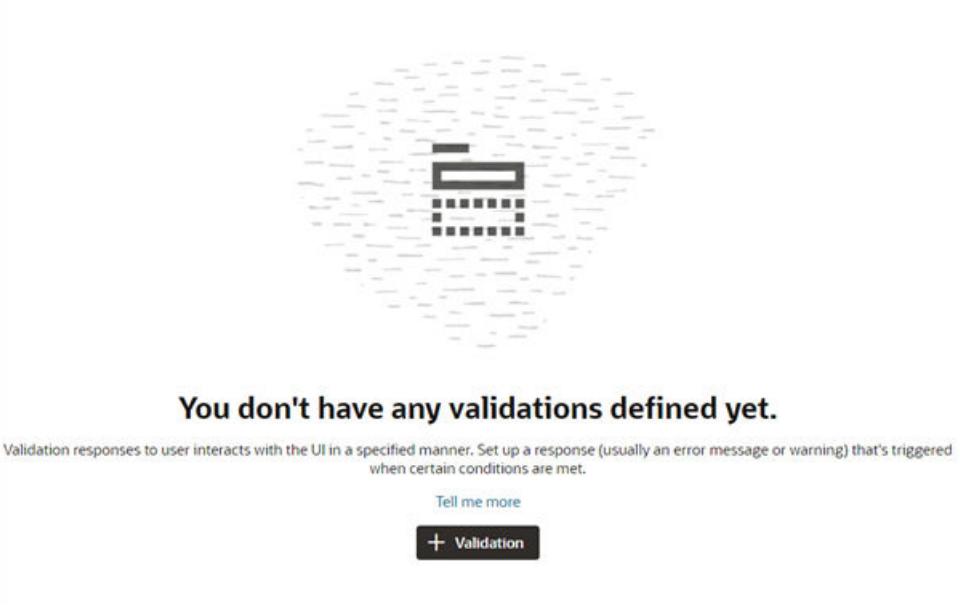
If the tab is not visible when the page is open in the Page Designer, open the Layout where the form is defined, and then open the **Validations** tab.

2. Create a new business rule.

- If some rules are already defined, click + Rule, then enter a label and description.



- If there are no rules yet, click **+ Validation**, and then enter a label and description in the popup.

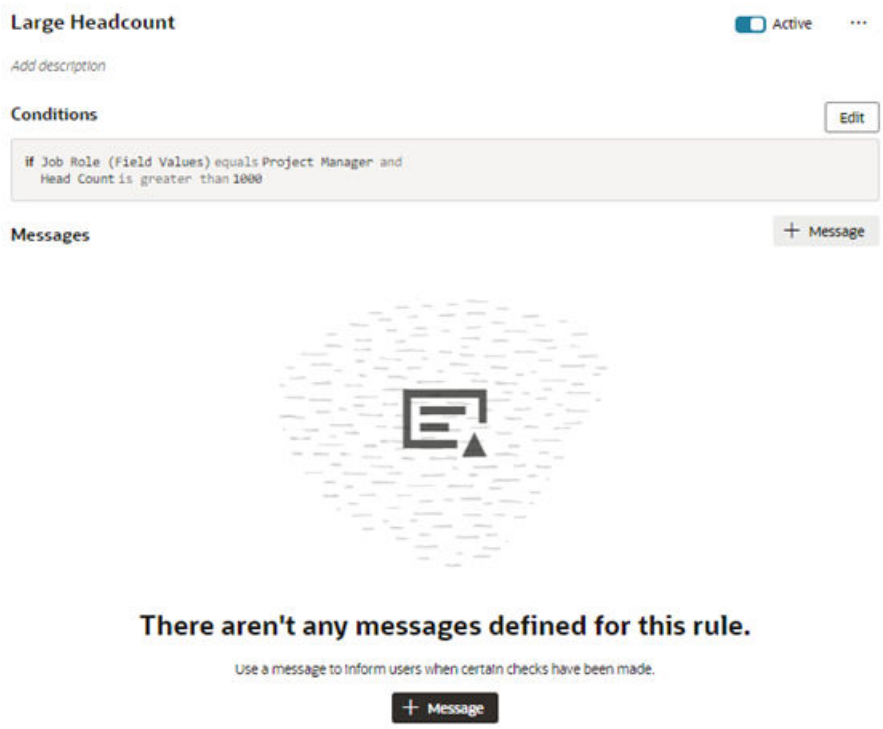


Click **Create**.

3. Click **Edit**, then specify the rule's conditions. Click **Done** when you're finished.

   You create conditions for validation rules using the standard condition builder, or you can click **Use Advanced Expressions** to use the advanced expression builder if you want to create more complex conditions.

   For this rule, we want to create two conditions: the Job Role must be 'Project Manager', and the value for Head Count must be greater than 1000. The rule will be applied when both these conditions are met:



4. Specify the message details.

   When the conditions are met, we want the warning message "Update budget" displayed under the Budget Amount field in the form.

   a. Click **+ Message**.

   b. Type the message texts in the **Summary** and **Details** fields.

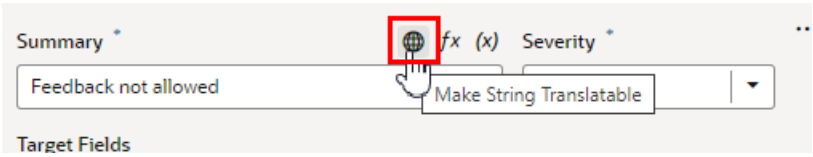
      You use these two fields to enter the text for the message.

      - **Summary**. The Summary text is displayed in the title of a message dialog. The text in the Summary field is not displayed on the page if you are using an inline warning, but it's still a required field.
      - **Detail**. The Detail text is the actual text message. This text is displayed inline in the form if you are displaying an inline warning, and in a message dialog if it's not displayed inline.

      Your message text can be a simple string, like "This number is invalid", or you can write more targeted messages by passing field values and context parameters into the message. For example, you could add something like this in the Detail field:

      ```
      [[ Restricted feedback applies to ${$fields.PersonName.$value()} who is part of ${$objectContext.Department}}. Select someone from the department without feedback restrictions.]]
      ```

To generate translations of the Summary and Detail texts, you can hover over the text field and then click to open the Translatable String popup:



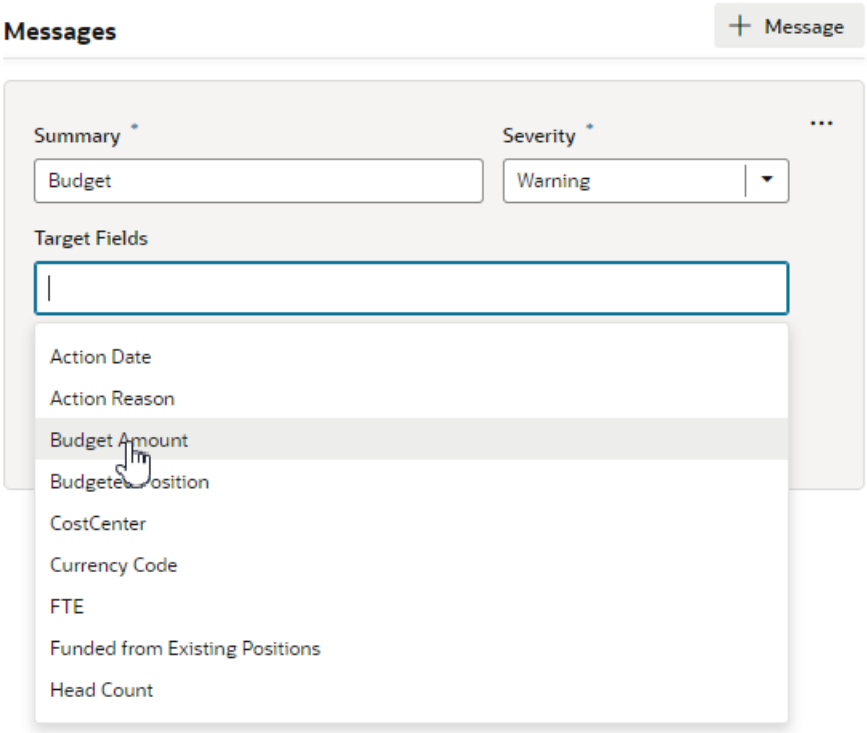
It's not possible to add parameters in the Translatable String popup, but you can click fx to open the Expression Editor, and then manually edit the translation object if you want to use parameters. For more, see Generate Translation Keys for Display Texts.

c. Select the Warning in the **Severity** dropdown menu.

   The Severity menu contains the following options:

   - **Error**. Choose this when there is some data that the user *must* correct before they can submit the form. This is the highest severity level.
   - **Warning**. Choose this to call attention to a field, for example, if you want to let a user know to check data that was entered. A warning message won't prevent the user from interacting with the page.
   - **Info**. Choose this for messages that are only informative.
   - **Confirmation**. Choose this for messages that confirm an operation or task was completed. This is the lowest severity level.

d. Select the field where you want the inline message displayed in the **Target Fields** dropdown menu.

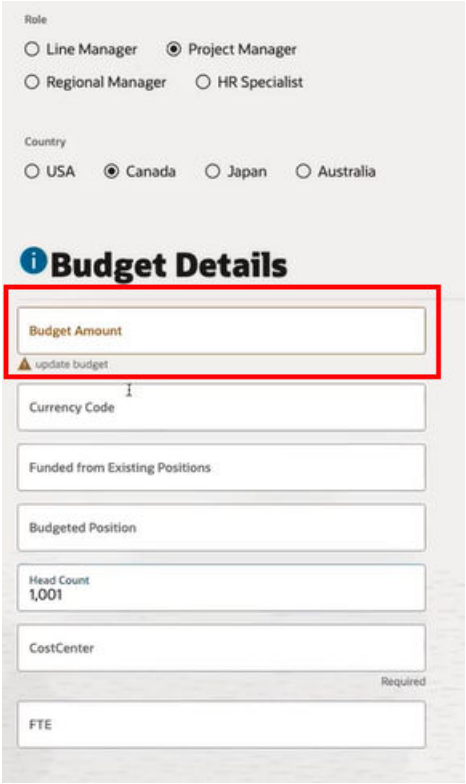


If you do not select a target field, and the form contains only one editable field, the message is automatically applied to it.

> **Note:**
>
> A message is not displayed inline when:
>
> - You do not select a target field, and the form has multiple fields.
> - You select more than one target field.
>
> If you want messages displayed in a dialog box instead of inline, a message component in the page needs to be manually configured to handle the messages.

To add a different message to the rule, say a message with different text displayed under a different field, click **+ Message** to create a new message and specify its details.

To check if your rule is working, view the page in Live mode and test the form by entering values to trigger the rule.



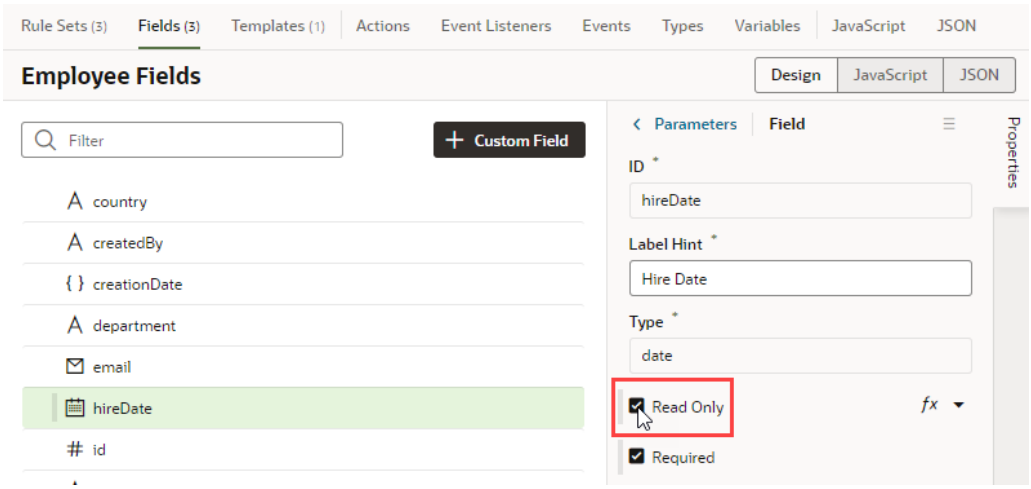
Edit a Field's Properties

When you edit a field's properties in a layout, your changes only apply to the field in the current layout. You might want to do this to override a field's properties in a specific layout, for example, to mark a field as Read Only. If you want to edit a property so that it's the same in all layouts—for example, if you want it to be Read Only always— you should edit the field's properties in the **Fields** tab.

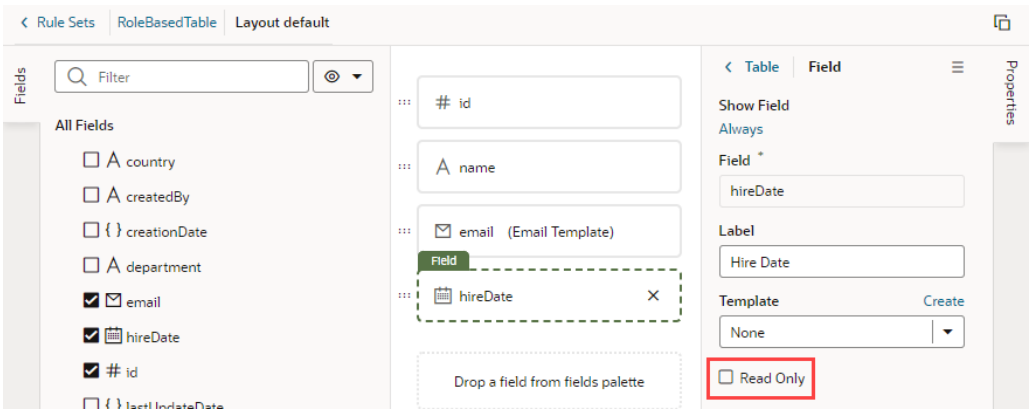
Set a Field to be Read Only

You can set a field to be Read Only when you don't want its value to be changed by everybody. For example, you might want only managers to be able to change an employee's hire date. You would then set the field to be Read Only in all layouts, except the one seen by managers.

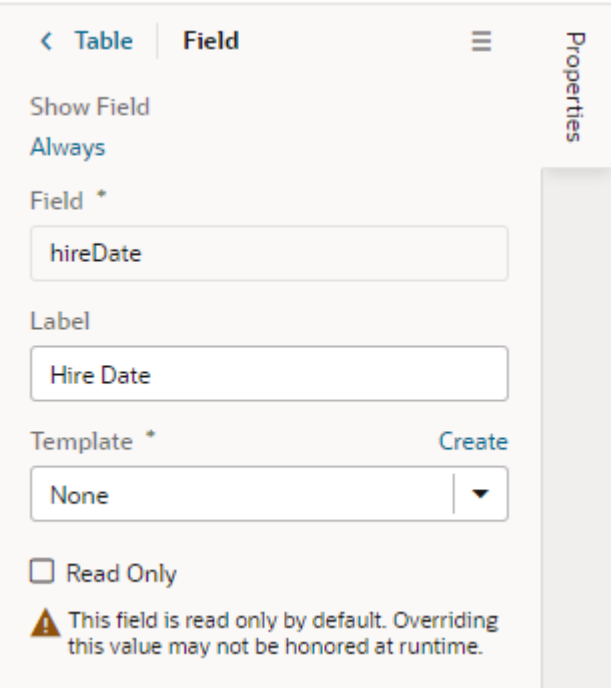
- To edit a field's Read Only property for all layouts:
  1. In the layout's **Fields** tab, select the field you want to work with.
  2. Edit the field's Read Only property in the Properties pane.



- To edit a field's Read Only property in a particular layout:
  1. In the Rule Set editor, open the layout and select the field in the center pane.
  2. Edit the field's Read Only property in the Properties pane.



If the field's Read Only property was set in the Fields editor to apply to all layouts, you would see a warning as shown here when you try to edit the property in a particular layout:



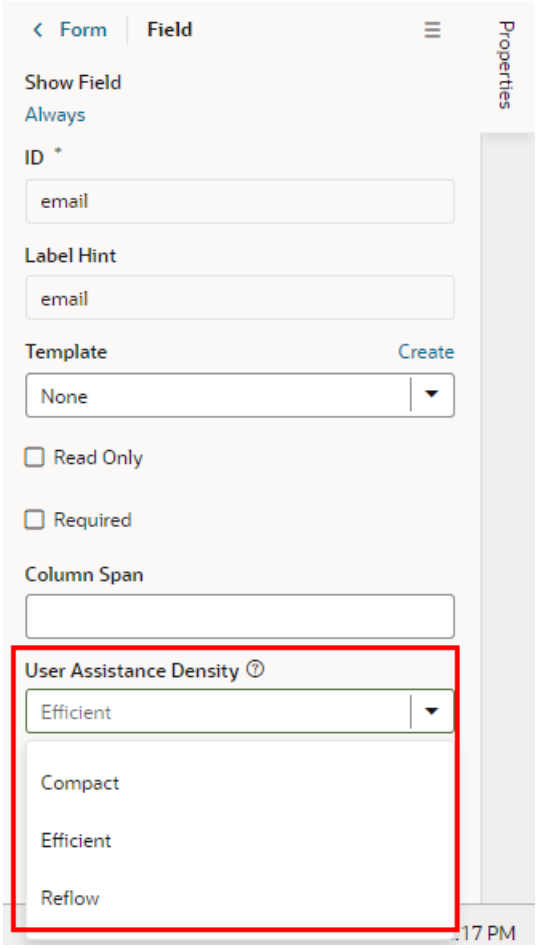
The Read Only property might not be editable if you select a field that has a template applied to it. You would need to remove the template if you want to edit the property in the layout.

Set How User Assistance is Rendered in a Layout

You use the User Assistance Density property to set how a field's user assistance text such as messages, help text and hints are displayed in the form.

To edit a field's User Assistance Density property in a layout:

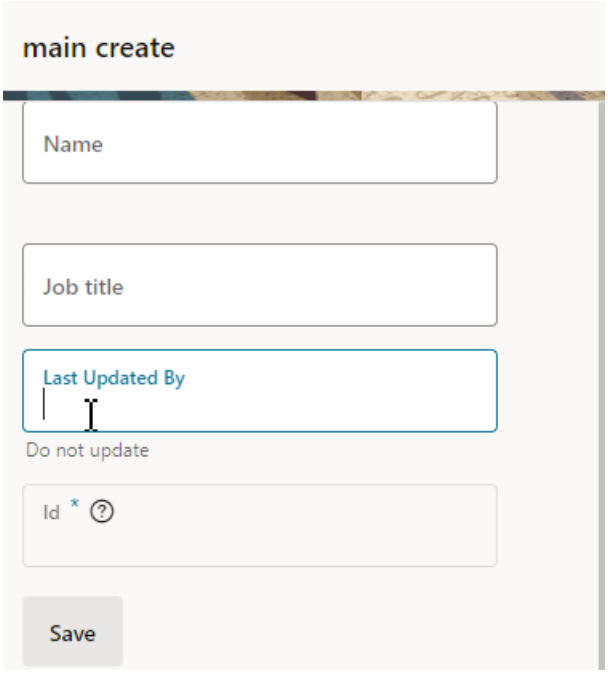
1. In the Rule Set editor, open the layout and select the field in the center pane.
2. Select the field's User Assistance Density property from the dropdown list in the Properties pane.



You can choose from three options:

- `compact` - With this option, user assistance text is displayed so that the layout is more compact, such as using a popup for messages and a required icon to indicate a Required field.
- `efficient` - With this option, any user assistance text is shown inline under the field. The form is rendered with space under the field for the user assistance text. This is the default option.
- `reflow` - With this option, any user assistance text is shown inline under the field. The form is rendered with no space under the field for the user assistance text. The space below the field expands to display the user assisstance text when the insert cursor is in the field.

This image of a form can help you visualize how these settings affect how fields are rendered:



In this form, the User Assistance Density property for the fields are set to different values:

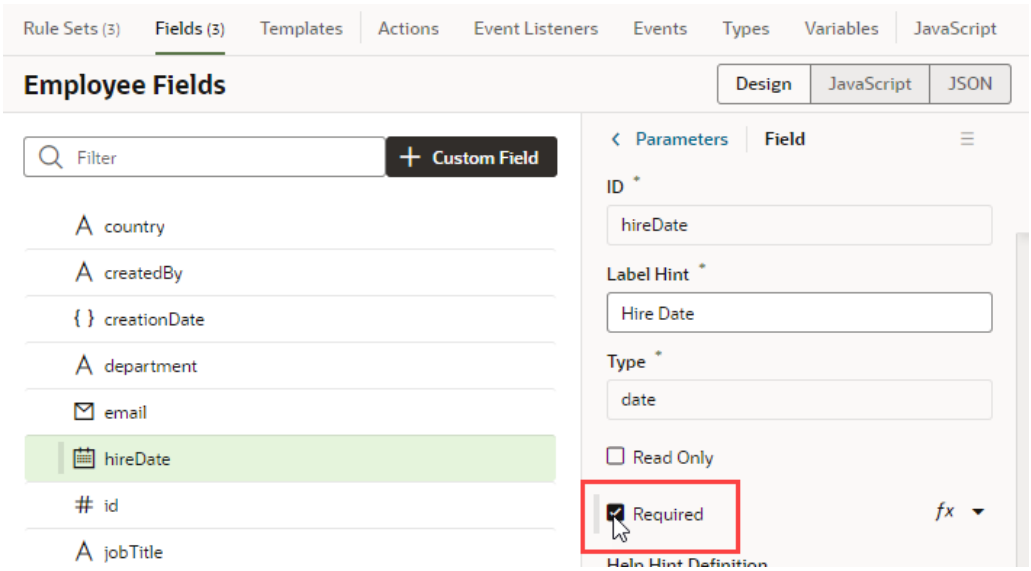
- the Name field is set to `efficient`,
- the Job Title field is set to `reflow`,
- the Last Updated By field is set to `reflow`, and
- the Id field is set to `compact`.

You can see that both the Job Title and Last Updated By fields are set to `reflow`, but that the space below the Last Updated By field, where the insert cursor is) has expanded so that the user assistance text can be rendered below it.

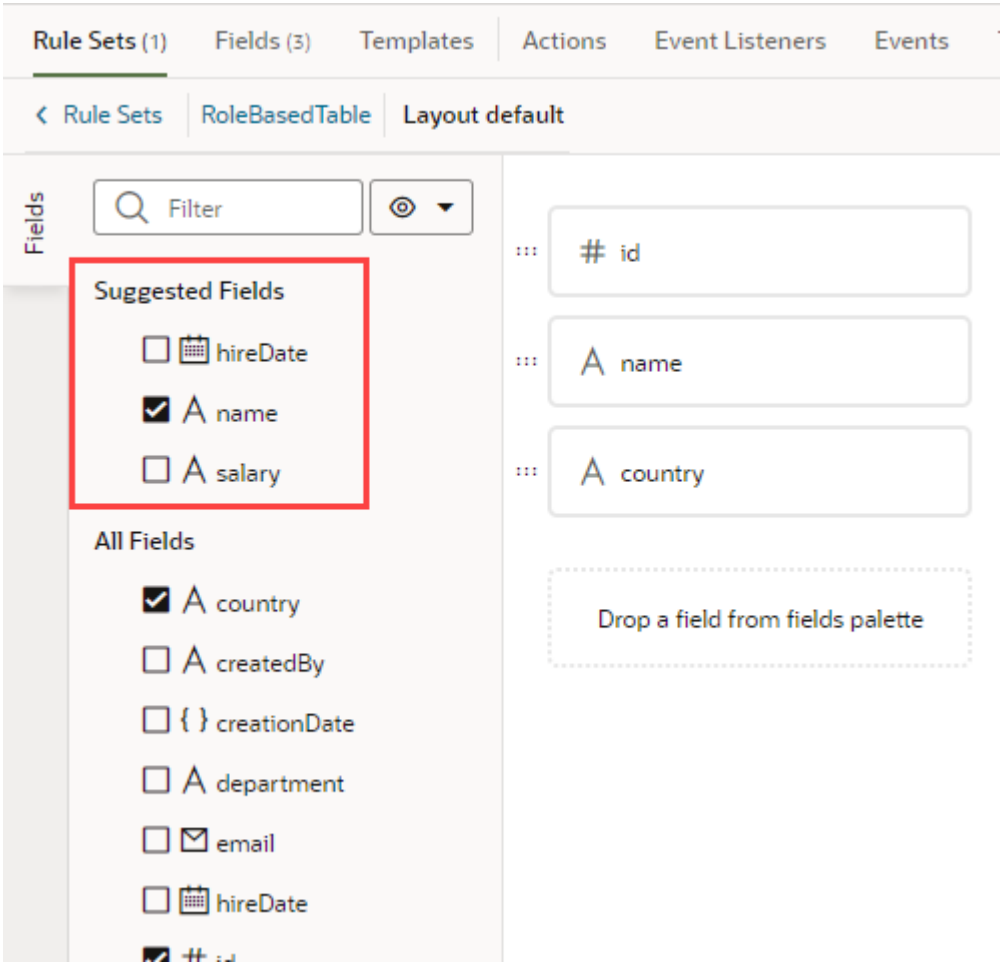
Set a Field as Required

When you set a field as required, users won't be able to save a record until the field's value is entered. You can set a dynamic form's field to be required in all layouts or in a particular layout, but the Required property for a dynamic table applies to all layouts and can only be set from the Fields tab.

- To edit a field's Required property for all layouts (dynamic forms and tables):
  1. In the layout's **Fields** tab, select the field you want to work with.
  2. Edit the field's Required property in the Properties pane.

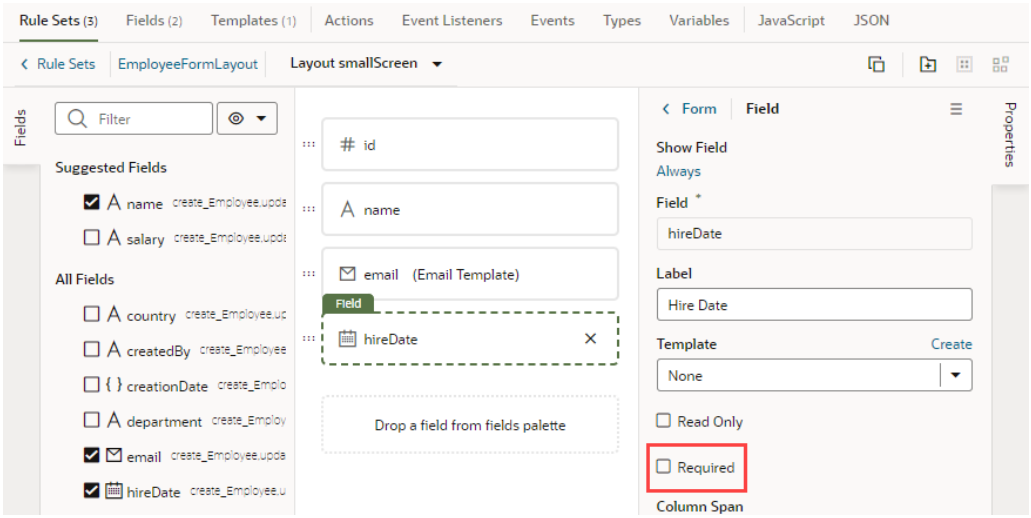


Fields that are marked as required show as Suggested Fields when you're building a layout to emphasize that they might be important or relevant to include in your layout:

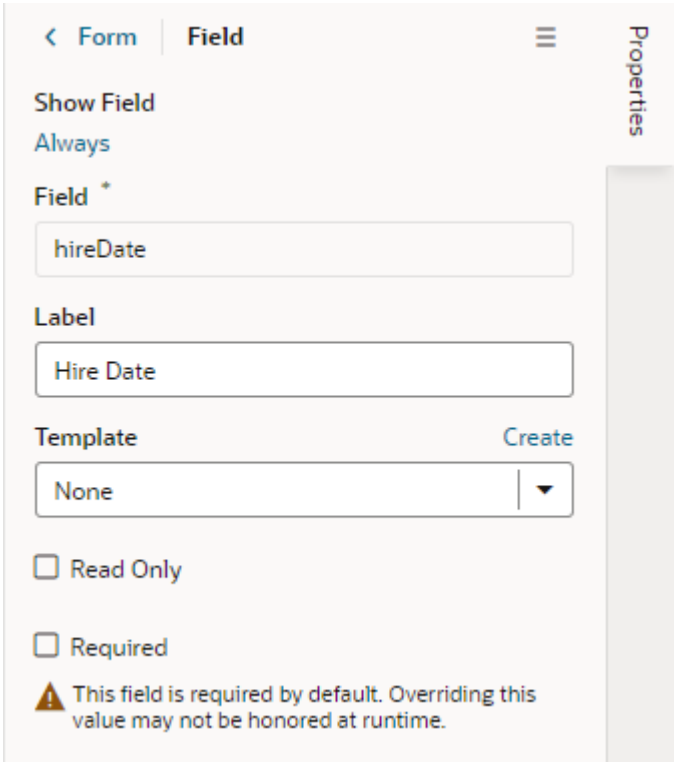


- To edit a field's Required property in a particular layout (only for dynamic forms):

1. In the dynamic form's Rule Set editor, open the layout and select the field in the center pane.
2. Edit the field's Required property in the Properties pane.



If the field's Required property was set in the Fields editor to apply to all layouts, you would see a warning as shown here:



The Required property might not be editable if you select a field that has a template applied to it. You would need to remove the template to edit the property in the layout.

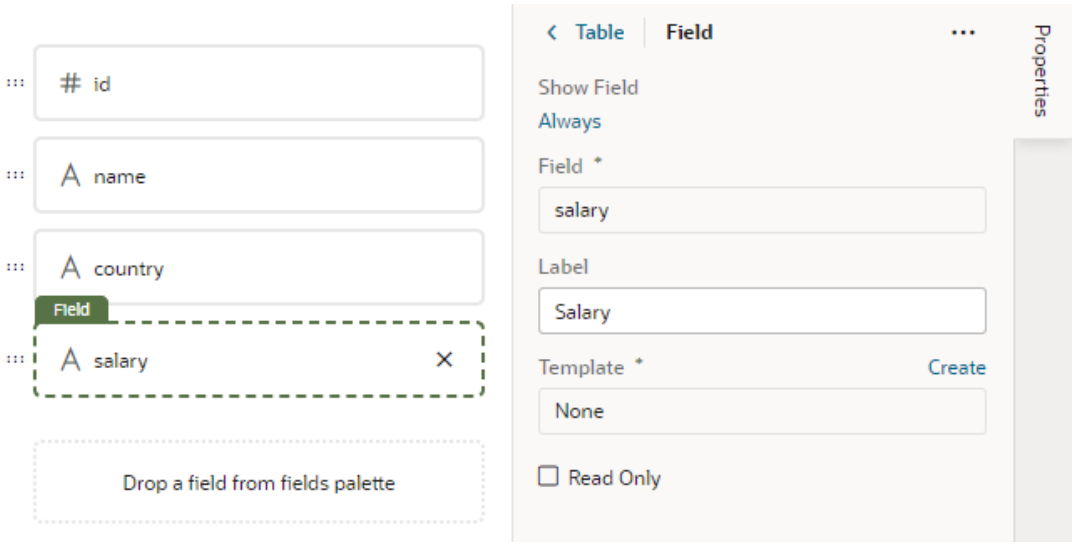
Use Conditions to Show or Hide Fields in a Layout

Fields in the active layout are displayed by default, but if you want to hide a field or group in a layout in some cases, for example, to hide it from everyone except managers, you can use the field's Show Field property to set conditions that determine when it is displayed. When you add conditions, the field is displayed only when the conditions you set are true. The conditions are only applied to the field in the layout you are editing.

To set display settings for a field in a layout:

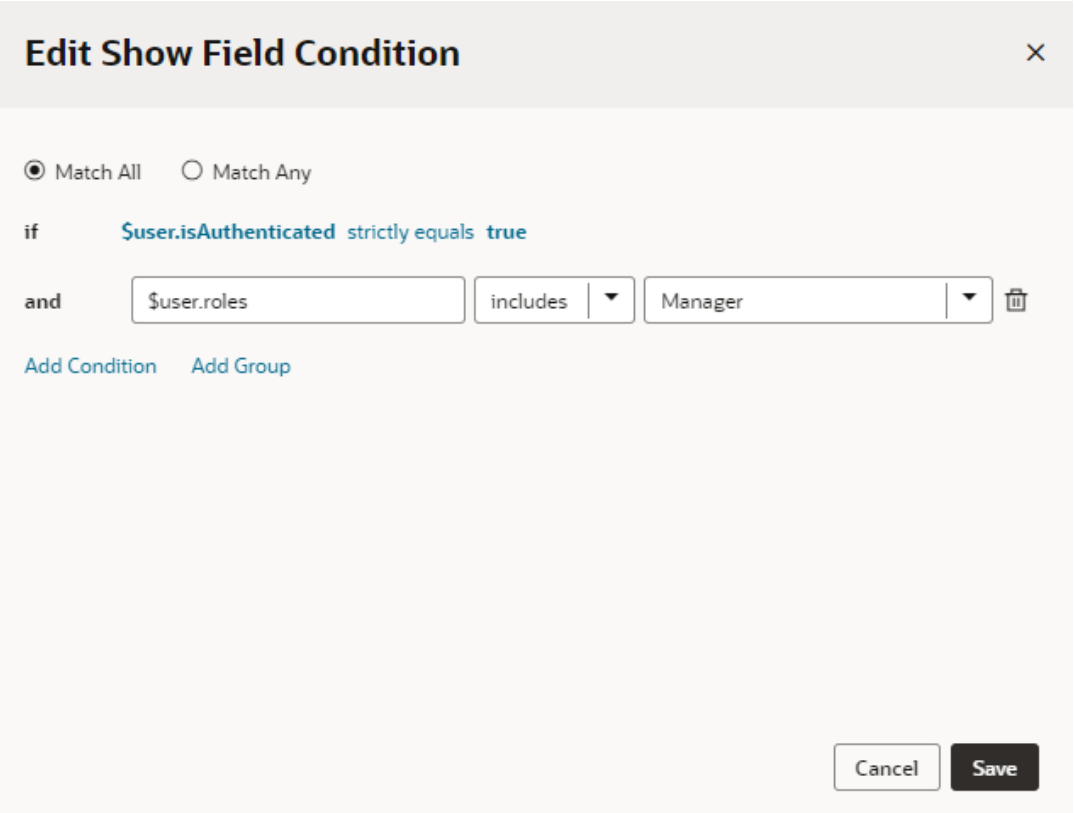
1. In the Rule Set editor, open the layout and select the field in the center pane.

   When you select the field, you can see the field's properties in the Properties pane. By default, the Show Field property is set to Always, so the field is always displayed.



2. In the Properties pane, click **Always** under the Show Field property to open the Edit Show Field Condition dialog box.
3. Define the field's conditions by selecting attributes, operators, and values in the condition builder in the dialog box. Click **Save**.

   You can add more conditions and group conditions to make the rule more precise. For example, you may want to display an extra field only if the user is authenticated AND is a manager. You would then create a rule with two conditions, and select Match All to require that both conditions are true.



Configure How Columns Render in a Dynamic Table's Layout

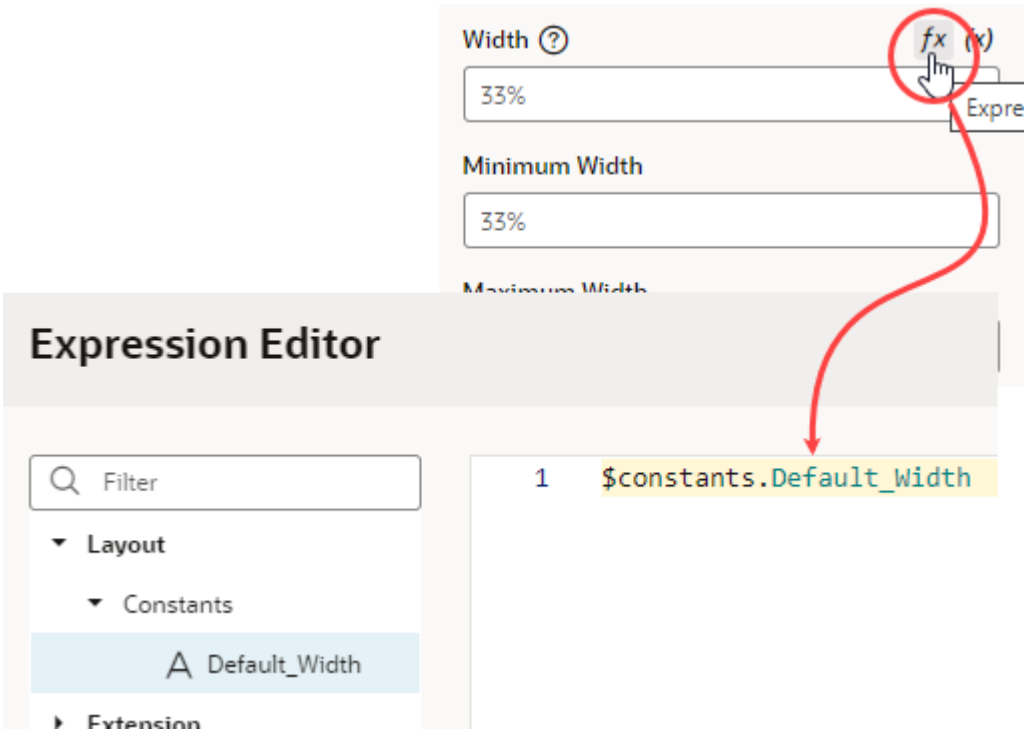
When you use a dynamic table in a layout, you can specify whether a field's data in the table should be sortable by its column header, and configure the field column's width to change the overall table display. You can also "freeze" table columns, so that a column's content remains visible, and a horizontal scroll bar is used to scroll the table's content.

Setting these properties for a field in a layout only applies to the current layout. Other layouts are not affected.

To configure columns in a dynamic table's layout:

1. In the Rule Set editor, open the layout and select the field in the center pane.
2. To manage the field's sortability, set a value for the **Sortable** property.
   - Select **enabled** to enable sorting on the column.
   - Select **disabled** to disable sorting on the column.
   - Select **auto** to enable column if the underlying model supports sorting.
3. To set the default width for the field's column on the table, set a value for the **Width** property. It can be a percentage or px value (for example, `100px`).

   You can also use expressions to control a column's size. For example, let's say you want a column to be 50 percent of the entire table. In this case, you could define a constant in the layout's Variables tab (for example, a string type constant called `Default_Width`, with a default value of `50%`). Then, hover over the **Width** property and open the expression editor, define an expression using the `Default_Width` constant, and click **Save**:



To further control the column's width, use the **Minimum Width** and **Maximum Width** properties to set the minimum and maximum widths of the column when the table is first rendered on the page. A user can manually resize the column width to make it narrower or wider.

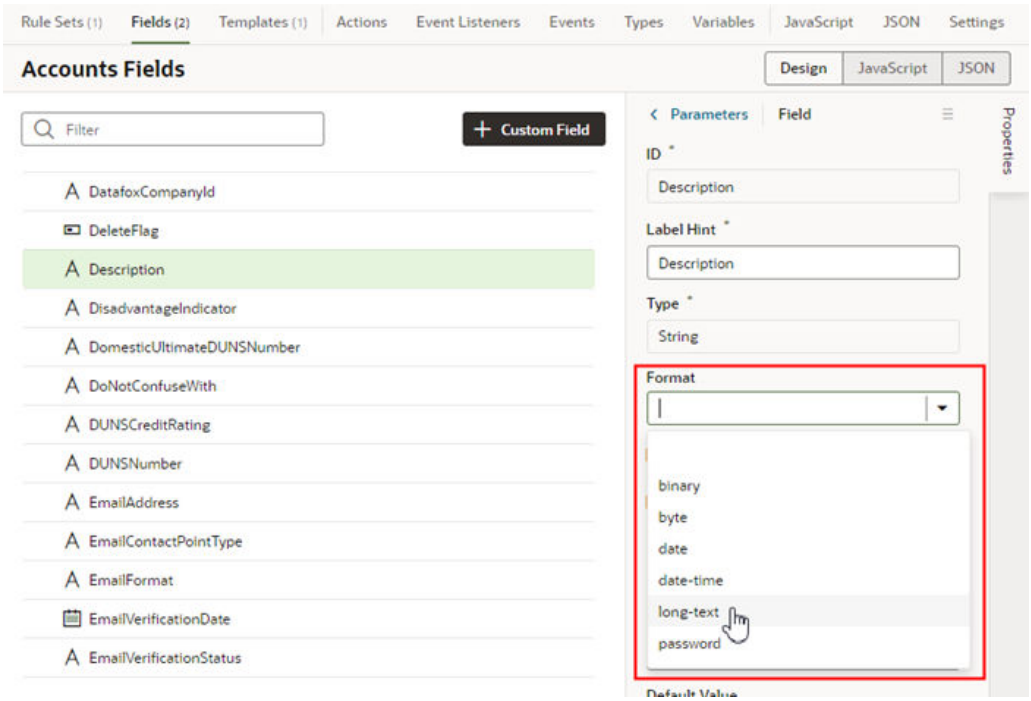
4. To "freeze" a column, select a value for the **Frozen Edge** property:
   - Select **start** to pin the column at the beginning, so that a user won't scroll horizontally past the column.
   - Select **end** to freeze the column at the end, so that the column is locked in view.

   For details on how to set frozen columns in a table, see **Frozen Columns** in the Oracle JET Developer Cookbook

Set a Field to Display as a Text Area in a Form

If a field's value is a long string, for example, when a field is used to display a long description, you can configure the field so that it is rendered as a multi-line text area in forms instead of the default single-line text field.

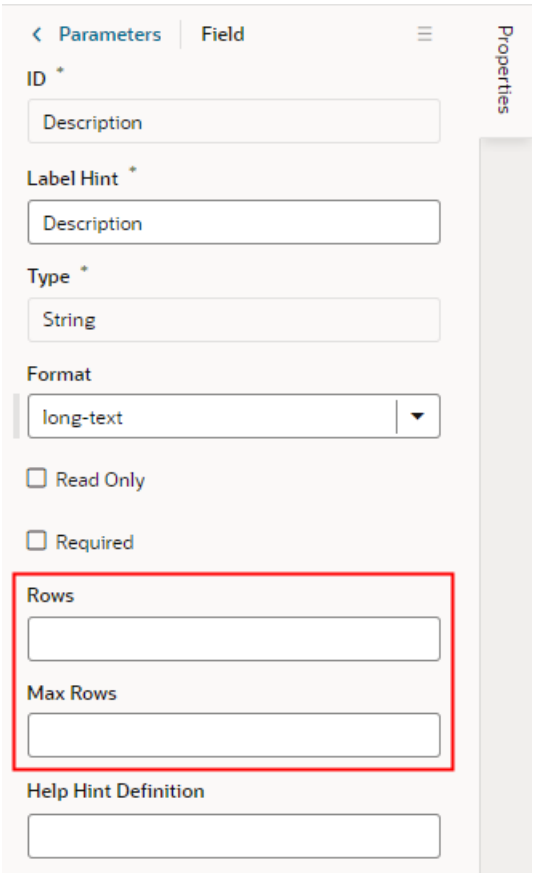
- To set a field to display as a text area in all layouts:
  1. In the layout's **Fields** tab, select the field you want to work with.
  2. Set the **Format** property in the Properties pane to `long-text`.



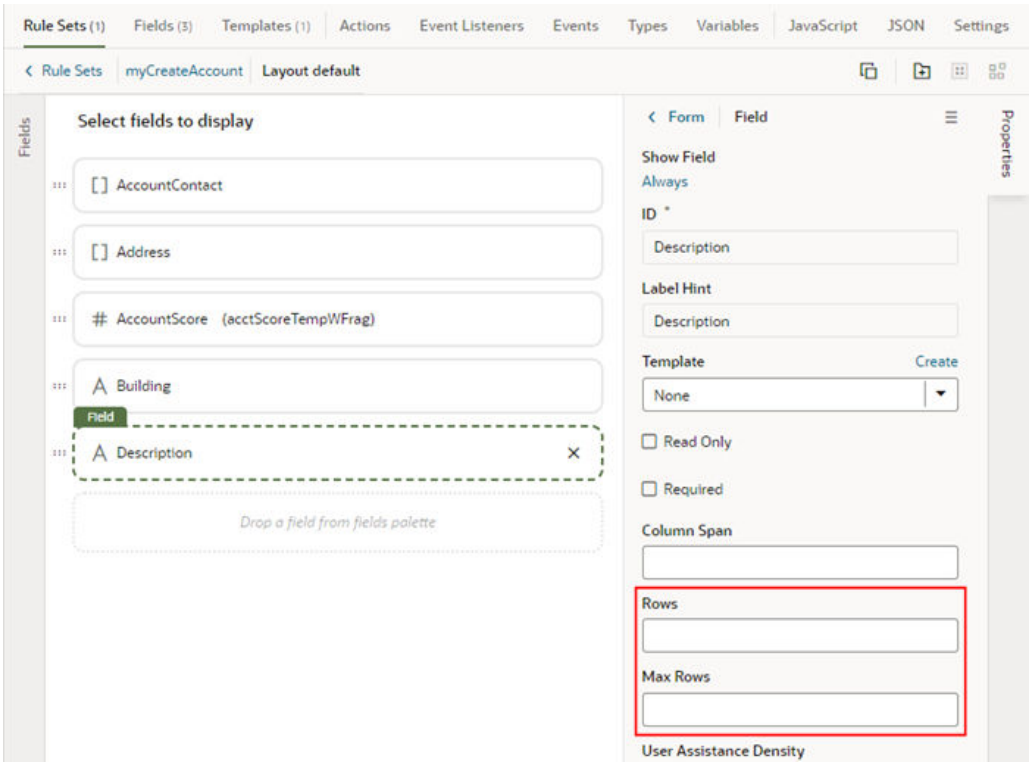
Notice that the field now has a blue dot bar next to it to indicate the field has been modified.

When you switch the format to `long-text`, two additional properties are displayed in the Properties pane.

3. Set the **Rows** property to the number of rows to display in the form by default.
4. Set the **Max Rows** property to the maximum number of rows you want to be displayed in the form. The text area will expand to the Max Row number if needed. The maximum number of rows defaults to three if you don't set a number in the Max Rows property.



- To set a field to display as a text area in a particular form layout:
  1. In the Rule Set editor, open the layout and select the field in the center pane.
  2. Set the **Rows** property to the number of rows to display in the form by default.
  3. Set the **Max Rows** property to the maximum number of rows you want to be displayed in the form. The text area will expand to the Max Row number if needed. The maximum number of rows defaults to three if you don't set a number in the Max Rows property.



Add Converters and Validators to a Field

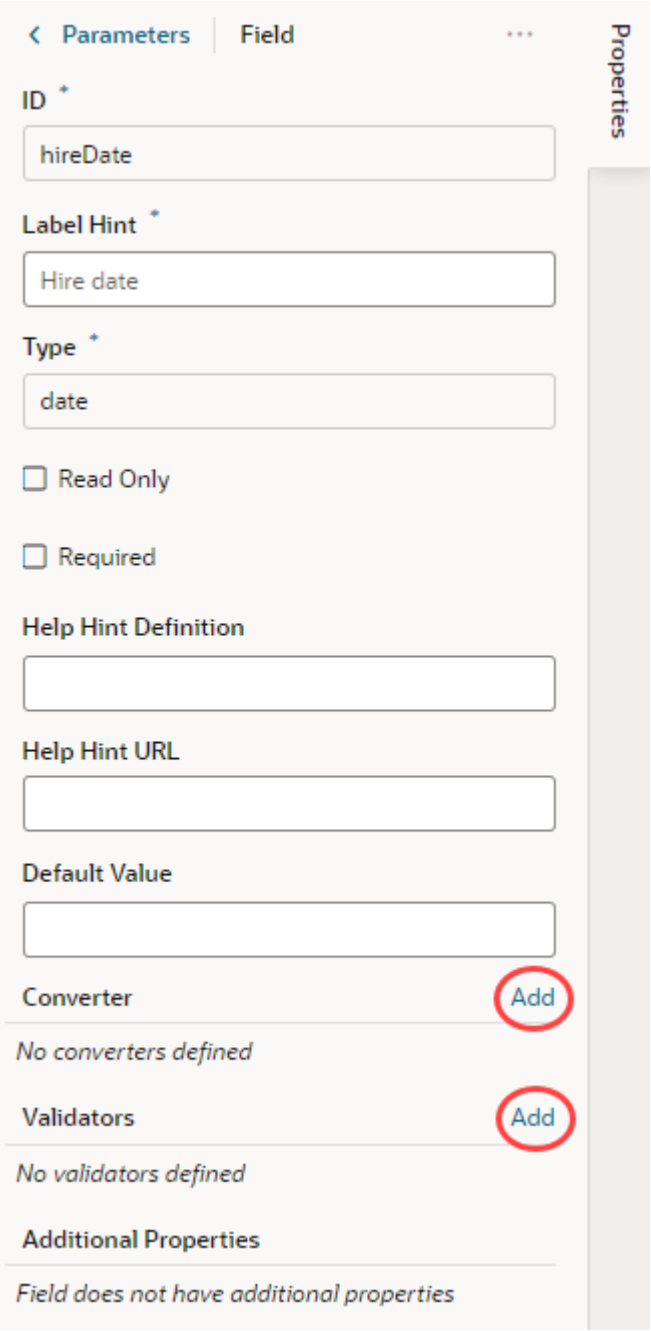
You can add converters and validators to a field, including some built-in ones provided by Oracle JET. You might want to add a converter to a field to change how the field's data is displayed in your page, for example, to display a date as month, day, and year instead of numerically. You might also want to add a validator to check if a value entered in a field is valid, for example, to check if a date is not earlier than the current date.

You can find details and examples in the Oracle JET Developer Cookbook:

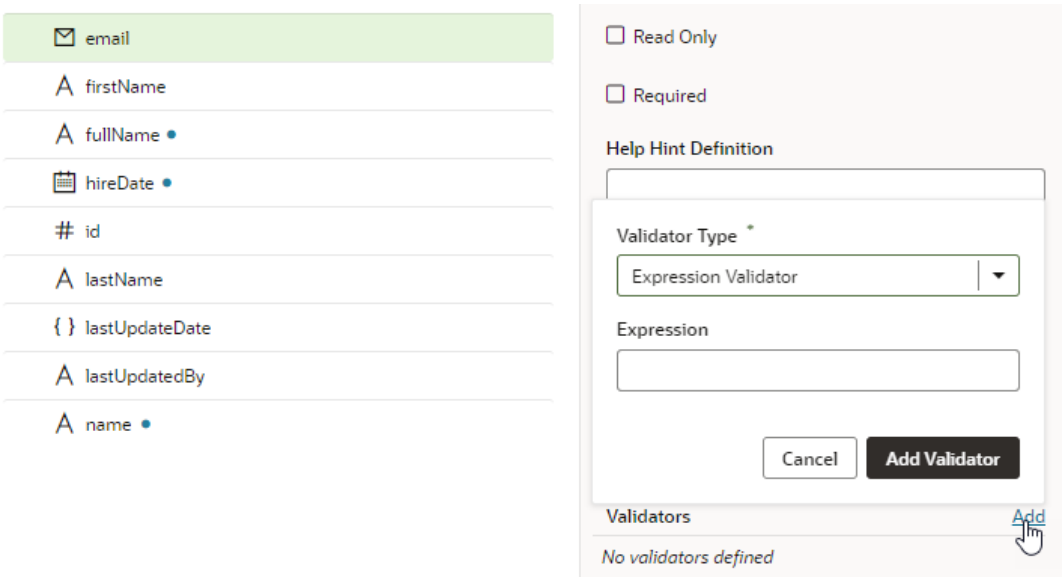
- Built-in Oracle JET converters
- Built-in Oracle JET validators

To add a converter or validator to a field:

1. In the **Fields** tab, select the field you want to work with.
2. In the Properties pane, click **Add** next to Converter or Validators, then select one from the list.

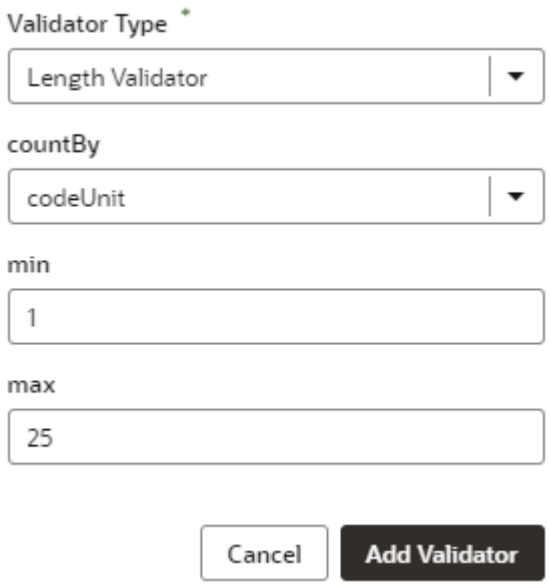


A default option is selected based on the field's type. For example, the default validator for an employee's Email field that uses the Email format is the Expression Validator:

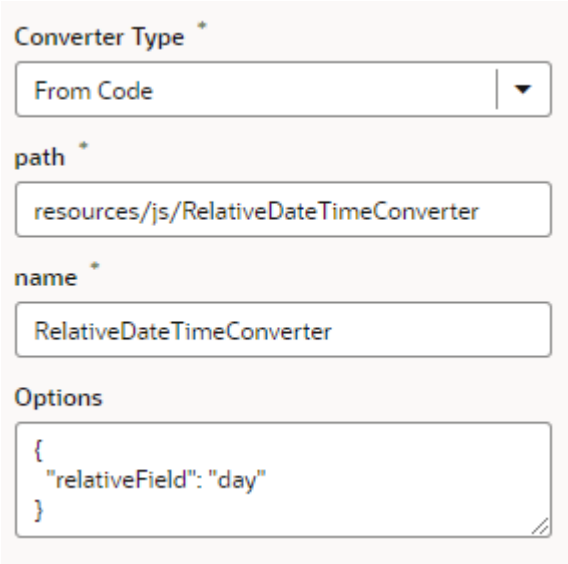


3. Change the type if needed, enter additional details, then click **Add Validator** or **Add Converter**.

   The details you'll need to enter will depend on the validator or converter you use, so you might need to consult the samples and documentation for the specific options. Use the JSON editor if you want to add options other than those shown on the UI. For the Length Validator shown here, the options specify how to count the characters and the minimum and maximum string lengths allowed:

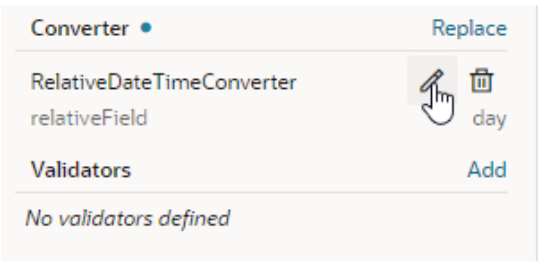


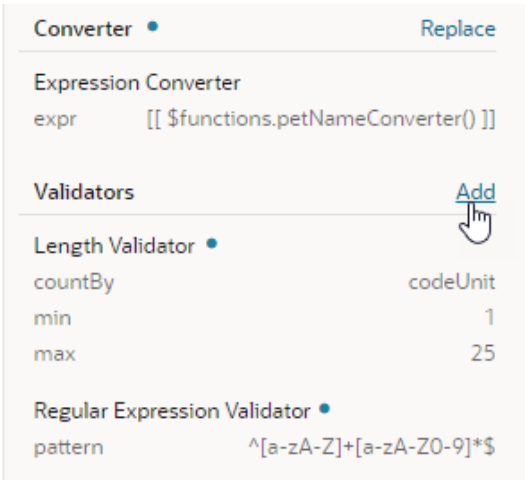
   You can also create your own validator or converter by selecting the **From Code** option. With this type, the **path** field specifies the location of a JavaScript file that implements the custom validator or converter; the **name** field specifies the name of the constructor; and the **Option** field specifies the options specific to the custom validator or converter, as shown here:



In this example, the RelativeDateTimeConverter JS file implements a converter with a constructor named `RelativeDateTimeConverter` and a `relativeField` option whose value can be, for example, `day`, `week`, `month`, and `year`. The implementation would convert a date value like `2014-01-02T20:00:00` to a relative date value, like `Today`, `Tomorrow`, `This Week`, `Next Week`, and so on, based on the value of the `relativeField`.

It's possible to update your validator and converter options any time after they've been added. Hover near the validator or converter name, click , and make your updates. You can add as many validators as you want, but a converter can only be replaced because a field can have only one converter.





Add a Dynamic Container to a Page

A dynamic container lets you display content in individual *sections*, or logical regions of the page based on conditions that are evaluated at runtime. You define the UI elements or

components displayed in each section, and then set the display logic conditions to determine which sections are displayed.

The display logic for determining what's displayed in a dynamic container is defined using *cases*. A case is similar to the rule sets used in dynamic forms and tables, but instead of selecting which fields to display, you select which sections to display. When you define a case, you specify the conditions for the case, and the sections you want displayed in the container when that condition is met. For each case, you can display any number of sections, so if a case defined two sections, you'd see two sections in the container.

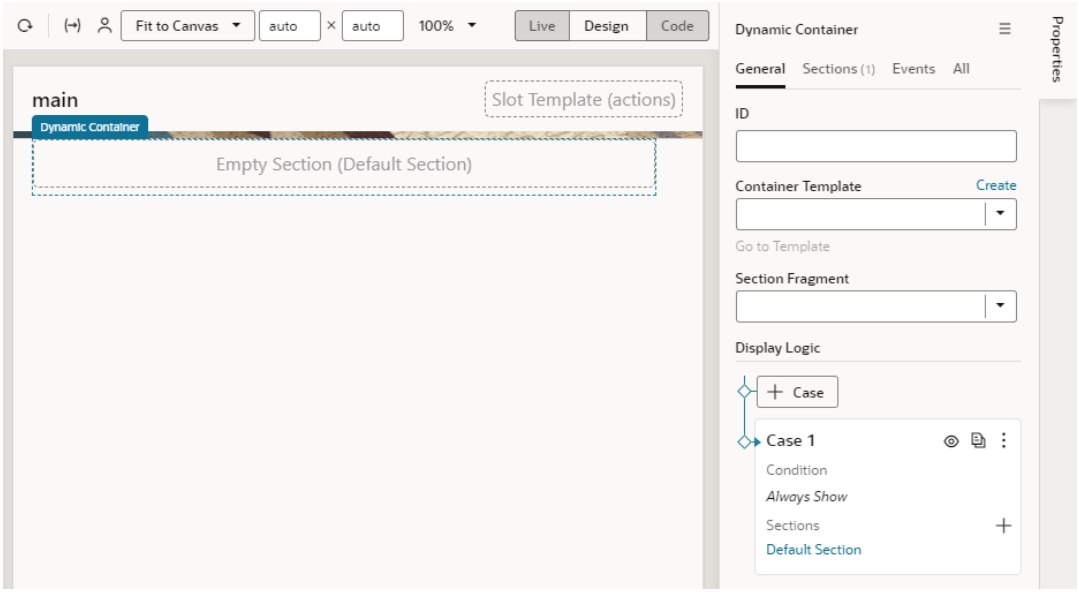
The easiest way to learn how to configure a dynamic container is to examine a real use case. Let's say you want to create a page that lets users toggle two layouts, one showing a form for adding an employee and another showing a table of employees. To do this, you'd create a dynamic container with two sections: one for a dynamic form and another for a dynamic table. You'd then add a button that the user can click to toggle the sections displayed in the dynamic container.

> **Note:**
>
> Before you begin, make sure you've defined a data source, such as a business object or a service that calls a data object through a service connection. See Work With Business Objects or Manage Service Connections.

To add a dynamic container component to a page:

1. With your page open in the Page Designer, drag the Dynamic Container from the Components palette onto your canvas.

   In this image, you can see one section (`Default Section`) automatically created for you when you added the dynamic container. The `Default Section` section doesn't have any content yet, so you see the section's name on the canvas instead.

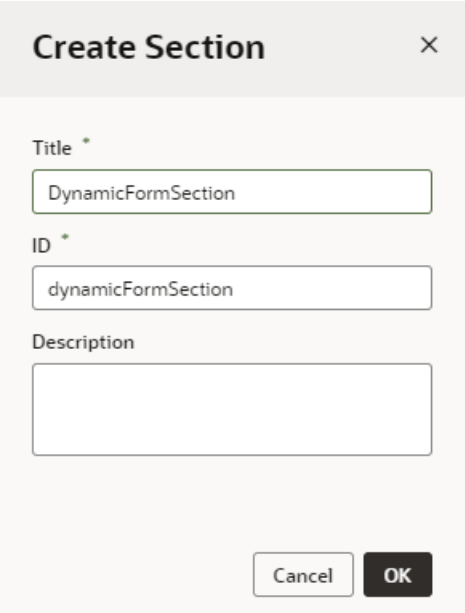


2. Create sections for the dynamic container. In our example, we'll create two sections, one for a dynamic form and another for a dynamic table.

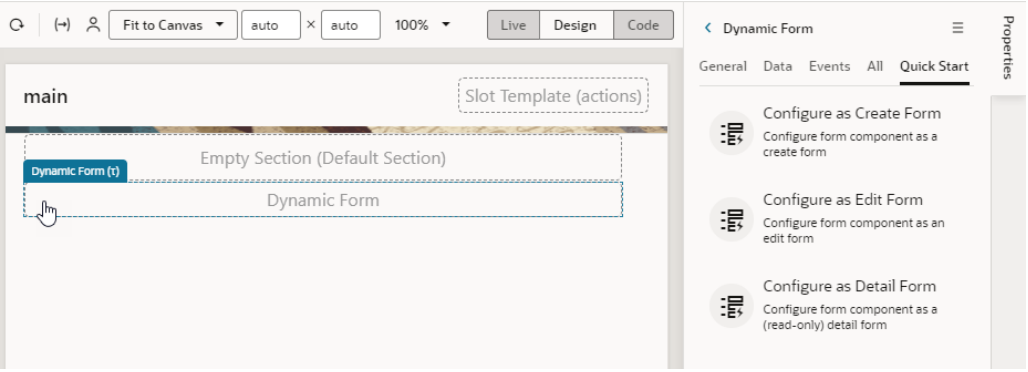
a. In the **Display Logic** section in the Properties pane, click + next to Sections under Case 1, then click **+ New Section** to create a new section.

> **Tip:**
>
> You also have the option to use existing fragments as sections in your dynamic container. Fragments tagged as `pageContent` (default tag) are automatically listed in the **Add Section** drop-down (see Add Fragments as Sections in a Dynamic Container).
> You'll also see existing but unused sections, in addition to the Default Section.

b. In the Create Section dialog box, give the section a name (for example, `DynamicFormSection`) and click **OK**.

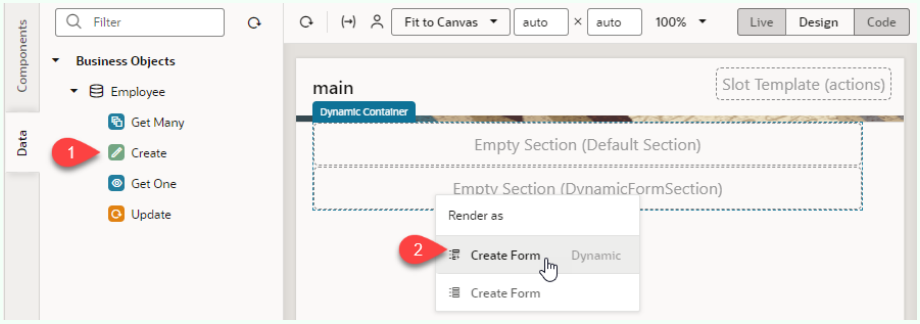


c. From the Components palette, drag the Dynamic Form and drop it onto the section on the canvas, select the Quick Start you want to use in the Properties pane, and follow the prompts to associate the dynamic form with a data source and a rule set. Click **Finish**.



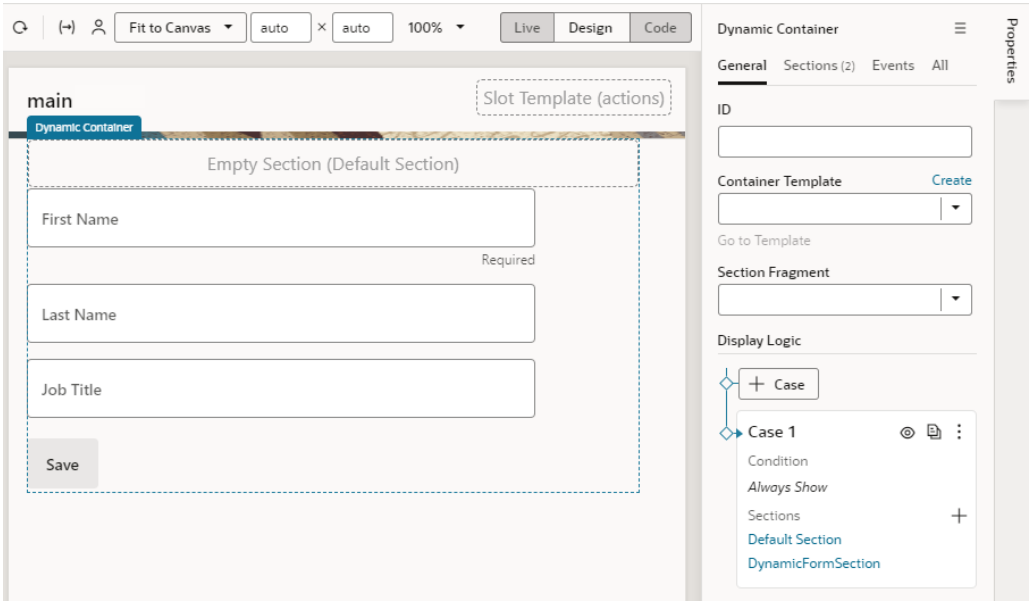
Tip:

Use the Data palette to try an alternative design approach, where you start with your data source's REST endpoints and leverage components that Visual Builder deems best to optimally display your data. For example, to display a create employee form, you can drag the **Create** endpoint from the Data palette onto the canvas. When the **Render as** pop-up appears, select an option (in this case, **Create Form Dynamic**), then follow the steps in the Configure as Create Form quick start to set up your form.



If you used the quick start to bind your dynamic form to a data source, the form will be rendered on the page with the fields you selected. You can change the form's fields by editing the dynamic form's layout in its rule set. If the form isn't bound to a data source, the section will contain a placeholder for the form because there won't be any data to render the form.

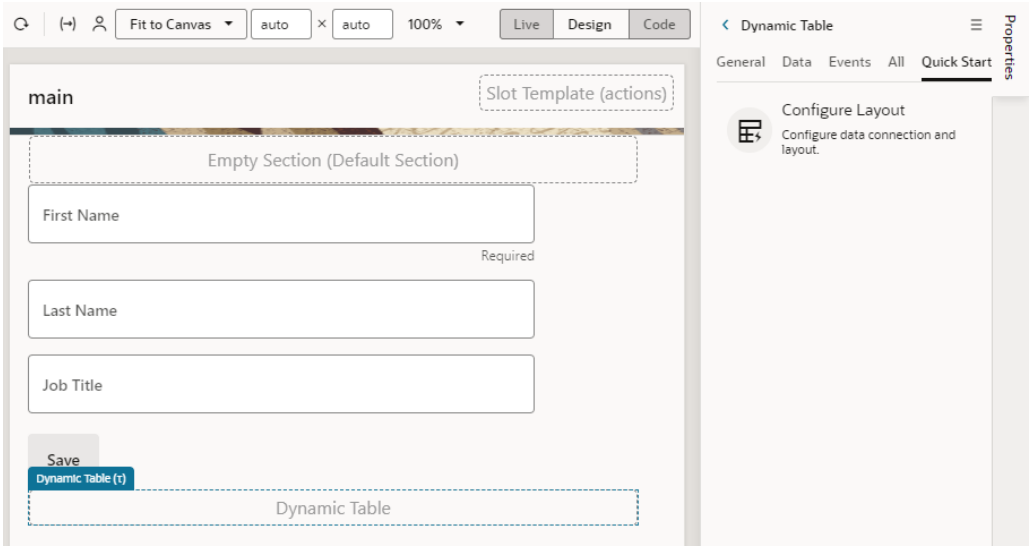
Click **Dynamic Container** in the Structure view to see the dynamic form added as a component to the page and its properties displayed in the Properties pane:



d. In Case 1 in the Properties pane, click + again, then click **+ New Section** to create more sections; in our case, a section for a dynamic table.

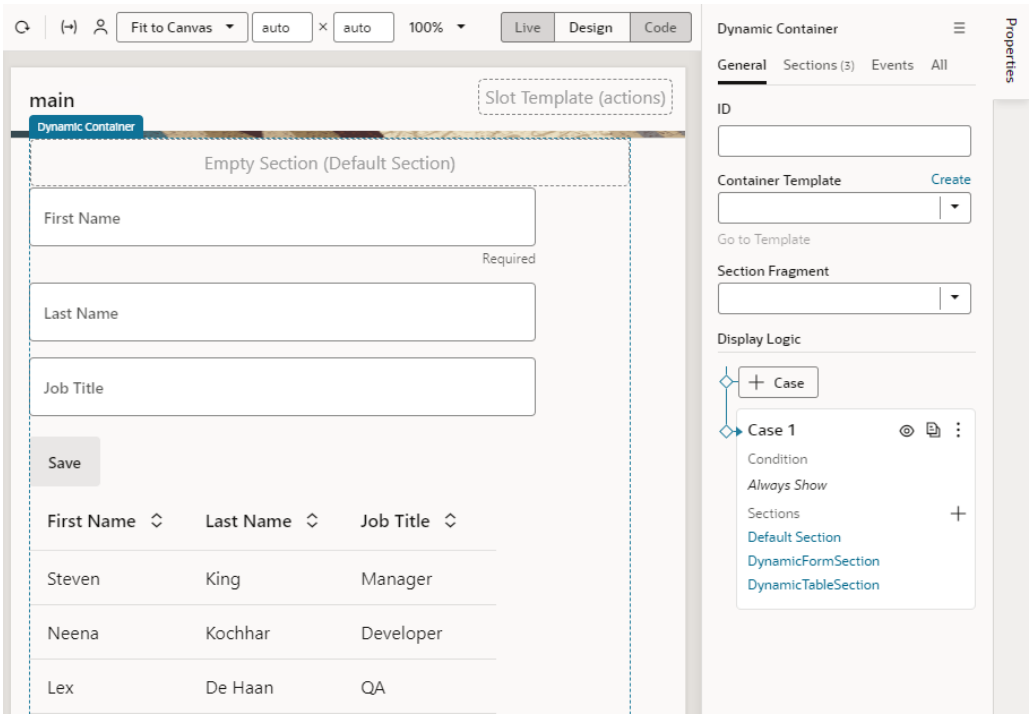
e. In the Create Section dialog box, give the section a name (for example, `DynamicTableSection`) and click **OK**.

f. Drag the Dynamic Table from the Components palette onto the new section, click the **Configure Layout** quick start in the Properties pane, and follow the prompts to complete the setup. Click **Finish**.

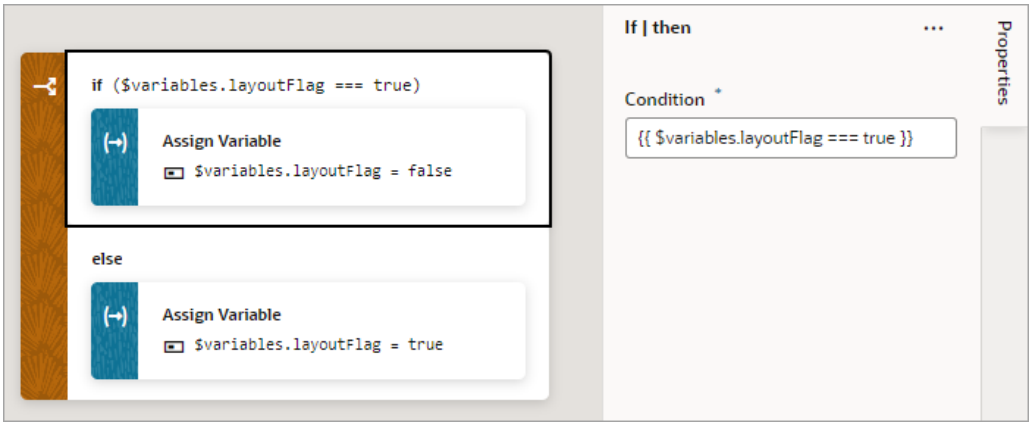


> **Tip:**
>
> Similar to using the Create endpoint to display a create form, you can use the Get Many endpoint to display your employee data in a dynamic table. From the Data palette, drag the **Get Many** endpoint onto the canvas, choose **Table Dynamic** in the **Render as** pop-up, then follow the steps in the Configure Layout wizard to set up your table.

After using the quick start to bind the dynamic table to a data source, the Page Designer will render the table with the fields you selected. Again, if you don't bind the component to a data source, your dynamic table won't have any data to actually render on the page. Click **Dynamic Container** in the Structure view to see the newly added section:



3. Set up the UI component and add the case logic to display your sections. Here, we'll add a Button component and a page variable to toggle the sections in the dynamic container.
    a. In the **Variables** tab, create a variable that you can use in your conditions. For example, a Boolean-type variable called `layoutFlag`, with the default value set to `true`.
    b. In the **Page Designer** tab, drag a Button component from the Components palette and drop it just above the Dynamic Container. Change its Text in the Properties pane to `Toggle Layout`.
    c. In the Button's **Events** tab, add an event listener for the `ojAction` event.
    d. Set the name of the new action chain as `ToggleLayout`, then build the action chain:
        - Drag and drop an **If** action onto the canvas and set its **Condition** to `[[ $page.variables.layoutFlag === true ]]` in the Properties pane.
        - Drag and drop an **Assign Variable** action onto the **Add Action** area of the `if {{ $page.variables.layoutFlag === true }}` block. Select **layoutFlag** from the **Variable** list and set the **Value** to `false`.
        - Drag another **Assign Variable** action and drop it onto the **Create Branch** area at the bottom of the `if {{ $page.variables.layoutFlag === true }}` block (the Create Branch area appears when you hover over the If action with another action). Select **Assign Variable** in the **else** block, then select **layoutFlag** from the **Variable** list and set the **Value** to `true`.

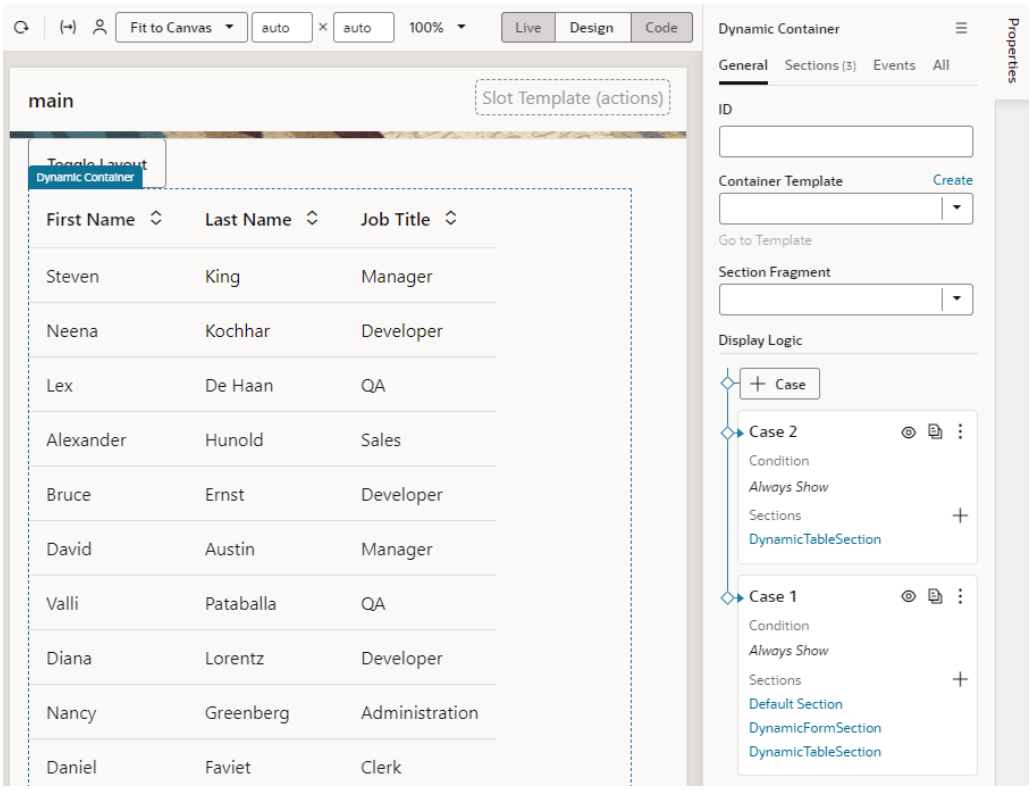


4. Return to the Page Designer to create another case in the dynamic container.
   a. Select the Dynamic Container component to view its properties in the Properties pane.
   b. In the container's **General** tab, click **+ Case** to create another display logic condition, called `Case 2`. You can use to update the name as required.

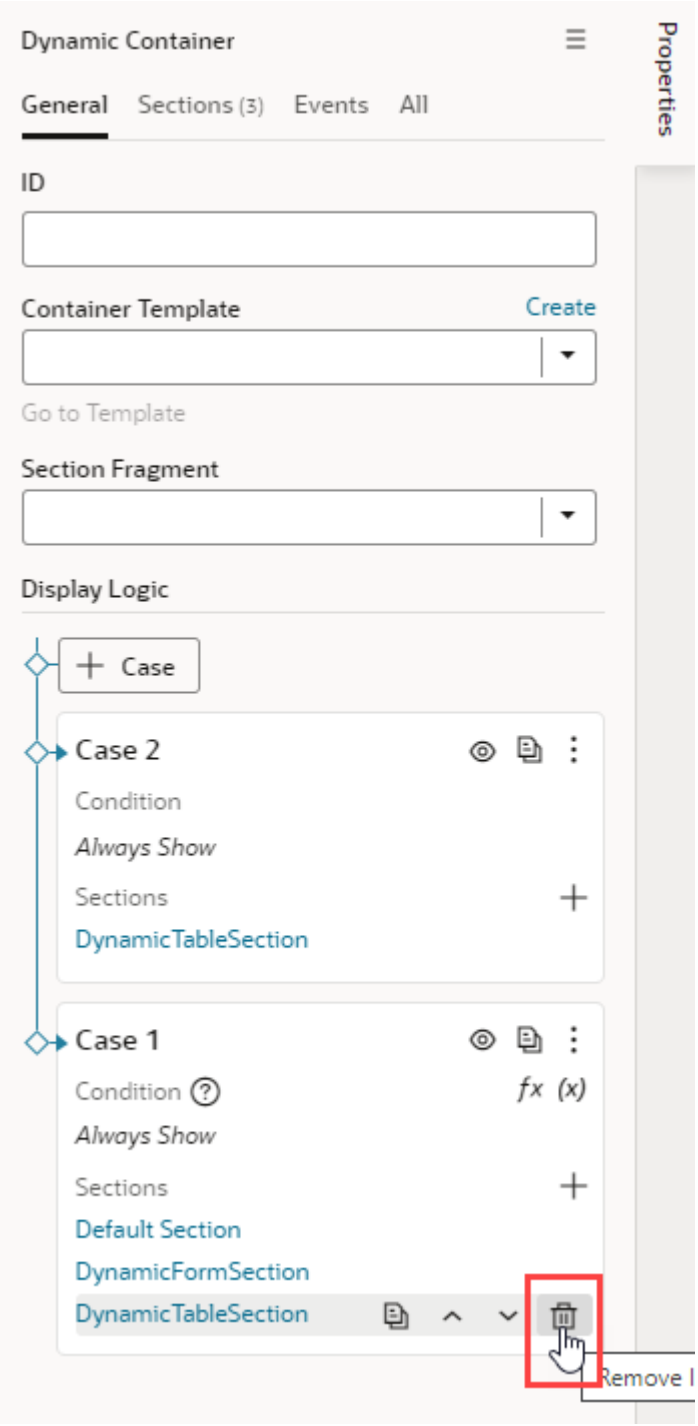
      You can also duplicate a case if you want to create a new case containing the same sections.

      When you create the new case, it is added as the first case in the Display Logic tree, before Case 1. Remember cases are evaluated from the top down. So in our example, at runtime the conditions for Case 2 will be evaluated first.
   c. In Case 2, click **+** next to **Sections** and select the sections you want to display, `DynamicTableSection` in our example. Note how this previously defined section is available for selection.

      When you do this, you'll only see Case 2 rendered in the Page Designer, so in the dynamic container you'll only see `DynamicTableSection`, which contains the dynamic table:

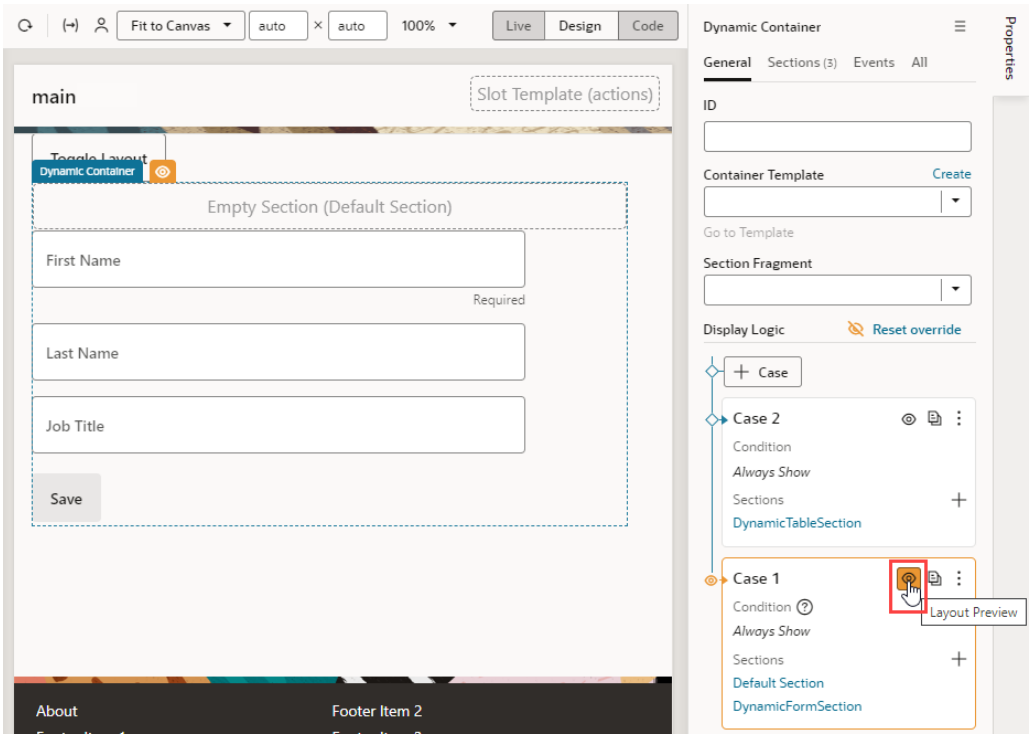


d. In Case 1, click next to `DynamicTableSection` in the list of sections to remove it from Case 1.



5. Preview how the different cases in your dynamic container will look using Layout Preview. Layout Preview forces a case to be temporarily rendered in the page for design purposes, regardless of its position in the case order and its condition.

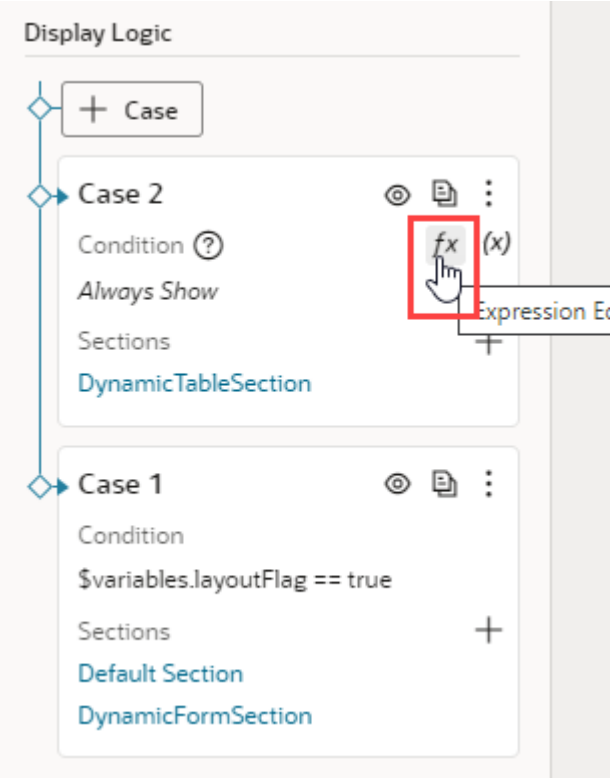
   a. Click the icon for Case 1 in the Display Logic section:



Now the sections in Case 1 (`Default Section` and `DynamicFormSection`) are rendered on the canvas instead of the section in Case 2 (`DynamicTableSection`).

b. Click **Reset Override** (or again) to remove the preview.

6. In the Condition field for `Case 1`, click to open the Expression Editor and change the default condition `Always Show` to the expression `$variables.layoutFlag == true`. In the Condition field for `Case 2`, enter `$variables.layoutFlag == false`:

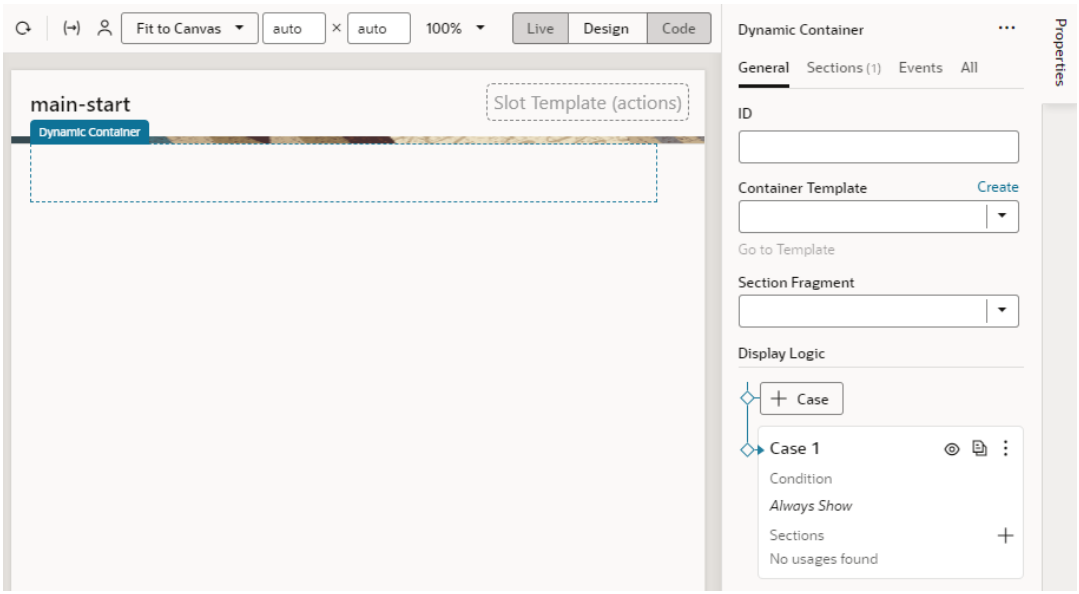


7. Click **Preview** to preview the page, then click **Toggle Layout** to toggle the sections displayed in the dynamic container.

Add Fragments as Sections in a Dynamic Container

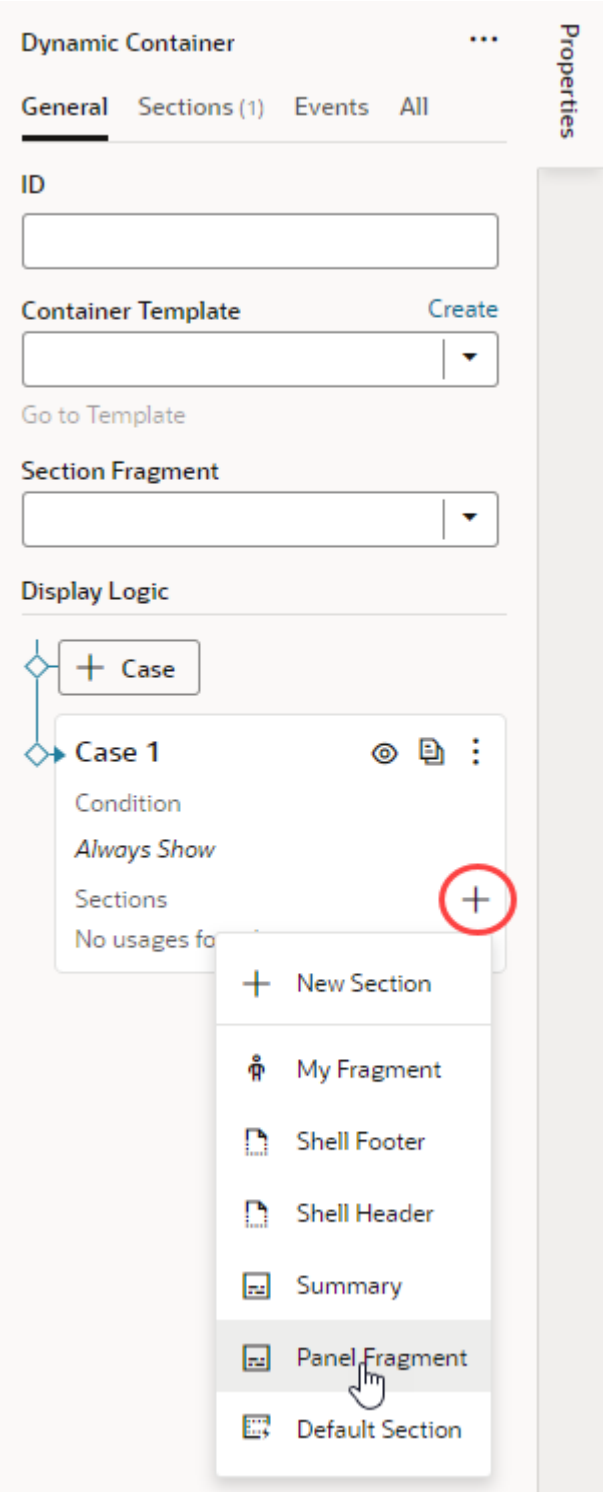
You can use existing fragments to define sections in a dynamic container. You also have the option to define a fragment as preferred content for sections.

1. With your page open in the Page Designer, drag the Dynamic Container from the Components palette onto your canvas. Here's a dynamic container with the `Default Section` removed.



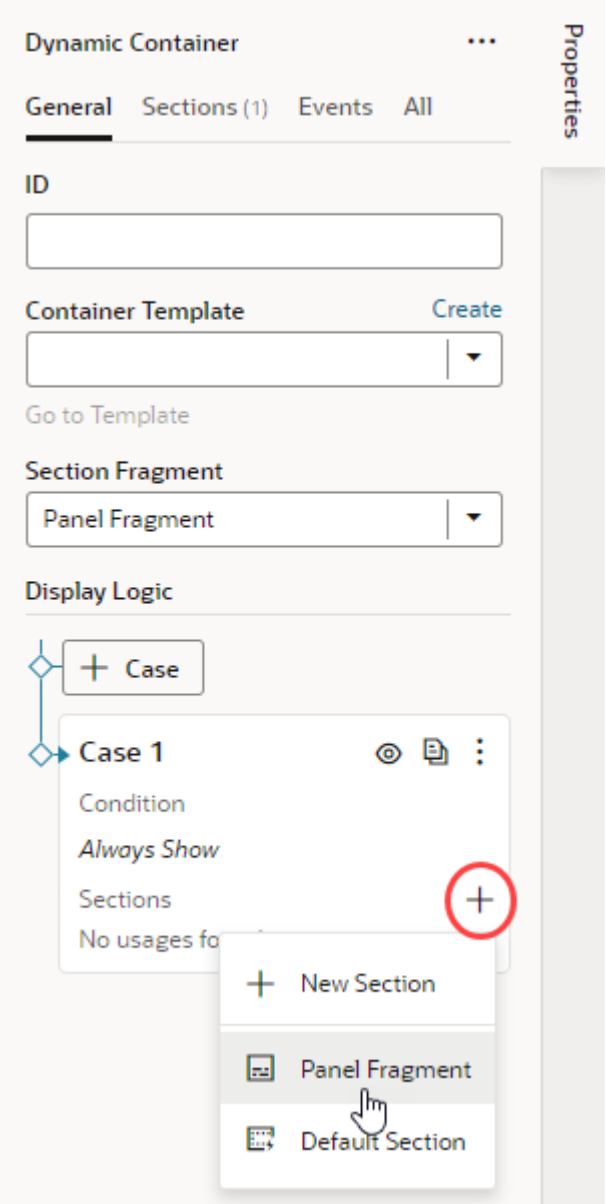
2. Add a fragment as a section to the dynamic container. Make sure the fragment already exists and is tagged `pageContent`, so it becomes available for selection in a dynamic container.

   - To add an existing fragment as a section, in the **Display Logic** section, click + next to Sections under Case 1 (or other Cases as defined), then select a fragment in the drop-down list. This list displays all `pageContent` fragments (default tag), including the `Shell Header` and `Shell Footer` created automatically by web app templates. Other unused sections (including the `Default Section`) are also listed.

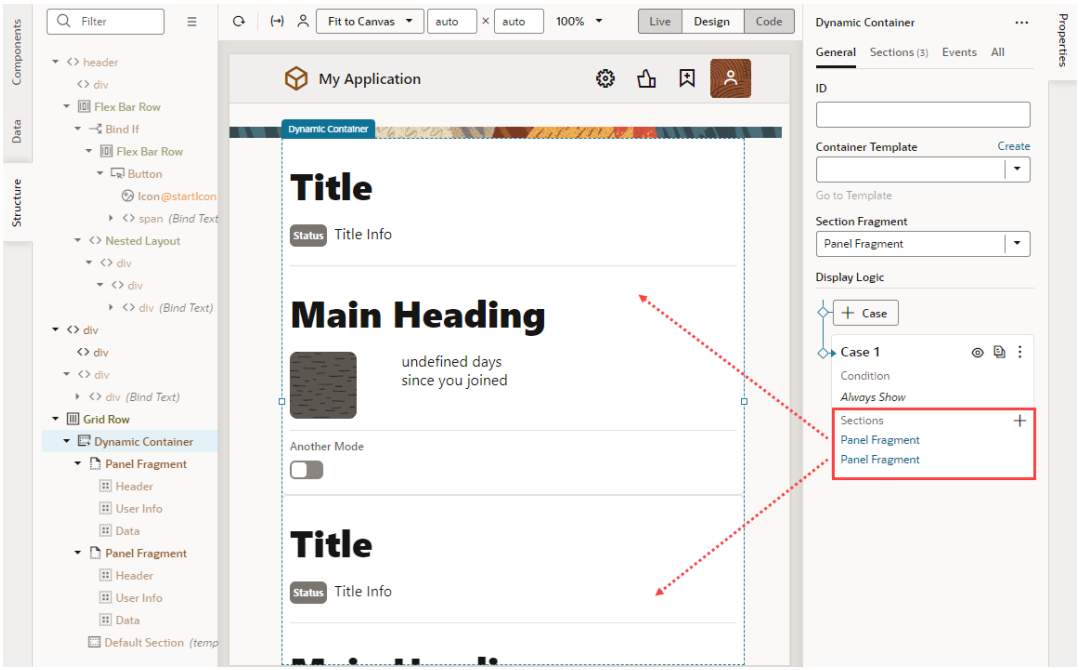


- To set a particular fragment as preferred content for all sections in the dynamic container:
  a. From the **Section Fragment** list, select the fragment you want to make available to a section. This list displays all `pageContent` fragments (default tag).
  b. In the **Display Logic** section, click + next to Sections under Case 1 (or other Cases as defined), then select the preferred fragment in the drop-down list. Only

the fragment specified as the Section Fragment can be added to the section (in addition to any unused sections).



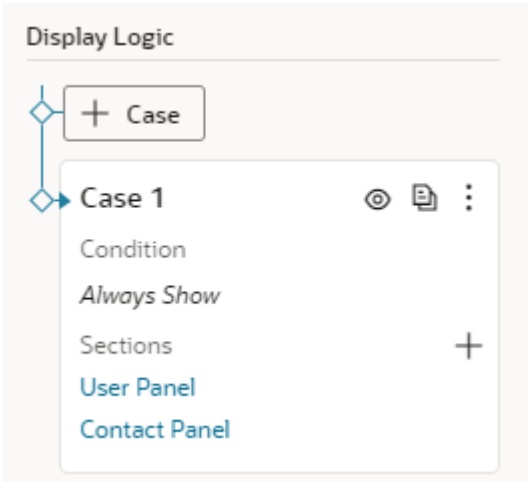
Let's say you have a foldout layout with a dynamic container and you want to use fragments as sections to define different foldout panels. You would simply select your fragment (`Panel Fragment`, in our example) as a section. Here's an example of a panel fragment used to add two sections, each to define two panels in a foldout layout:



3. Select each section and configure its content (fragment) to suit your requirements. For example, select the first `Panel Fragment` and change its title to make the panel more identifiable, say, to `User Panel`. Then click **Fragment Container** in the Structure view to view the contents of the fragment being used as the `User Panel` and configure it as desired.

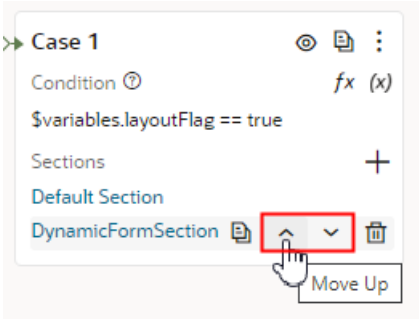
   Repeat the steps to configure the second `Panel Fragment`; this time, say as `Contact Panel`.

   If you click **Return to Page**, your dynamic container now displays the two panels as configured:



Re-Order a Dynamic Container's Content

Besides defining new cases and sections for a container, you can also change the order the sections are displayed in the container. Just use the Move Up and Move Down arrows under Sections for a case:



You can also use the icon to remove a section from a case. The removed section will still be available to use in other cases for dynamic containers in the page.

Guidelines for Working with Sections

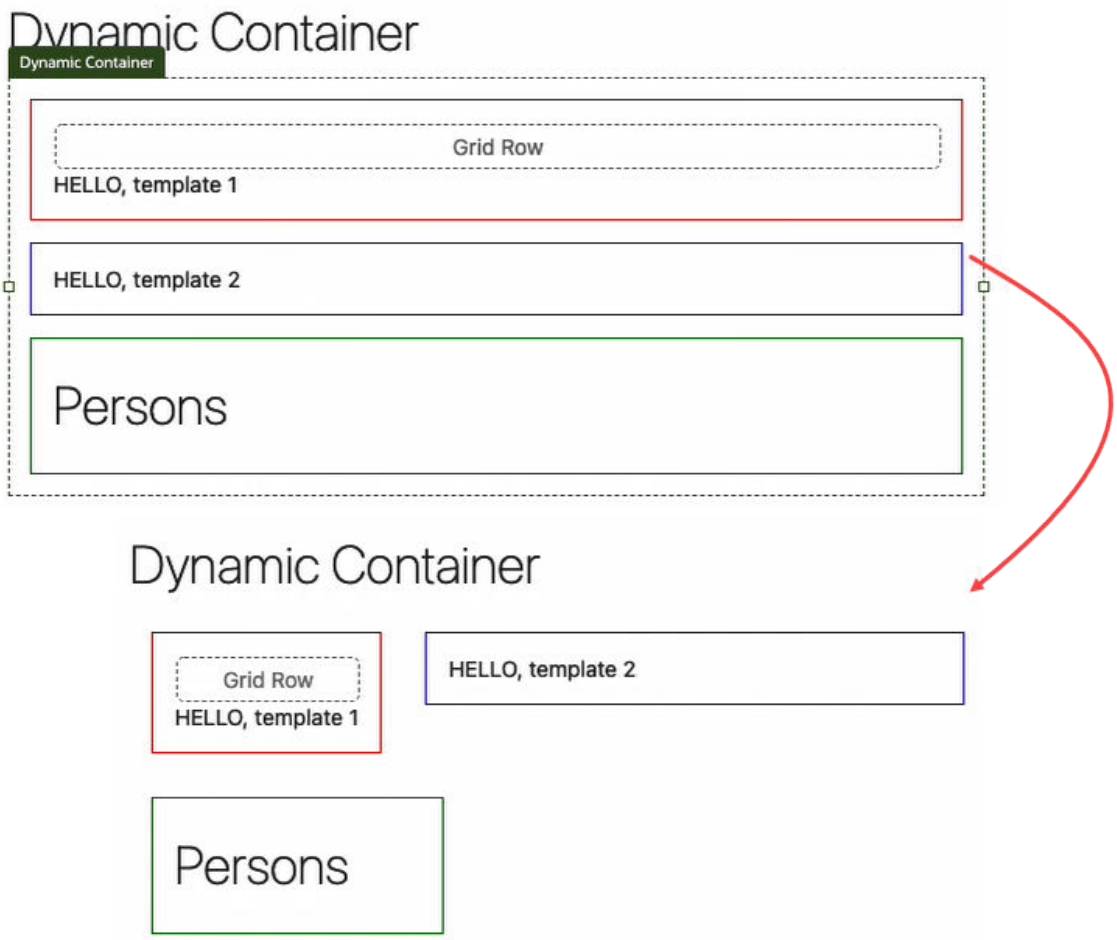
Here are some things to keep in mind while working with sections:

- A section is the full width of the container. That is, while you can't have two sections side-by-side, the container can stretch to any height required to accommodate all the sections you define. If you want to lay out sections side-by-side, you'll need to change the container's default layout.
- In addition to simple components like text fields and images, you can also add more complex components to your sections. For example, you might include fields for displaying data from a service, a button that starts an action chain in your visual app, a page fragment, or even other dynamic components.
- Components in a section can access variables and constants, and trigger events to start action chains.
- When working with sections, sometimes it’s easier to work in the Structure view, which helps you more readily visualize the position of components. You can also drag components within the Structure view to reorganize them.

Change a Dynamic Container's Layout

Dynamic containers display sections vertically by default. That is, sections are rendered in a single column, one on top of the other. You can change this layout to show them instead in a row using *container templates*.

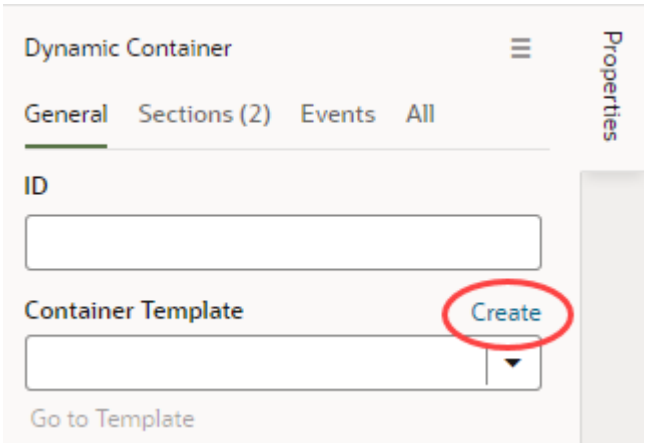
Container templates help lay out sections in any layout other than the default. You can use them to arrange sections within a dynamic container however you like, even adding other components, including dynamic components and fragments. So you could include a fragment in your container template, and that fragment could include a dynamic form and other components.



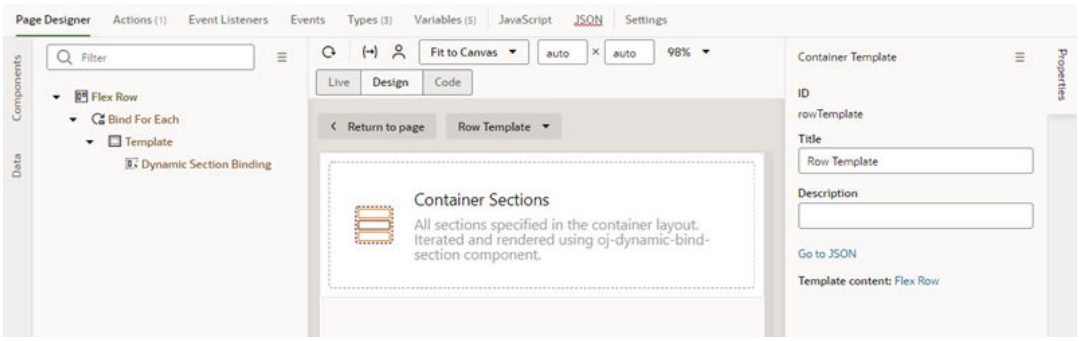
The preceding image shows a dynamic container's sections stacked vertically. Once you create a container template and switch its layout, the sections are placed next to each other. When they run out of space on the row, they automatically wrap to the next row (as shown at the bottom of the image).

To change the layout of sections in a dynamic container:

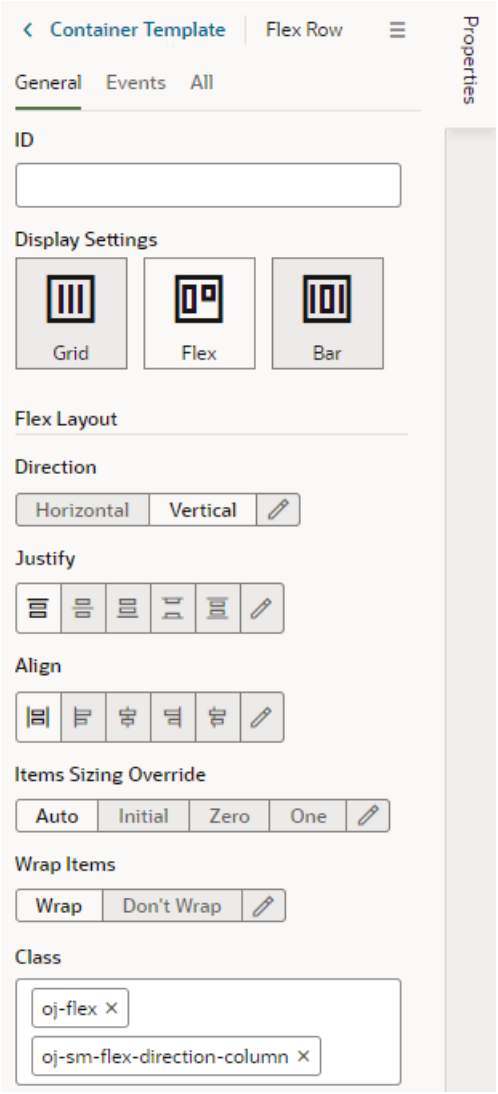
1. With your page open in the Page Designer, select the Dynamic Container component to view its properties in the Properties pane.
2. In the General tab, click **Create** next to Container Template.



3. Enter a name for the container template (for example, `Row Template`) and click **OK**, then click **Go to Template**.
4. When the template opens in the template designer, click **Flex Row** next to Template Content in the Properties pane.



5. In the Flex Row's properties, look for the **Direction** field, which by default is set to Vertical.



6. Select **Horizontal**. Take note of the `oj-sm-flex-direction-column` class, which sets the direction to a column and is included by default, is removed from the Class field.
7. Click **Return to Page** to see your sections laid out horizontally in a row, rather than vertically in a column.

Create Fields For a Layout

If you'd like to use a field in your layout that isn't defined in your data source (either a business object or a service definition), you can create fields that you can set to variables, or to expressions that reference other fields.

If the existing fields don't meet your needs, you could create *calculated fields* or *virtual fields*. You would use a calculated field when you want to use an expression, set a default value, modify labels, and set read-only and required properties. You would use a virtual field if you want a field that has editable sub-fields.

> **Note:**
>
> The fields you create are only used in your layouts; creating a field doesn't create a field in your business object or your Oracle Cloud Application, and doesn't change the service definition. For details on creating fields in your business object, see Create and Edit Business Objects. For details on creating fields in an Oracle Cloud Application, see Define Fields in *Configuring Applications Using Application Composer*.

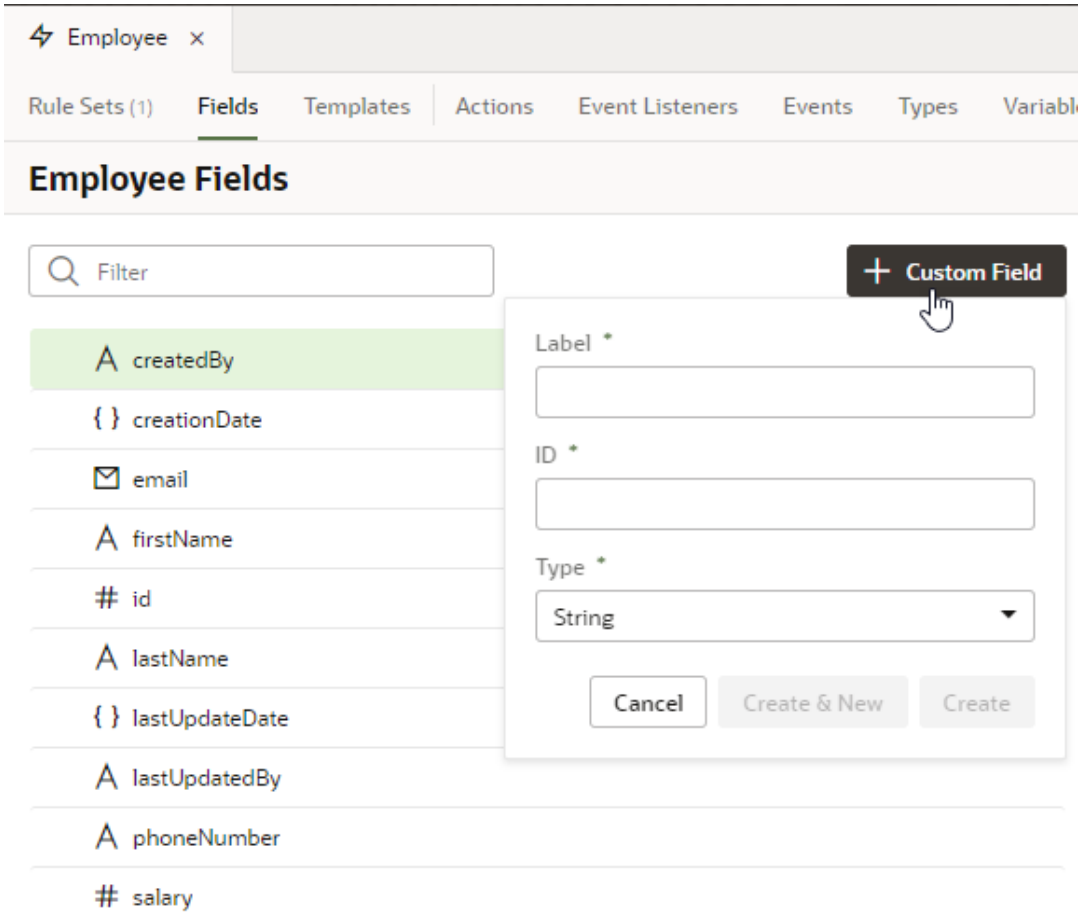
Create a Calculated Field

You can use a calculated field when you want to have a single field in your layout that, for example, contains some static string or an expression that is computed from the values of other referenced fields or objects.

Suppose your data source has separate fields for a user's first name and last name. You could create a custom field that combines these fields into a single field called `fullName` and use that in your layouts instead. The value of this new field is calculated using an expression like `[[ 'Name: ' + $fields.firstName.value() + $fields.lastName.value() ]]`. In a calculated field, referenced fields defined in the expression are read-only, so they can't be edited in a layout.

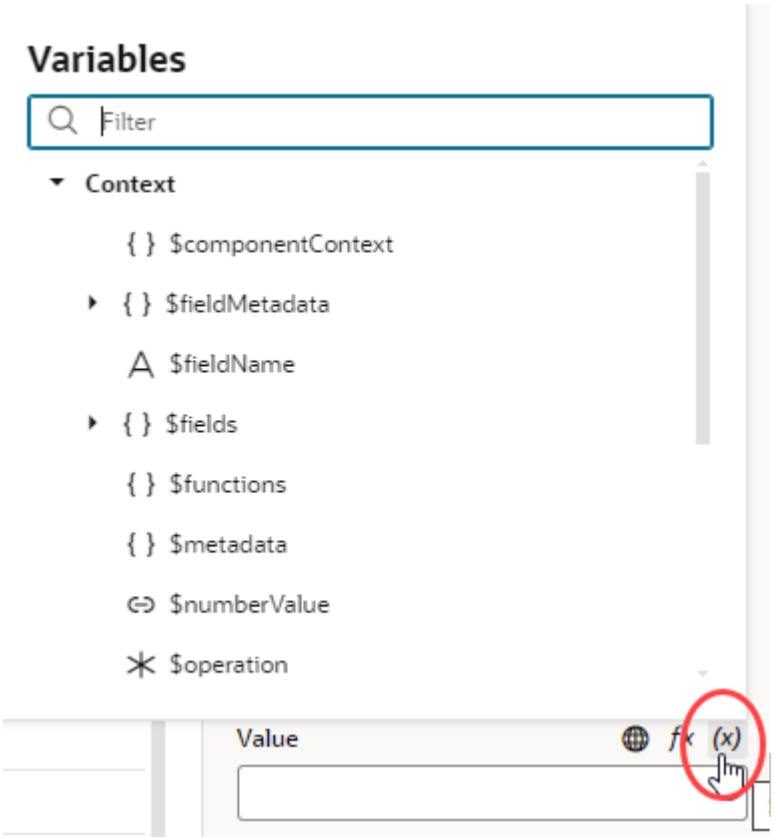
To create a calculated field:

1. Open the dynamic table or form you want to work with in the **Layouts** tab.
2. Click the **Fields** tab, then **+ Custom Field**.



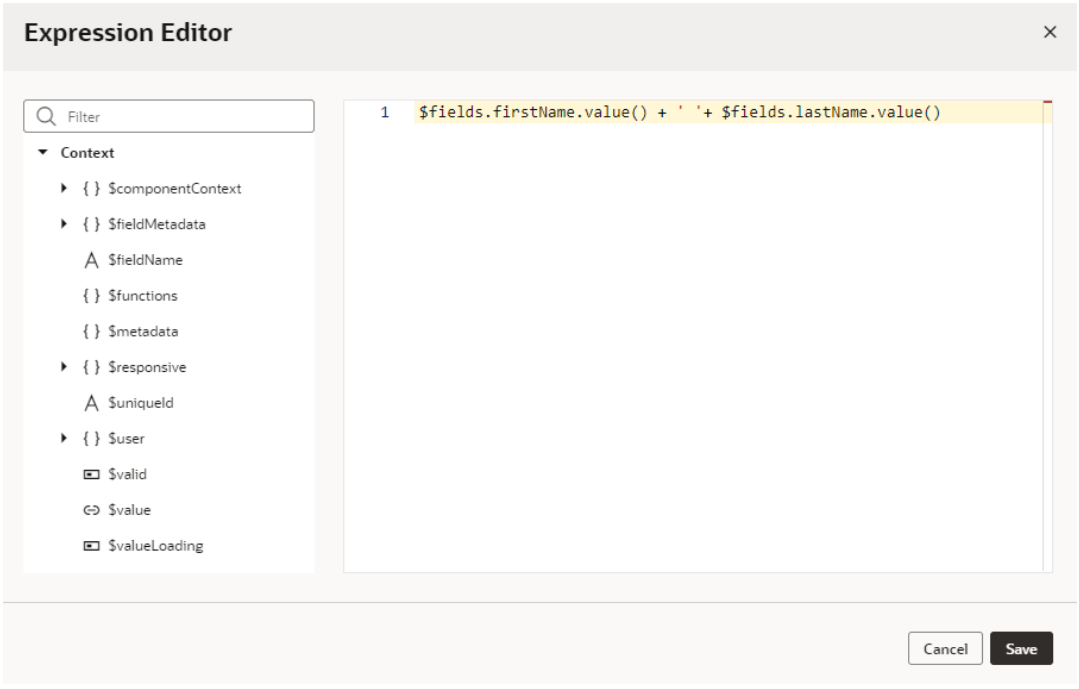
3. Enter a label for the field (the field's display name). When you enter the label, a suggested ID is generated for you. The ID can't be changed later.
4. Select the field type. When selecting a type for a calculated field, you should consider the types of the referenced fields you'll include in the expression.
5. If you want to create an expression and use an existing field, click **Add** next to Referenced Fields, then select a field in the list. Click **Add Field** to add it.
6. Define an expression in the Value property. The expression can include variables, static strings, and referenced fields.

   If you want to use a single variable, click to open the Variables picker.



If you want to use an expression, click fx to open the Expression Editor. In the Expression Editor, you can select field variables in the Variables pane to add them to your expression. You can also add text strings to your expression by typing in the editor. Click **Save**.

For example, here's an expression that combines the `firstName` and `lastName` fields:



The expression you create in the editor is added to the Value field, for example:

```
[[ $fields.firstName.value() + ' ' + $fields.lastName.value() ]]
```

You can also use the `$fieldmetadata` variable to access field-level metadata. For example, to invoke a function to calculate a field's default value based on its metadata, your expression might look something like this:

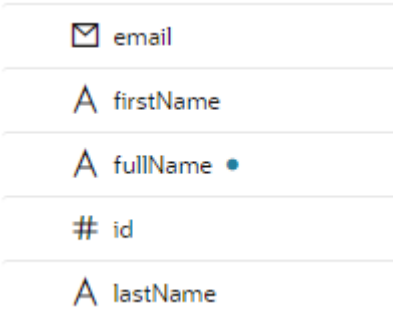
```
[[$functions.getDefaultValue($fieldMetadata, $componentContext)]]
```

where `$fieldMetadata` represents the metadata of the field.

To write efficient expressions that handle situations where a referenced field might not be available or the field's value could be null, see How Do I Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null?

7. Optionally, you can click **Add** next to Converter and Validator to add suitable built-in convertors or validators, or create a custom one. If you're using a referenced field, you might want to add converters or validators so that, for example, dates are formatted the way you want, or to make sure a string in a field is not too long.

Your custom fields (and any fields that you have modified, for example, in the Properties pane) are indicated by a gray bar to the left of the field name. In this screenshot, you can see the gray bar next to `fullName`:



Create a Virtual Field

You might want to create a virtual field if you would like to combine multiple fields together into a single field that you can add to your layouts. For example, you can create a single field that combines several contact details stored in different fields in the layout. A virtual field is similar to a calculated field, except:

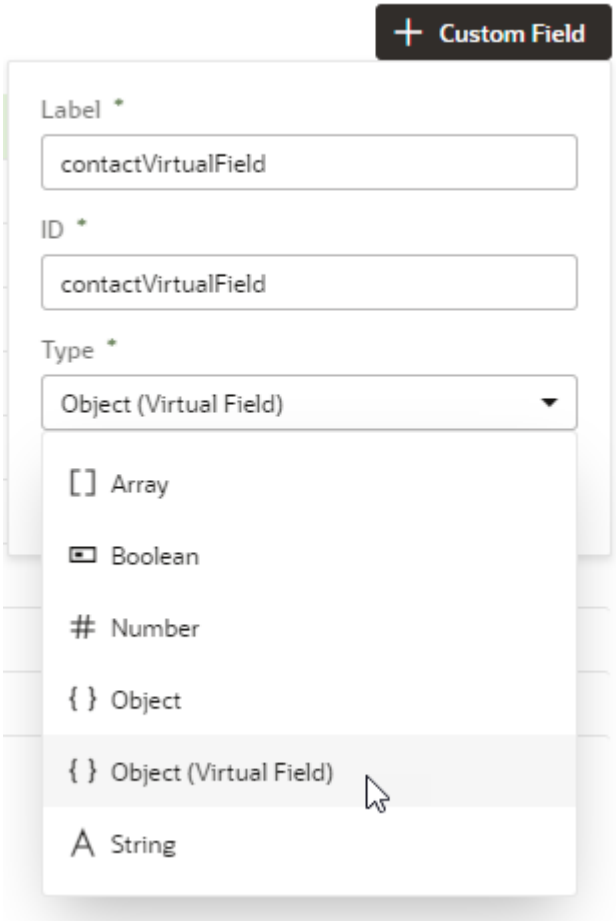
- the referenced fields can be edited in the layout; and
- the virtual field is rendered using a field template.

When you add a virtual field to a layout, you'll define a field template to display it. You'll need to create the field template if it doesn't exist. The template will contain components for each of the referenced fields that you want to display in the layout.

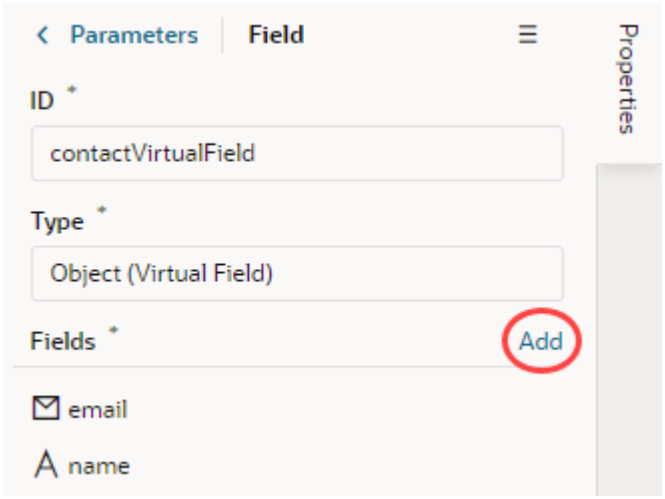
To create a custom virtual field:

1. Open the dynamic table or form you want to work with in the **Layouts** tab.

2. Click the **Fields** tab, then **+ Custom Field**.
3. Enter a label for the field (the field's display name) and select the **Object (Virtual Field)** type. Click **Create**.



4. In the Properties pane, click **Add** next to Fields and select the fields you want to include as reference fields. You can add any field in your layout as a reference field, including sub-fields of objects.



5. Select a field in the Sort By drop-down list to define the field that should be used for sorting when the virtual field is used in a table.

   Only one field in a virtual field can be used for sorting. For example, if the virtual field `FullName` consists of a `FirstName` and `LastName` field, select `LastName` if you want it to be used when the table is sorted by `FullName`. The Sort By field will be used for sorting regardless of how the virtual field is rendered in the table by the template. Remember, you need to use a field template to display a virtual field in a component.

   The table won't be sortable by the virtual field if you don't select a Sort By field.

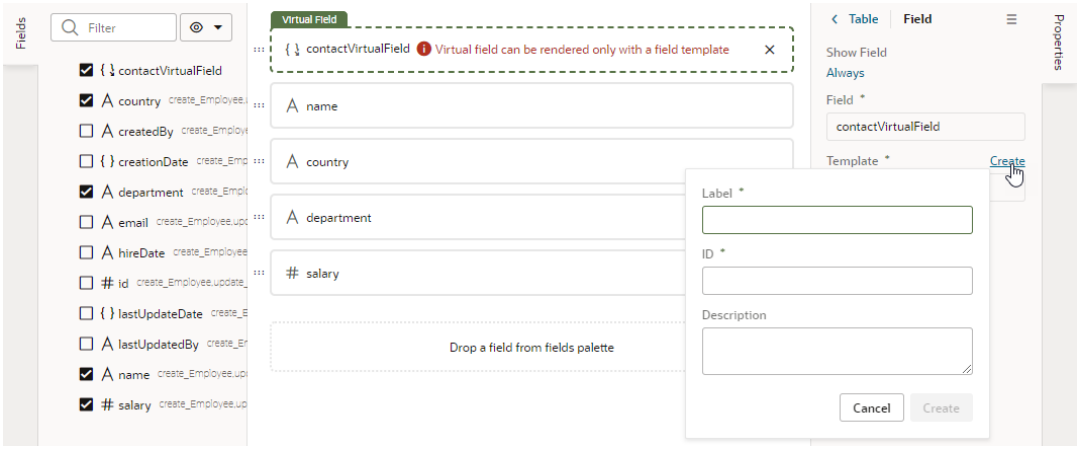
6. In the Rule Sets tab, open the layout in the dynamic component where you want to add your field.

7. Add the virtual field to the layout. You can drag it from the Fields palette into the center pane, or select it in the list and then adjust its position in the center pane.

8. While your virtual field is selected, define a field template for the virtual field when you add it to a layout.

   If a suitable field template for the virtual field already exists, you can select it in the Template drop-down list in the Properties pane.

   If no template exists, click **Create** and enter a name for the template in the Label field. Click **Create** to open the new template in the editor.



   In the template editor, add a component and define the properties for each referenced field in the virtual field that you want the template to display. Click **Return to layout** when you're finished.

   The template is applied to your virtual field.

You can add the virtual field to other layouts and apply the same field template, or create other field templates that you apply to the virtual field.

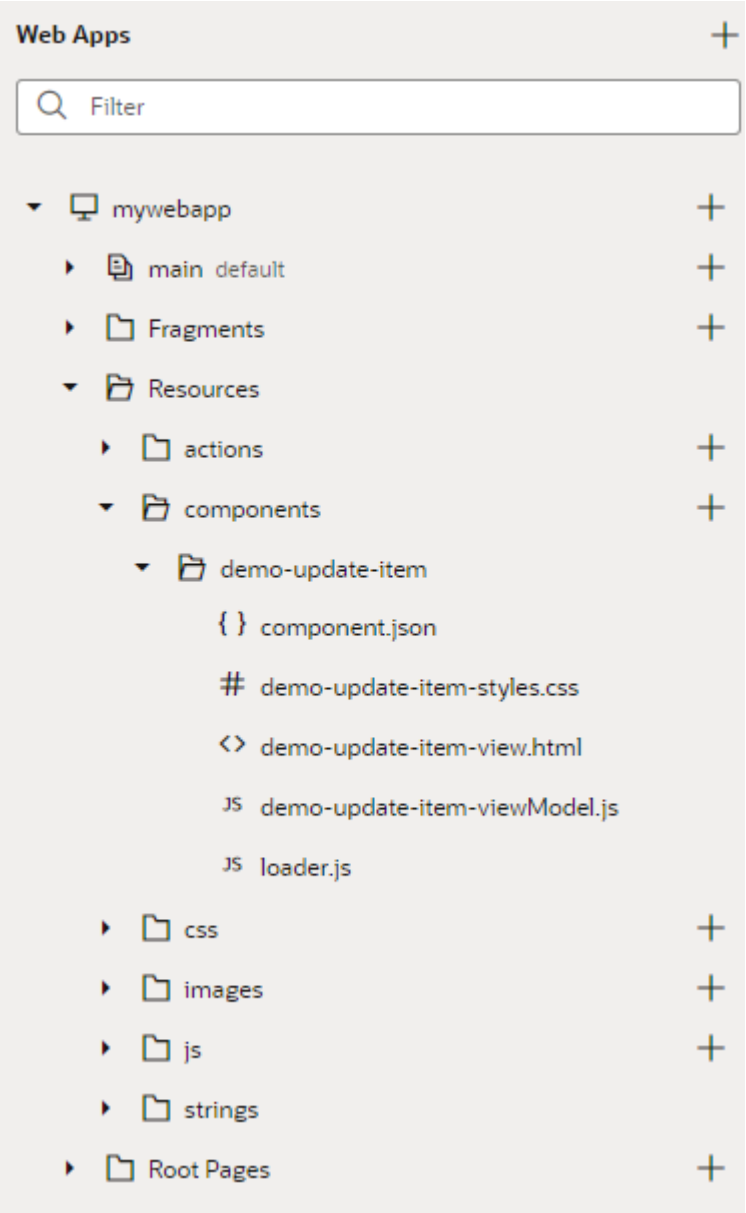
Work With Custom Web Components

Oracle JET web components are reusable pieces of UI that can be composed of multiple component types. Web components that you create can be used in your app or they can be uploaded to the Oracle Component Exchange to share with other developers.

There's a variety of supported web component types (as described in Design Custom Web Components). For an example of a web component, see Web Components in the JET Cookbook.

You can import web components in two ways: as a ZIP archive or directly from the Component Exchange associated with your instance. Likewise, web components can be published to a Component Exchange to share with other developers, or shared via a ZIP archive.

To view the web components that have been imported into your web (or mobile) app, expand the `Resources` node in the Navigator:



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. For a more detailed description of the JET web component architecture, see About Web Components in *Developing Applications with Oracle JET*.

File	Description
`loader.js`	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`.
`component.json`	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`.
`view.html`	The view for the web component
`viewModel.js`	Describes the ViewModel for the web component where methods defined in the web component metadata are defined
`styles.css`	Contains the custom styling for this web component

Note:

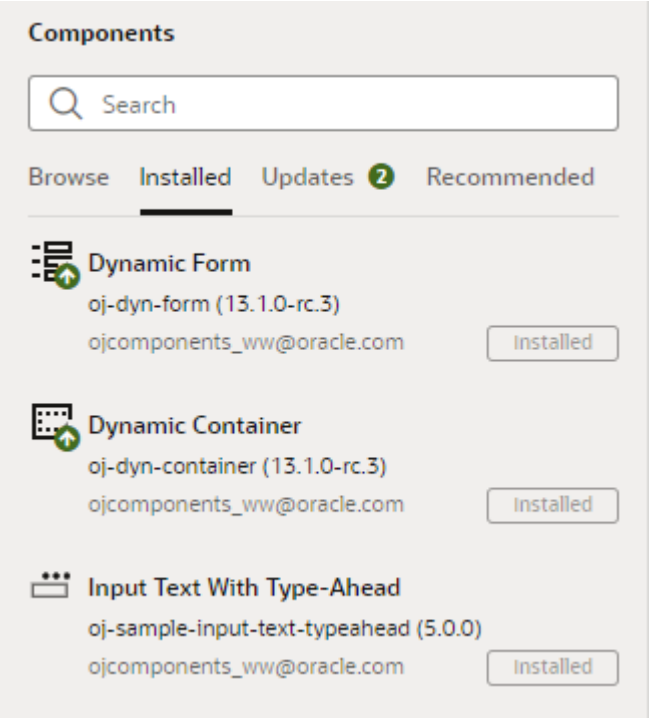
Web components are currently not backward-compatible. When importing web components, and when upgrading web components or your app, you need to ensure that your application and all web components that you use in your application are using the same version of JET.

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 use 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.

The Components tab in the Navigator helps you 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**: Use to search the Component Exchange for components that can be installed, to open a component's details page, and to install components.
- **Installed**: Use to view a list of installed components. The tab displays details about each component, and components are badged to indicate warnings or available updates.
- **Updates**: Use to view a list of available updates and to update components to the latest version.
- **Recommended**: Use to view suggested components based on what you've installed. For example, if you installed the Input text with type-ahead component, other components in the `oj-sample` JET pack might be recommended. By default, components in the Dynamic UI pack are always listed.

Get Components From the Component Exchange

If your administrator has associated your instance with the Component Exchange, you can use the Components tab in the Navigator to add and manage those components.

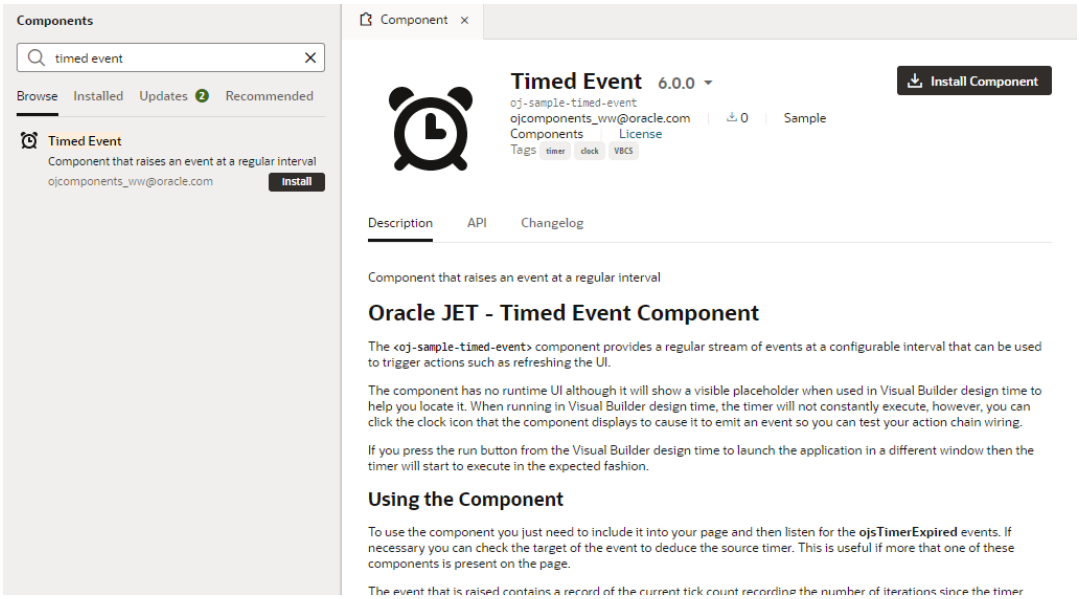
To add a component from the Component Exchange:

1. Open the Components tab in the Navigator.

   Alternatively, in the Components palette, select **Get Components** in the Components 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 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, it's added to your Components palette. You can now drag the new component onto the canvas and use it in your pages.

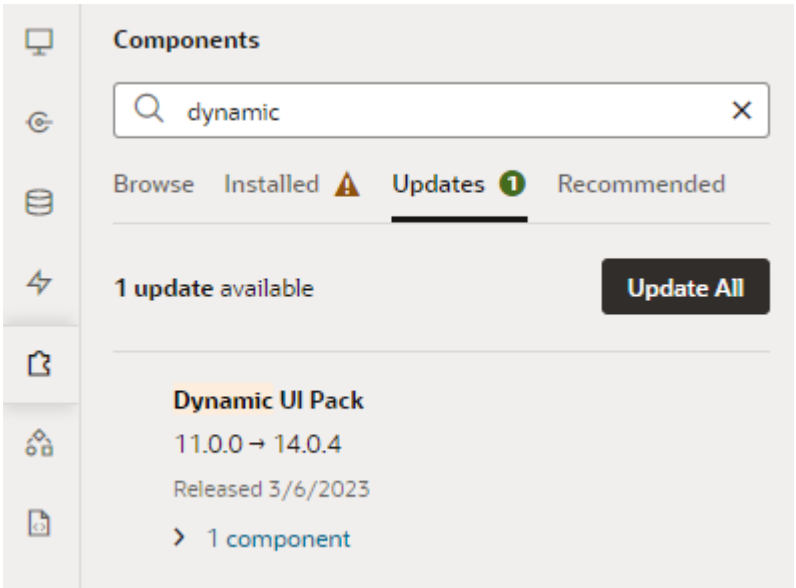
Update a Component from the Component Exchange

When a newer version of an installed component is available, you can install it in the Updates tab in the Components pane. You'll know an update is available when you see a notification in your browser window or a badge over the Components icon in the Navigator.

To update a component from the Component Exchange:

1. Open the Components tab in the Navigator.
2. Open the Updates tab in the Components tab.

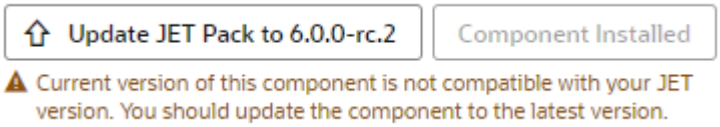
   If you installed a component that is part of a pack, the Updates tab displays the name of the pack containing the newer version of your component:



3. Click **Update All** to install all updates available for installed components.

   To update an individual component, click the component's name to open its detail page, then click the Update button.

   If the installed component is not compatible with the JET version in your Visual Builder instance, you'll see a notice to that effect.



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

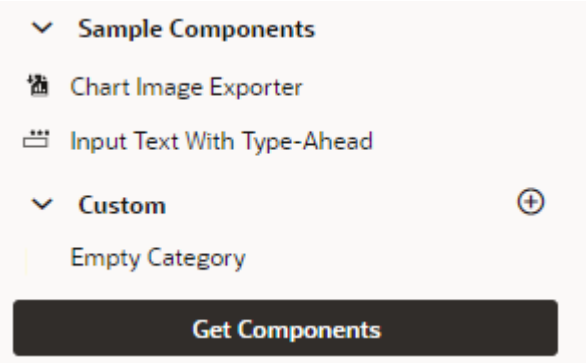
If you want to use a web component that's not available in your Component Exchange, you can import the component as a ZIP archive. For example, when your team member is developing a component, they can give it to you as an archive so you can try it in your application.

You also have the option of importing web components that allow you to:

- Embed Integration Insight dashboards in your visual application (Oracle Integration Generation 2 only). See Embed Integration Insight Dashboards in Visual Builder Applications.
- Use Process components in your visual application. See Use Process UI Composite Components (CCA).

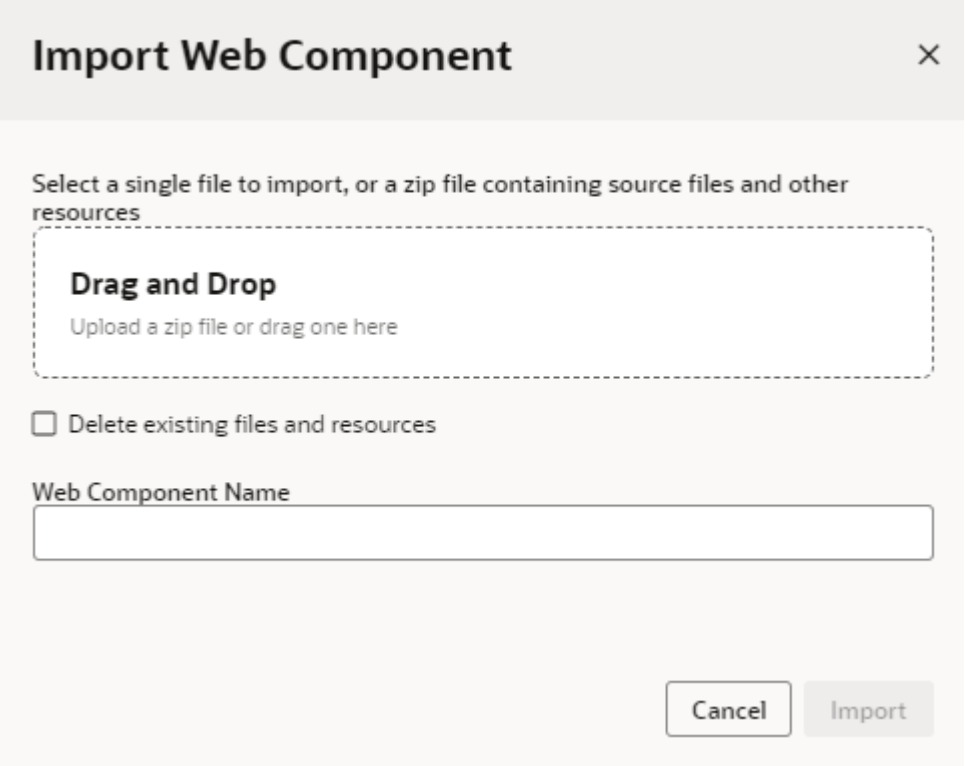
To import a custom web component archive:

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 Web Component** ( ).

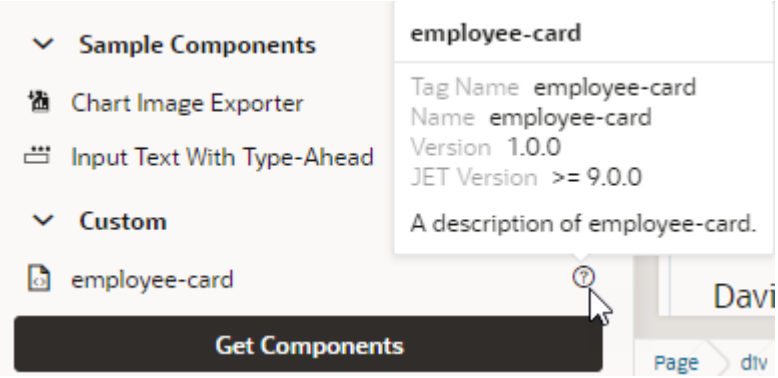


4. Upload your ZIP archive in the Import Web Component dialog box. Click **Import**.

   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 web component is added to your app's `resources/components` folder and displayed in the Custom Components category of the Components palette or in the category specified by the component's metadata. After importing the web component, you can position it in your page and configure its properties in the Properties pane as you would a standard component.



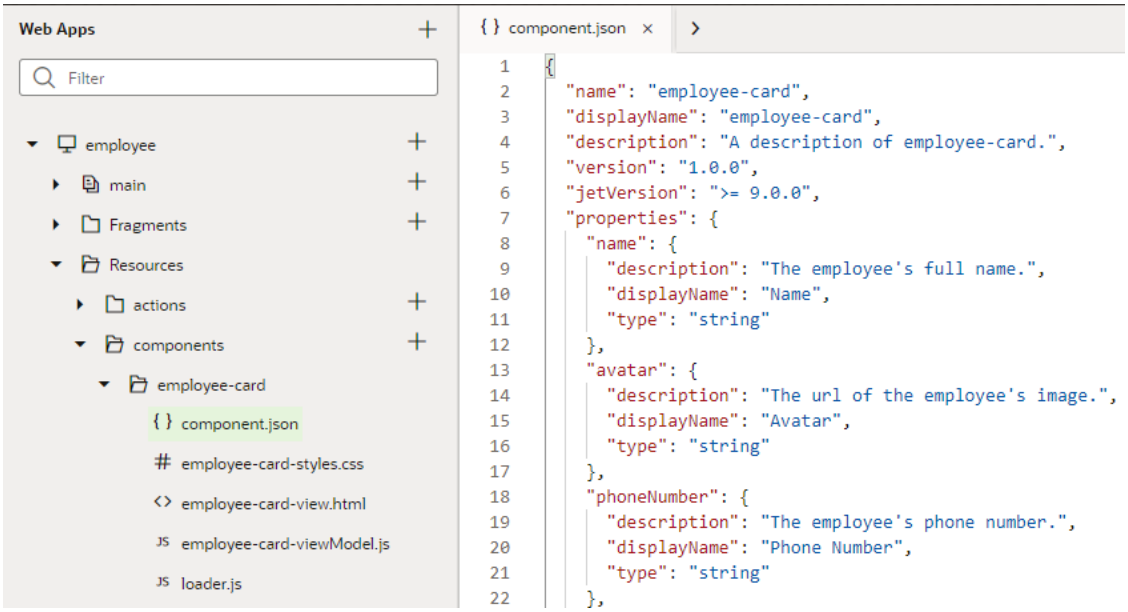
Note:

Importing web components makes them a part of your application. Because these components are not cached, you're likely to run into performance issues when they are downloaded each time you reload the Page Designer for preview, or at runtime when you publish an update to your app. As a best practice then, it helps to publish your components to a CDN (Content Delivery Network) or an external location that your browser can cache requests from. This is useful especially when you have multiple apps that use the same components. Talk to your administrator for site-specific information on how to publish these components externally.

Create a Web Component

Oracle JavaScript Extension Toolkit (JET) web components are reusable pieces of user interface code that you can embed as custom HTML elements. You can create a web component from a template that comes with Visual Builder.

When you use the Create Component dialog box to create a component, the new component contains the JavaScript, HTML, stylesheet,and JSON files required. The files contain sample code to help you get started. After you create the web component, you can click the component artifact in the Navigator to edit each of the component's files.



To create a web component:

1. Expand the **Resources** node of your web (or mobile) application in the Navigator.
2. Click **Create Component** (+) next to the **components** node.
3. Type the ID for the component in the Create Component dialog box. Click **Create**.

The new web component is added to your app's `resources/components` folder.

> **Note:**
>
> Manually creating web components makes them a part of your application. Because these components are not cached, you're likely to run into performance issues when they are downloaded each time you reload the Page Designer for preview, or at runtime when you publish an update to your app. As a best practice then, it helps to publish your components to a CDN (Content Delivery Network) or an external location that your browser can cache requests from. This is useful especially when you have multiple apps that use the same components. Talk to your administrator for site-specific information on how to publish these components externally.

9

Work with Variables and Types

When you use Quick Starts to build your application's pages, you won't have to worry about variables and types because Visual Builder automatically creates whatever is necessary for you. But if you're working with code to customize default logic or build your own, variables are key. You will use them to interact with UI components and data sources in order to implement your application's logic.

What are Variables and Types?

Variables are named pieces of information that hold business state and are bound (via expressions) to *components* on your application's pages. A variable, when bound to a component, can provide data values retrieved from a REST endpoint and display them to your users. It can also hold other state that is required by the component.

If users were to enter or change the component's value, the change is also written to the variable. How the variable behaves in this case is largely governed by *actions*, which may call underlying REST endpoints to apply the change. Components, variables, and actions together form the basic building blocks of an application.

So what do you define as a variable? Any piece of information really. It can be a simple variable, for example, a number-type variable to hold an employee's ID or a string-type variable for a name. It can also be a complex data structure, for example, an Employee structure with lastName, firstName, phoneNumber, address, and email elements. Here are the different types of variables available in Visual Builder:

- Primitive variables such as String, Number, Boolean, even a wildcard-type Any.
- Structured variables such as Object or Array, used to store data structures.
- Built-in variables used to get metadata, for example, to access the current page's ID and title or to access information about the current user.

Complex variables that define the type and structure of a variable's data are known as types. Every variable is assigned a type, either built-in or custom. A developer can create a type that, for instance, matches the REST payload and pass data using a variable assigned that type.

A variable's value can vary, as the name suggests. So if want to store values that don't change over time, such as your company name or "foot-to-meter" conversion rates, use *constants*. Constants are typed like other variables, but unlike variables, you can't change their values after they've been initialized—except when the constant's default value is an expression that refers to other variables; in this case, the constant's value changes when the other variable's value changes.

Variables, constants, and types are all defined within a *scope* and are automatically created and destroyed when the framework enters and exits a particular scope. They can be used in different places in your application based on the scope it is defined in:

- application: Variables defined at the application level are available anywhere in the application. They are useful for storing login names and other data that you want accessible both within and across an application's flows and pages.
- Flow: Variables defined at the flow level can be used within the flow and pages within that flow.

- Page: Variables defined at the page level can only be used within that page.
- Fragment: Variables defined at the fragment level can only be used within that fragment. Additionally a fragment, though referenced by an outer page, cannot reference variables defined in the page.
- Layout: Variables defined at the layout level can only be used within that layout container.
- Action chain: Variables defined at the action chain level can only be used within that action chain.

Variables as Input Parameters

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

Parameters can also be passed on the URL of the pages or flows that you're invoking. This approach makes it possible to bookmark pages that show specific data based on the parameter.

Default Value and Expressions

The initial value of a variable is determined by its default value. If a default value isn't provided, the value is "not set" or undefined and its initial value is determined based on its type. If provided, the default value can be a static value or an expression, which in turn can refer to other variables including constants, system properties, static values, and the like via implicit objects (such as `$variables` and `$page`, used to extract the value of a variable).

When defining the default value as an expression, the variable updates when any reference in the expression changes value. For example, a `fullName` variable might have the default value set as `{{ $variables.firstName + ' ' + $variables.lastName }}`. Any time `firstName` or `lastName` is updated, the `fullName` variable will be updated.

Here's a list of implicit objects you can use in expressions:

Name	Description	Where Available
`$application`	Retrieve the value of variables defined at the current application level. For example, if a variable called `empName` was defined at the application level, the `$application.variables.empName` expression is used to get its value.	Current application
`$flow`	Retrieve the value of variables defined at the current flow level. If `empName` was defined at the flow level, the `$flow.variables.empName` expression is used to get its value.	Current flow
`$page`	Retrieve the value of variables defined at the current page level. If you defined `empName` at the current page level, the `$page.variables.empName` expression is used to get its value.	Current page
`$variables`	A shortcut to retrieve the value of variables defined in the current scope. For example, if `empName` was defined at the current page level, `$variables.empName` can be used the same way you'd use `$page.variables.empName`.	Every scope that has a `variables` property

Name	Description	Where Available
`$fragment`	Retrieve the value of variables defined within a fragment. If you defined `empName` at the fragment level, the `$fragment.variables.empName` expression is used to get its value, particularly in action chains. You can also use `$variables.empName` to get the value local to the fragment. **Note:** Every fragment has a unique ID, which is accessible within a fragment using `$fragment.info.id`. You can use `$fragment.info.id` within expressions set on a component's ID property, or even the ID of a nested fragment.	Current fragment
`$layout`	Retrieve the value of variables defined in the current layout level. If you defined `empName` at the layout level, the `$layout.variables.empName` expression is used to get its value.	Current layout
`$chain`	Used to refer to variables in actions executing in an action chain.	The chain in which an action is executing
`$parameters`	Refer to a page's input parameters only in the `beforeEnter` event, because page variables do not exist until the `vbEnter` event.	In the `beforeEnter` event
`$listeners`	Refer to event listeners of an application, flow, or page in a component, for example, `$listeners.onSelectionChange`.	In a flow or page
`$event`	Retrieve the content of an event's payload in an event listener. For an event listener on a custom event, `$event` contains the payload for that event. For a event listener that listens for a variable's `onValueChanged` event, `$event` is a structure with the properties `name`, `oldValue`, `value`, and `diff`. See Start an Action Chain When a Variable Changes.	Event listeners and variable `onValueChange` listeners

Name	Description	Where Available

`$initParams`	Declarative initialization parameters which can be used in expressions. Initialization parameters—or `initParams`—are evaluated early and can be used in expressions that are evaluated before variables exist (for example, in service declarations or translation bundle paths). The `initParams` are defined within a `configuration` block in `app-flow.json`, for example:
	
	`"configuration": {`
	`  "initParams": {`
	`  "myServicePath": "some/path/",`
	`  "anotherPath": "http://somehost/foo"`
	`  }`
	`},`
	`"services": {`
	`  "myservice":`
	`"{{ $application.initParams.myServicePath +`
	`'myservice.json' }}"`
	`},`
	`"requirejs": {`
	`  "paths": {`
	`  "myPrefix":`
	`"{{ $initParams.anotherPath }}"`
	`  }`
	`}`
		Everywhere

These variable definitions can be used much the same as any other variable in Visual Builder, meaning, you can use them in component attributes, as parameters for JS functions, in actions (such as if), and so on.

Variable Events

When its value changes, a variable emits an `onValueChanged` event. (You can also add an `onValueChanged` event to constants if its default value is an expression containing a variable.) 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.

You can get the old and new variable values using 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 and provides the diff between the old and new values.

Note that the `onValueChanged` event is triggered only when the value is actually changed; setting a variable value to the same value does not trigger this event.

It's possible to trigger an action chain whenever a variable raises this event. For example, when a user clicks a row on a employee's table, you can set up an action chain to retrieve employee information whenever the employee's ID changes.

Data Binding

Variables are principally bound to components to display data, but these variables don't know where the data is derived from or what it is used for. To populate a variable from a REST call, you assemble an action chain using an action making that REST call and an action assigning the result to that variable. For common cases, Visual Builder provides quick starts to automate 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 pages. Typically, the variable's type matches the structure of the REST payload, though you have the option of defining your own type that matches your use case more closely, and then mapping the REST payload to a variable that uses that type.

Built-in Variables

Visual Builder provides several built-in variables that allow you to access application metadata.

currentPage
Use the `currentPage` variable on the `application` object to access some of the current page's metadata, such as ID and title. This variable automatically updates as the current page changes during navigation and can be used to update a navigation component with the currently selected page.

Name	Description
`$application.currentPage.id`	Path of the current page. The path describes the location of the page in the flow hierarchy.
`$application.currentPage.path`	Path of the current page for the application. The path describes the location of the page in the flow hierarchy.
`$application.currentPage.title`	Title of the current page.
`$flow.currentPage`	ID of the current page for this flow.

currentFlow
If there is a `routerFlow` in the page, use the `$page.currentFlow` variable to retrieve the ID of the current flow.

path
Use the `path` variable to build the path to a resource, such as an image located in a folder.

Name	Description
`$application.path`	Path needed to retrieve a resource located in the application folder.
`$flow.path`	Path needed to retrieve a resource in the flow folder.

user
Use the `user` variable to access information about the current user, based on the information returned by the security provider.

Name	Description
`$application.user.userId`	User ID (string).
`$application.user.fullName`	Full name of the user (string).
`$application.user.email`	User email (string).

Name	Description
`$application.user.username`	User name (string).
`$application.user.roles`	One or more user roles (array of strings)
`$application.user.roles.`*`roleName`*	Returns `true` if `roleName` is a role of this user.
`$application.user.permissions`	One or more 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 the user is authenticated.

info
Use the `info` variable to retrieve some information about the application and page descriptor.

Name	Description
`$application.info.id`	Application ID as defined in `app-flow.json`.
`$application.info.description`	Application description as defined in `app-flow.json`.
`$flow.info.id`	Flow ID as defined in *`flow-id`*`-flow.json`.
`$flow.info.description`	Flow description as defined in *`flow-id`*`-flow.json`.
`$page.info.title`	Page title as defined in *`page-id`*`-page.json`
`$page.info.description`	Page description as defined in *`page-id`*`-page.json`

See also Built-in Variables in the *Oracle Visual Builder Page Model Reference*.

Create Variables in Artifacts

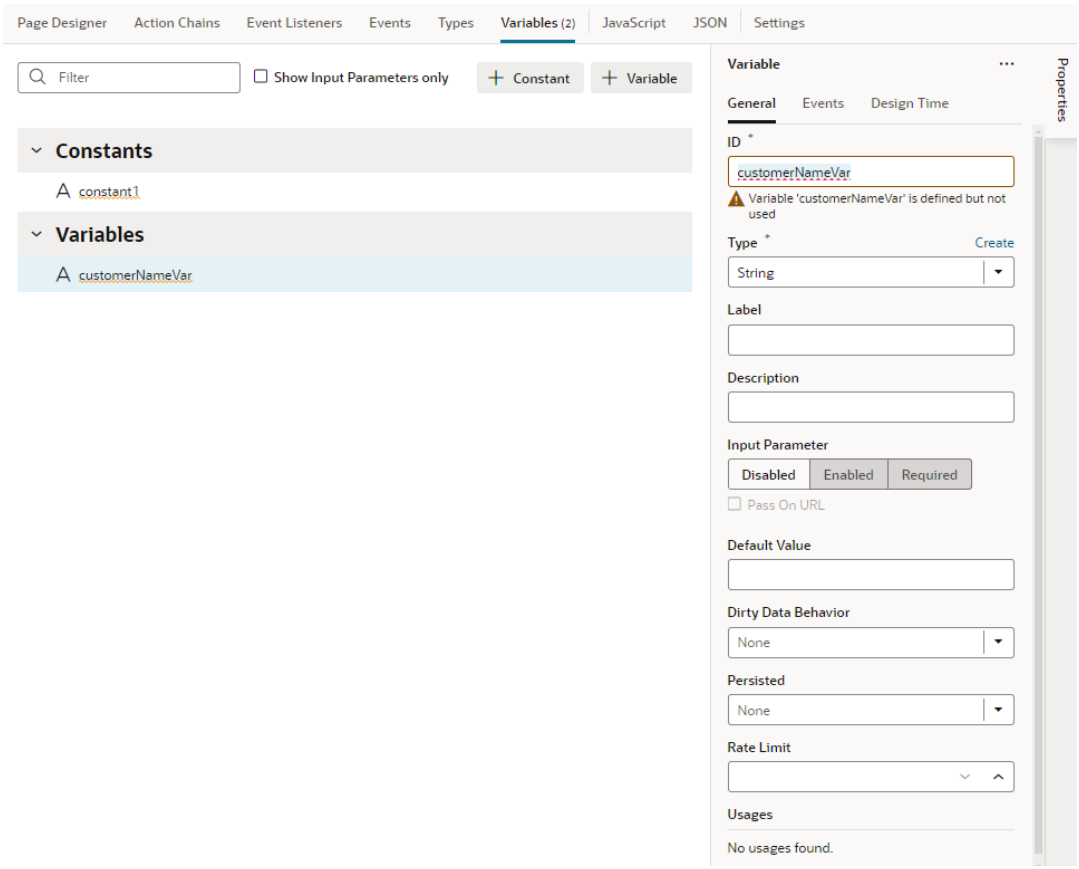
You can create variables and constants in application, flow, and page artifacts, as well as dynamic layouts and fragments. Variables and constants are assigned a scope based on where they are created, and the scope determines where they can be used. When you're deciding where to create a variable or constant, consider where it might be used.

Note though that unlike variables, constants remain the same throughout their scope. You can't change the value of a constant once it has been initialized.

To create a variable or constant in an artifact:

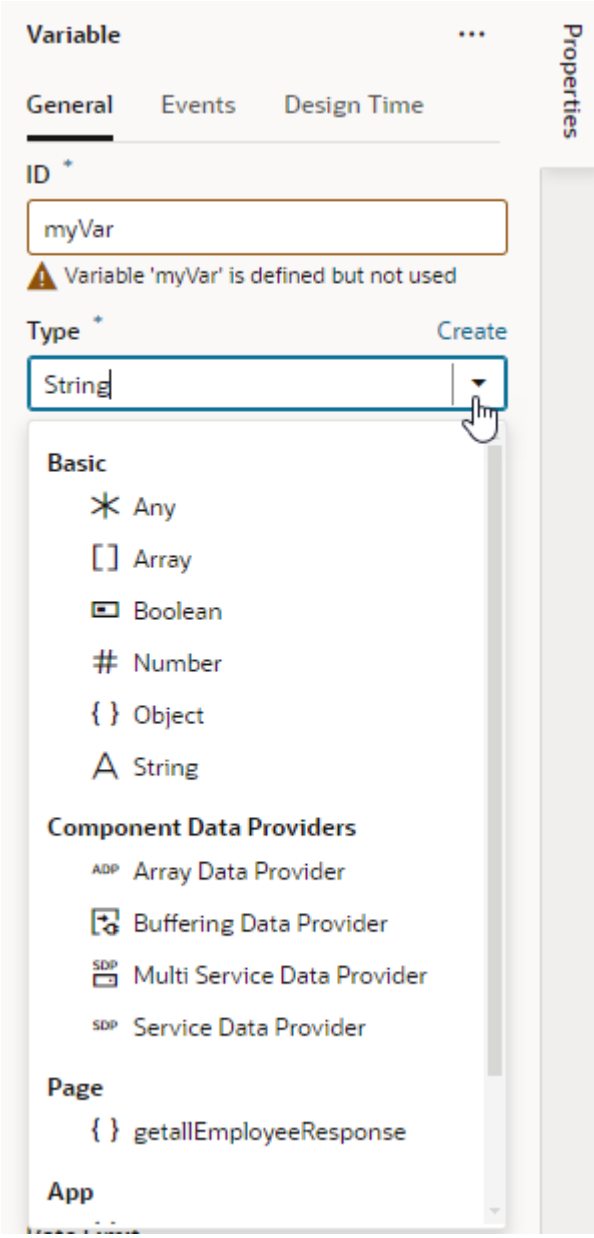
1. Click the **Variables** tab to open the Variables editor for your scope.

   If no variables or constants are defined, you'll see a message. Otherwise, you'll see a list of variables and constants defined for the artifact. Select a variable or constant and to view its attributes in the Properties pane:



2. Click **+ Variable** to create a variable, or **+ Constant** to create a constant.
3. Change the default name for the variable or constant in the **ID** field in the Properties pane.

   Default IDs are assigned as `variable1` or `constant1`. If the ID already exists, the number is incremented, for example, `variable2`, `variable3`, and so on.
4. Select the type in the **Type** drop-down list (default is **String**).

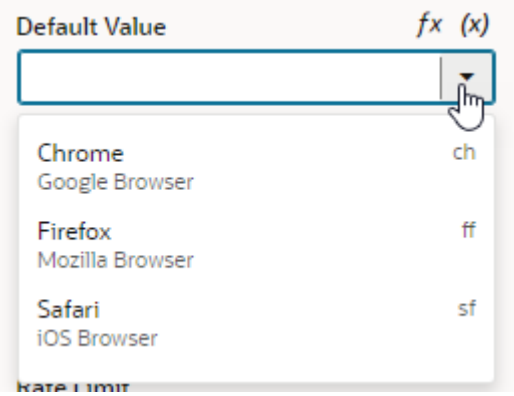


The type can be one of the built-in types (string, boolean, number, and so on) or a custom type (object, array, enumeration, or one based on an endpoint). To use a custom type with your variable, make sure the type is already defined (see Create Types).

> **Tip:**
>
> To quickly create variables for a custom type, switch to the **Types** editor, select your type, then right-click and select **Create Variable**.

After you create a variable, you can select the variable in the list and edit its properties, for example, to identify it as an `input`, or to add attributes if its type is `array` or `object`. You can also:

- Use the **Events** tab in the Properties pane to trigger an action chain when the variable's value changes, by adding a listener for the `onValueChanged` event and selecting the action chain that the change will initiate. See Start an Action Chain When a Variable Changes.
- Use the **Design Time** tab in the Properties pane to display a custom component for setting the variable's Default Value in the General tab. For example, if you want to use a color picker component instead of the default text field to select a color, select **Color** as the **Subtype** in the Design Time tab. If you want to use an enumerated list, select **Enum Values** as the **Subtype** and add your enumerated list, which then appears as a drop-down menu for the Default Value:



  The properties you see in the Design Time tab will depend upon the variable's type, and the Subtype property you select. For example, if Date is selected as the Subtype, you'll also see fields for setting Minimum and Maximum limits.

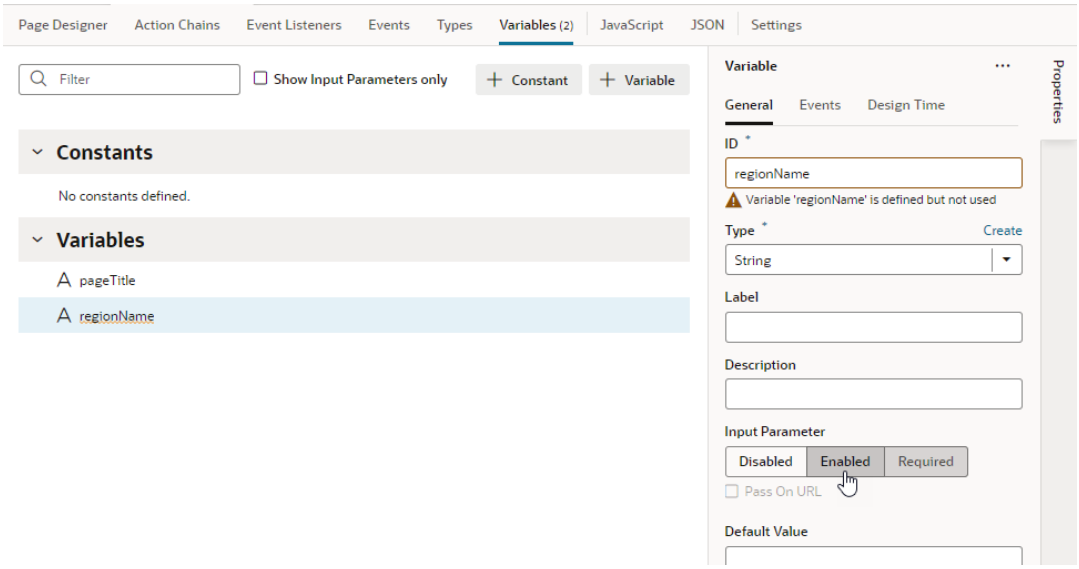
  Selecting Color, Time Zone, Date Time, Date, or Enum Values as the subtype will display custom components for the Default Value. If you select any of the other subtypes, you'll see a text field for entering the Default Value.

Once a variable is used in an application, you can view its usage information under **Usages** in the variable's Properties pane. You can see which pages access the variable and click links to readily navigate to those pages.

Enable Variables as Input Parameters

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

1. Create your variable or constant on the **Variables** tab for your scope.
2. In the variable or constant's properties, select an **Input Parameter** option (default is **Disabled**):
   - Click **Enabled** to pass the variable's value as an input parameter.
   - Click **Required** to require that the variable's value must be passed as an input parameter.

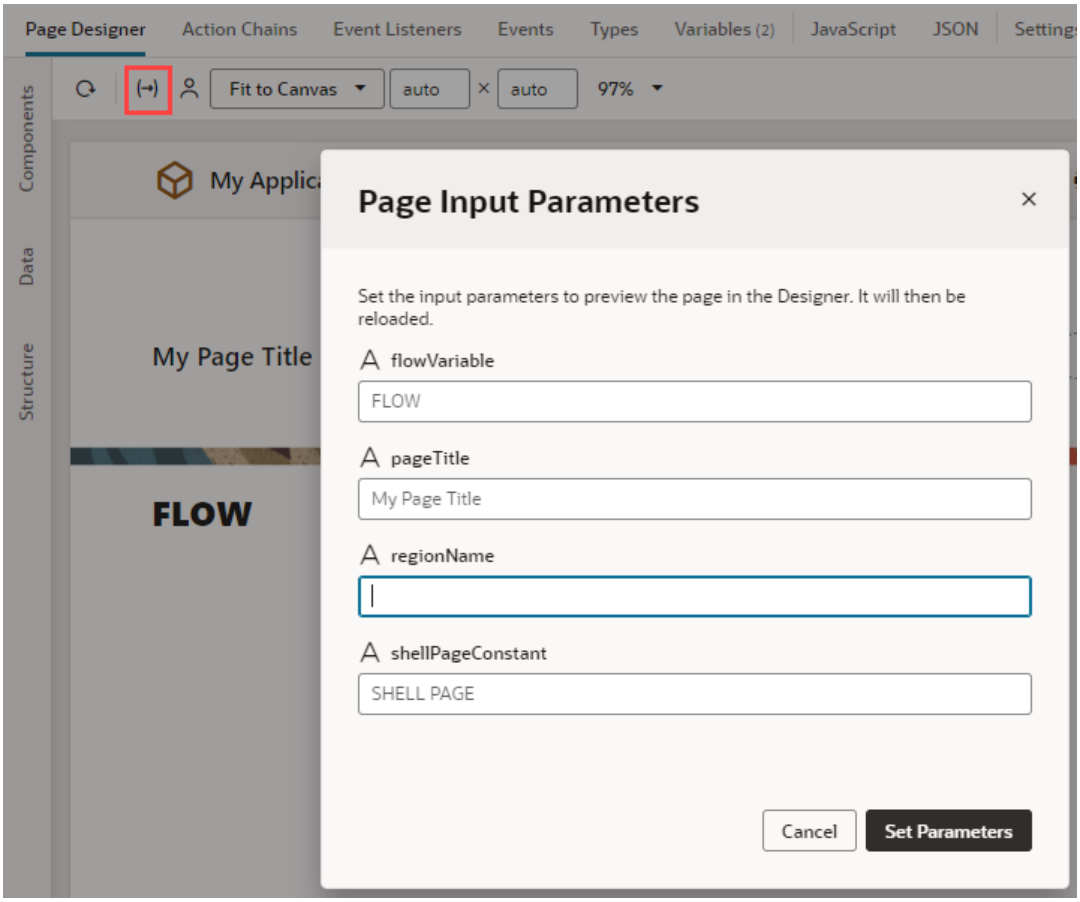


3. If you want to pass the parameter on the URL of the pages or flows that you are invoking (by adding `?paramName=Value` to the end of the URL), select **Pass on URL**. This option allows you to bookmark pages that will show specific data based on the parameter.
4. Optionally, set a default value.

> **Tip:**
>
> If you have a large number of variables defined, select **Show Input Parameters only** on the Variables editor to filter and view only those variables used as input parameters.

When a variable marked as an optional or required input parameter is used in your app, you can set its value on a page to see how the page displays. To do this:

1. Switch to the **Page Designer** tab.
2. Click (↪) in the toolbar to open the Page Input Parameters dialog. You'll see a list of input parameters for the page, including those from its parent flows and pages (if defined).
3. Set the parameter value and click **Set Parameters**.



If the variable was marked as a required input parameter but a value isn't assigned to it, you'll see a red dot on the Page Input Parameters icon, like this: . Click the icon then to set the missing parameter. Required parameters appear at the top in the Page Input Parameters dialog.

When a default value is set for the variable used as an input parameter, it will show when a value is yet to be assigned to the parameter. Deleting an assigned value will automatically apply the default value.

Track Variables to Detect Unsaved Changes

You can track changes in a variable's state as a way to detect unsaved changes in your application. Tracking a variable marks it as "dirty" any time its value changes, that is, when its current value differs from its initial value. You can then build an action chain to query for dirty variables and trigger a suitable response.

Suppose you want to notify users of unsaved changes before they leave a page, here's what you might do: set a page variable for tracking, then build an action chain as follows:

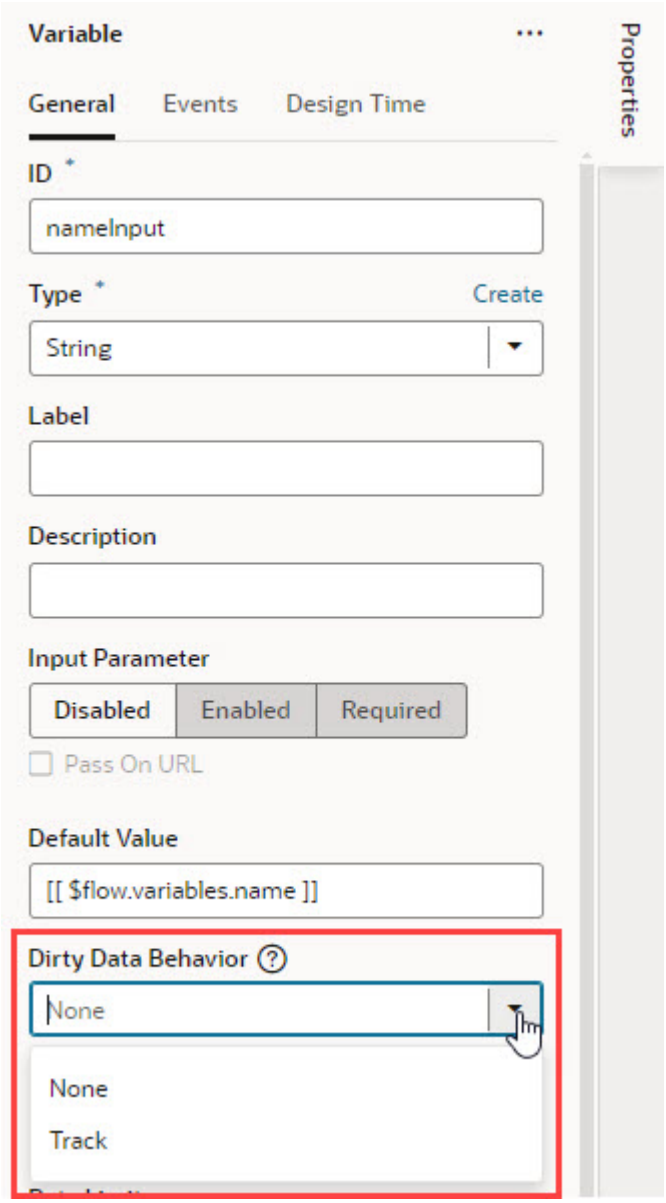
- Use the Get Dirty Data Status action to query for dirty variables in the action's current scope as well as in any containers within
- Take some action if dirty variables exist, for example, use the Fire Notification action to display a message. Note that a variable whose value was updated but changed back to its initial value is not considered dirty.

To enable a variable for tracking:

1. Open the **Variables** tab and access the variable whose state you want to track.

   You can track changes in a variable's state for all types of variables—except SDP variables and constants—in an app, flow, page, fragment, and layout.

2. In the variable's Properties pane, select **Track** from the **Dirty Data Behavior** list. The default is **None**, indicating that the variable's state is not tracked.



3. With the variable enabled for tracking, use the Get Dirty Data Status action to query changes in variable state and take appropriate action.

   Because the Get Dirty Data Status action queries for dirty variables in its current scope as well as in any containers within, make sure the scope of its action chain is correct for your tracked variable. For example, when you want to check for unsaved changes on a page, the tracked variable and corresponding action chain can be defined at the page level. If the page contains fragments and/or layouts, those will be checked as well.

   See Add a Get Dirty Data Status Action for more information.

4. When you don't want a tracked variable's changes to be flagged as dirty, you can reset its dirty state using the Reset Dirty Data Status action. Here are a few scenarios when you might want to do this:
   - Say a variable's initial value is "0" and a REST call changes it to "1". When you don't want this change to be tracked as dirty, calling the Reset Dirty Data Status action resets the variable's dirty data state such that "1" is considered the new initial value.
   - Say you have a page that allows users to save or cancel their changes. If users click a Cancel button to not save their changes, you might want to reset the tracked variable's state, so the change is no longer considered dirty.

   > **Note:**
   >
   > Be aware that when you add the Reset Dirty Data Status action to an action chain, it resets the dirty state on *all* tracked variables in the scope where the action chain is invoked as well as any containers within. For example, if your action chain is defined at the page level, all tracked variables at the page level will be reset. If the page contains fragments and/or layouts, tracked variables in those scopes will also be reset.
   >
   > Make sure the scope you define for the action chain that contains the Reset Dirty Data Status action is appropriate for your use case.

   See Add a Reset Dirty Data Status Action for more information.

Create Variables to Temporarily Store Data Changes in a Buffer

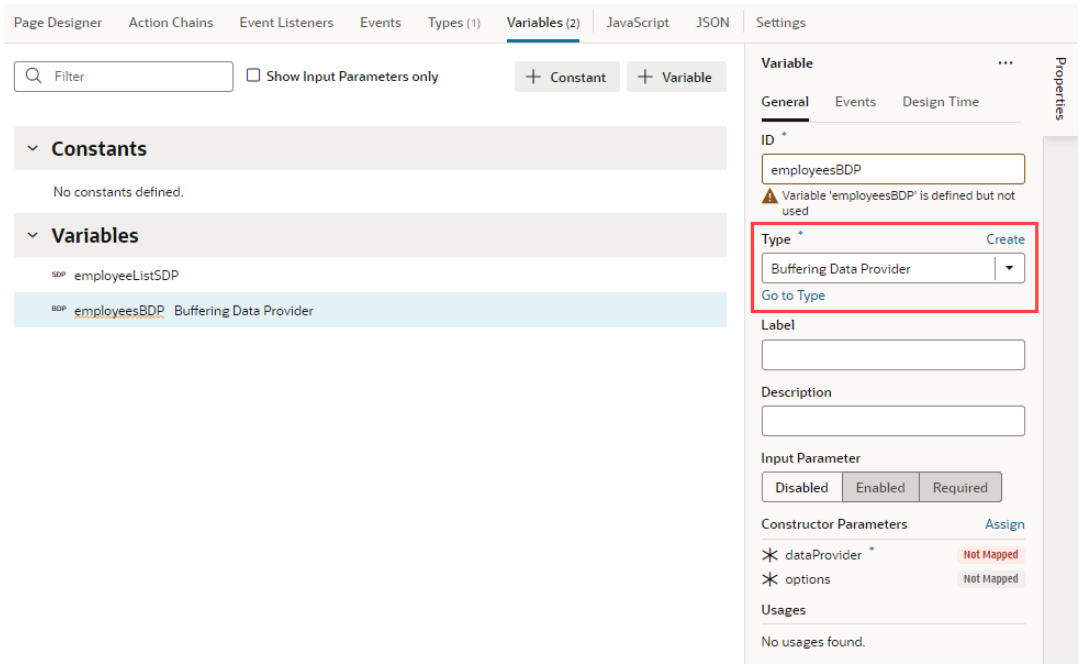
When you work with variables based on component data providers such as Service Data Provider and Array Data Provider, you can temporarily store data changes in a buffer until they are ready to be committed to the data source. To do this, you use variables based on a Buffering Data Provider.

The Buffering Data Provider is a wrapper that provides buffering for an underlying data provider, so edits can be committed to the data source later on. The underlying data provider is responsible for data fetches, while the Buffering Data Provider takes care of merging any buffered edits with the underlying data. This is useful functionality for batch processing, especially in the context of editable tables. For example, when users edit multiple existing rows or create new rows in a table, all changes can be stored in a buffer until the user clicks a Save button, at which time a REST call posts the buffered changes to the backend service. The buffered changes are held in a variable based on the built-in Buffering Data Provider type.

Buffering Data Providers require data providers that provide the actual data. For example, when you create a table based on a quick start, a Service Data Provider type variable is created to hold the table's data. The Buffering Data Provider simply wraps this underlying data provider to provide additional functionality, such as buffering for CRUD operations, commit, and revert. So before you create a Buffering Data Provider variable, make sure the underlying data provider is available.

Here's how to create a Buffering Data Provider variable and map it to the underlying data provider:

1. Open the Variables editor for your scope and create a variable of type Buffering Data Provider (for example, `employeesBDP`). Here's an example of a newly created Buffering Data Provider variable:



2. The Buffering Data Provider variable allows you to pass values to the constructor to initialize the newly created instance's properties. Here are the two Constructor Parameters it accepts:

   - `dataProvider` which must be set to an underlying data provider, and
   - `options`, as supported by the Oracle JET DataProvider API. See https://www.oracle.com/webfolder/technetwork/jet/jsdocs/CollectionDataProvider.html.

   a. To map the Buffering Data Provider variable to your component's underlying data provider, in the newly created variable's Properties pane, click the **dataProvider** Constructor Parameter.

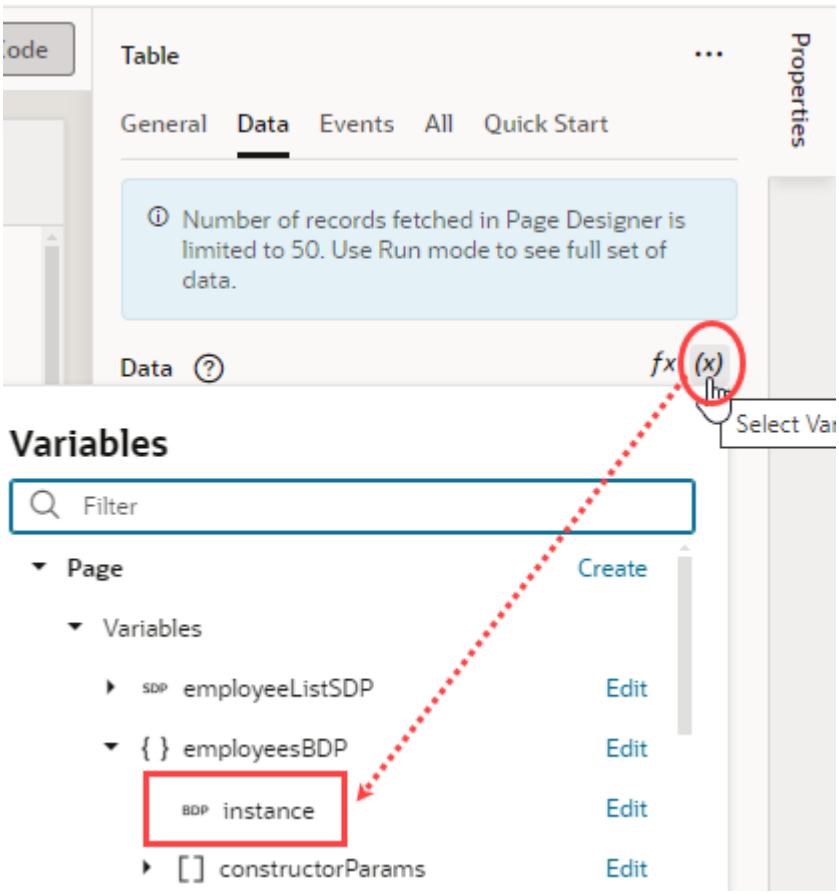
      In the Map Variables to Parameters dialog, map the original data provider on the Sources pane (for example, the `employeeListSDP` variable created when a table component is mapped to its data source) to the **dataProvider** parameter on the Target pane. Click **Save**.

   b. To pass any options from the underlying data provider, click **options** under Constructor Parameters, map the required variables, and click **Save**.

3. Bind the Buffering Data Provider variable to your component. Variables based on the Buffering Data Provider type can be bound to tables, list views, or any component that accepts a data provider.

   a. Switch to the Page Designer and select (for example) your table component.

   b. In the table's Data tab, click ***(x)*** to open the variables picker and select the instance of the Buffering Data Provider, instead of the original data provider. For example, you might replace `$variables.employeeListSDP` with `$variables.employeesBDP.instance`:



Create Types

Every variable in Visual Builder is assigned a **type**, something that defines the type and structure of the data stored in a variable. Two kinds of types can be assigned to variables: standard JavaScript built-in types and custom types that can be declared and instantiated as needed.

Standard *built-in* types can be used to specify data that are:

- 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 Array Data Provider, Service Data Provider, or Multi Service Data Provider.
  Service Data Provider (SDP) is typically used to store data retrieved from a REST endpoint and populate collection components such as tables and lists; Multi Service Data Provider is commonly used for list of values components when different fetch capabilities are required. Array Data Provider (ADP) is used when some operations need to be performed on the data.

  The Buffering Data Provider (BDP) type is a special built-in type. It wraps underlying data providers such as SDPs and ADPs to provide enhanced functionality such as buffering for CRUD operations, commit, revert, and so on.

When you use Quick Starts to develop your application's pages, Visual Builder creates whatever types are necessary. If you do want to create your own type, you have the option of creating *custom* types. Custom types can be based on an *endpoint* to make sure the shape of the variable's data matches what the endpoint expects in its payload. You can also create types to define custom objects and arrays. Advanced users can further define a type *from code* (such as a type class written in JavaScript or a typescript class) and associate this type to a special **InstanceFactory** variable. This way, they can simply plug their type into a variable without writing any extra JavaScript code.

Types from an endpoint or from code define custom type structures either by using the endpoint (response) data structure, or by using a type class (or a type declaration file when provided).

When creating types, remember that they are defined within a scope, just like variables. They can be created at the application, flow, and page level. They can also be defined at the dynamic layout level to be shared between dynamic layout templates, and at the fragment level.

Create a Custom Object or Array

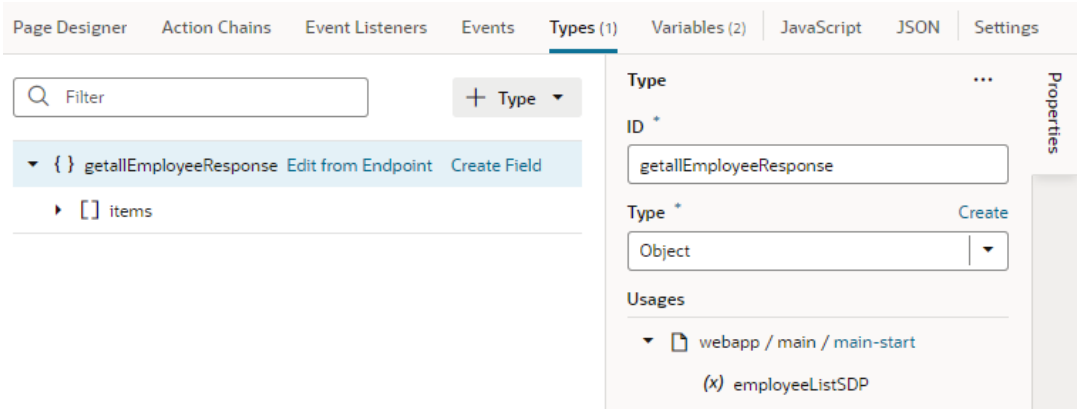
Custom types can be used to define the structure of any variable. Create a custom type when you want a type that defines an array or an object, and you want to individually add the attributes that define the type's data structure.

You create a custom *object* when you want a type to define an object that contains properties, and a custom *array* when you want a type to store multiple variables of the same type.

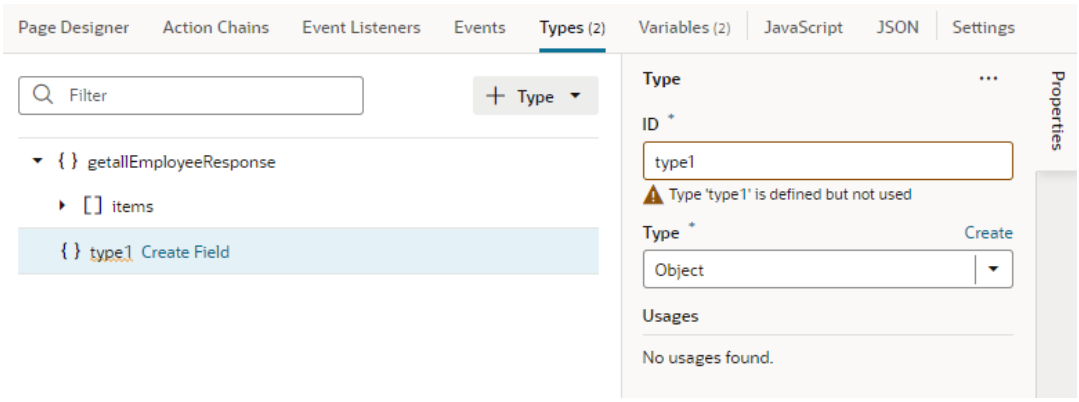
To create a custom object or array:

1. Select your application, flow, or page artifact in the Navigator. You can also select a dynamic layout or a fragment.
2. Click the **Types** tab to open the Types editor.

   The Types editor displays all the types defined for the artifact.



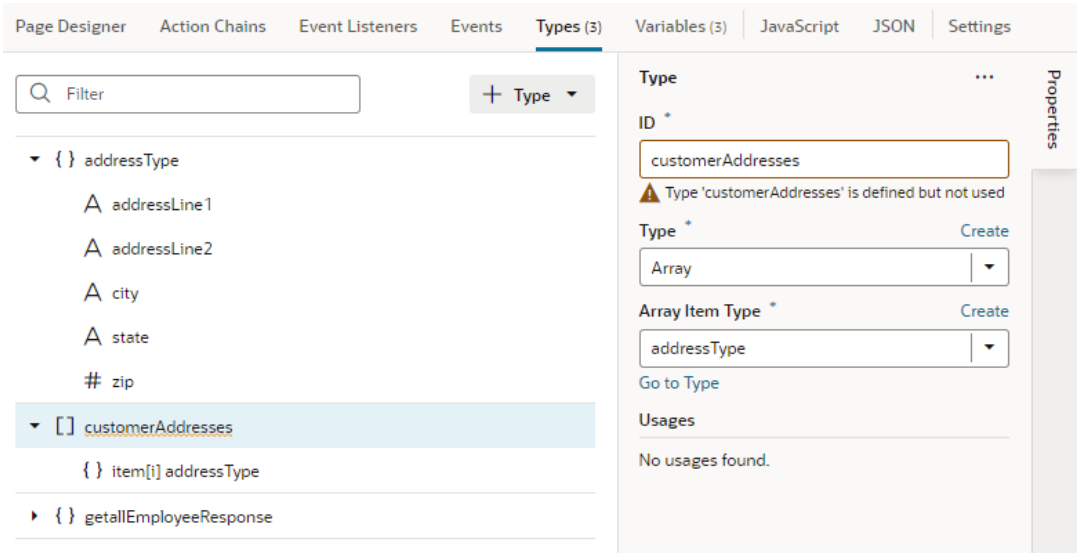
3. Click **+ Type** and select **Custom**.
4. In the type's Properties pane, give the type a unique name in the **ID** field, then select **Object** or **Array** from the **Type** list.



- To create a custom object (for example, an `addressType` that defines the fields of an address), select **Object**. An object type can also contain nested arrays.
- To create a custom array (for example, a `customerAddresses` type that defines an array of addresses), select **Array**.
  Arrays are defined the same way as objects, but the object type is inside an array. Arrays can have nested objects or arrays as well.

5. Define the type's structure by adding attributes:
   a. Hover next to the new type (or item type), then click **Create Field** to add an attribute.
   b. In the field's Properties pane, update the ID as needed and select a type for the new attribute.

   You can add as many attributes as you need to refine the type's data structure:



After you've defined your type, create a variable that uses this type. Right-click the type and select **Create Variable**, or go to the **Variables** tab to create one.

When a type is associated with a variable, you can view its usage information under **Usages** in the Properties pane (for example, to see which variables are based on it and which pages use those variables). Simply click a usage to readily navigate there.

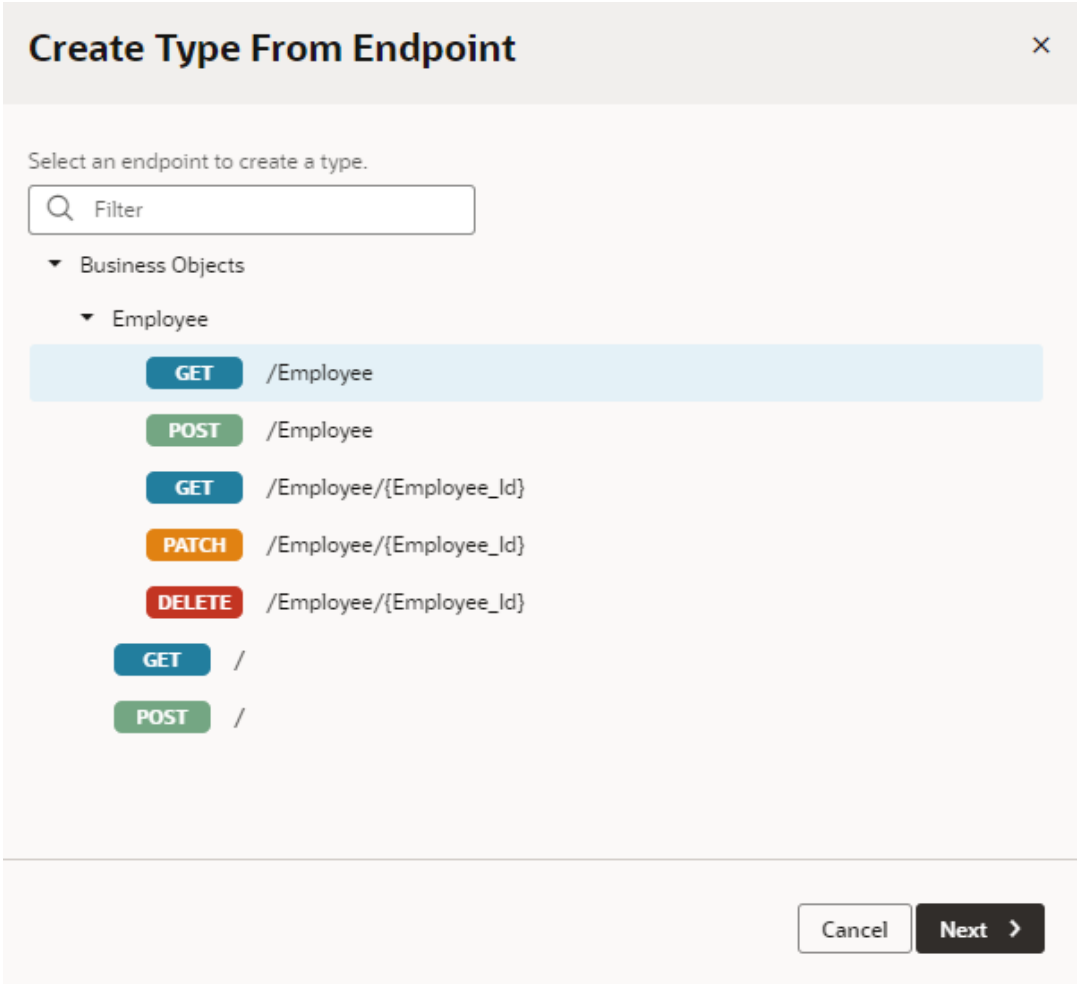
Create a Type From an Endpoint

When you create a type from an endpoint, you define a custom type structure by using the endpoint's (response) data structure.

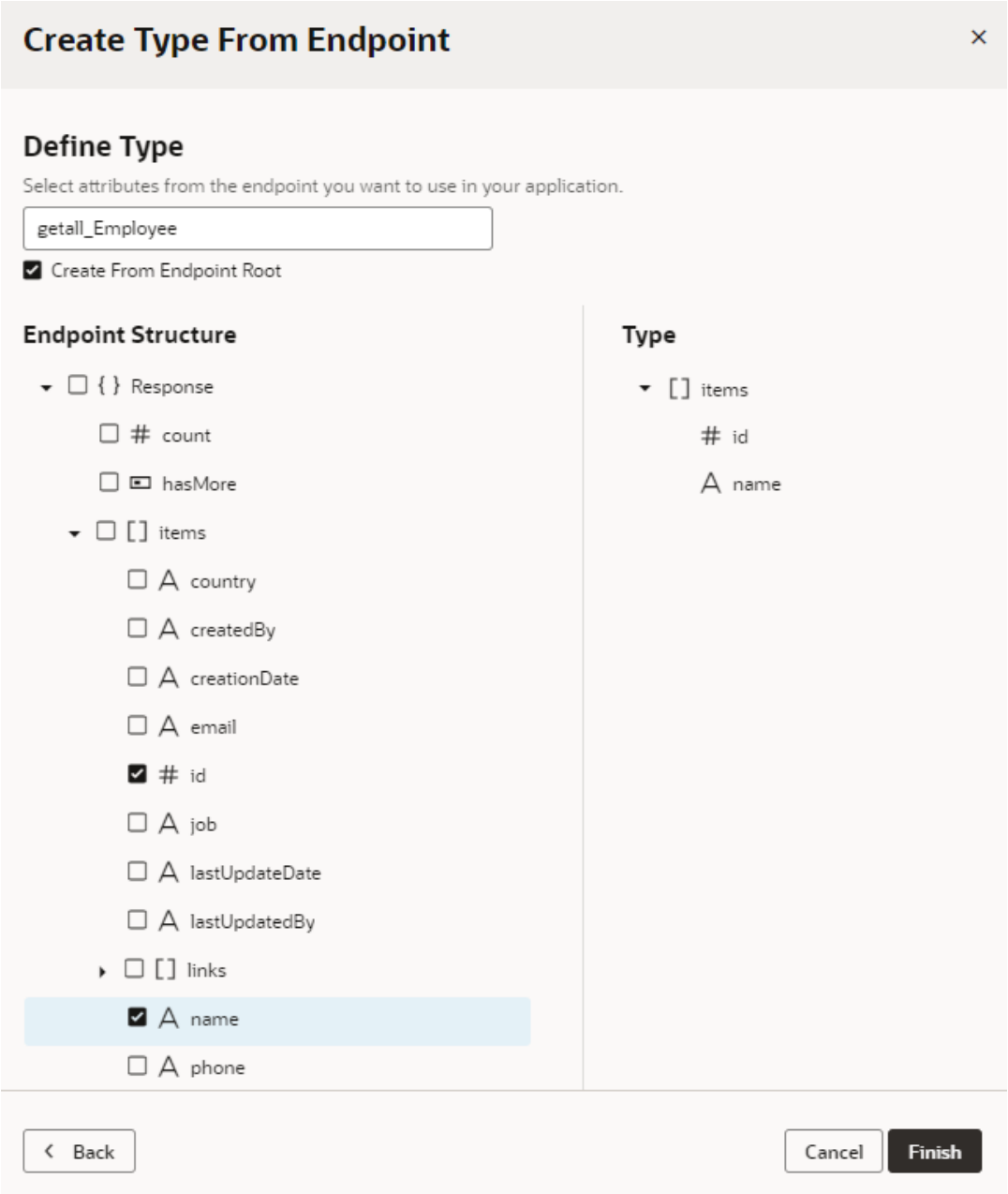
For example, when sending a request to a `getall_Employee` endpoint, you might want the structure of the response to be an array with the id and a few specific fields (say, a string name and email, among others). You can create a type by selecting the endpoint, then selecting the fields that you want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

To create a type from an endpoint

1. Select your application, flow, or page artifact in the Navigator. You can also select a dynamic layout or a fragment.
2. Click the **Types** tab to open the Types editor.
3. Click **+ Type** and select **From Endpoint**.
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 type in the Types editor, you can see it is an object type with an array `items` containing the fields in the endpoint that you selected.
Once your type is used in an artifact, you can view its usage information under **Usages** in the Properties pane (for example, which variables are based on it). Simply click a usage to readily navigate there.

Create a Type From Code

When you want to use your own type (for example, a type class written in JavaScript or a typescript class) with a variable in Visual Builder, you can create a type from code to create an instance of that type class. Types from code, called InstanceFactory types, can be created by

importing your type definition to declaratively plug in any Oracle JET type class or a custom type class, then using it with a category of variable known as an InstanceFactory variable.

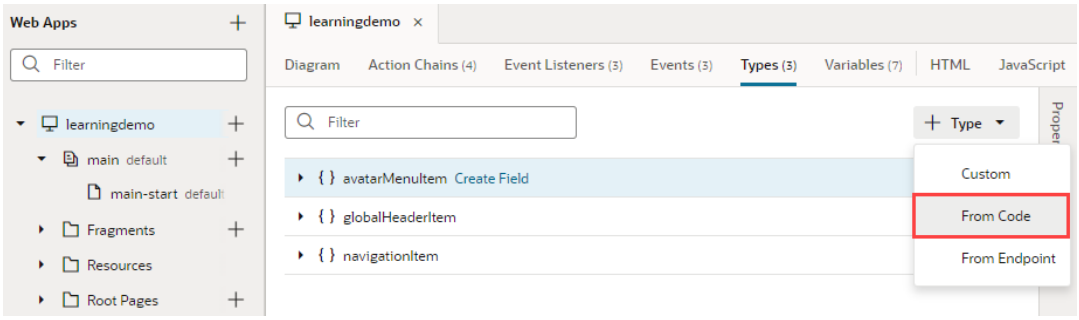
InstanceFactory types and variables let you use your own type class (say, the `myapp/MyTypeFromCode` JavaScript class) as a type with Visual Builder variables without having to re-implement a new type (Javascript class) that uses or extends the VB Extended Type framework. You'll only need to provide a typescript definition file (`*.d.ts`) or a typescript file (`*.ts`) that defines your type's details. Visual Builder will parse your type definition, generate a JSON representation that is compatible with existing type schema, and create an InstanceFactory type, which you can then assign to an InstanceFactory variable (`vb/InstanceFactory`). The InstanceFactory variable uses the InstanceFactory type and additionally the list of constructor arguments declared by the type to define its constructor parameters (`constructorParams`).

The InstanceFactory variable that uses an InstanceFactory type creates immutable instances of the type class. When a page loads, the InstanceFactory variable creates the first instance of the type (using the configured constructor parameters). It also creates a new instance of the type class whenever its constructor parameters change. You can use the Assign Variables action or the Reset Variables action to update constructor parameters.

You can also use the Call Variable action to call methods on the variable instance (see Add a Call Variable Action).

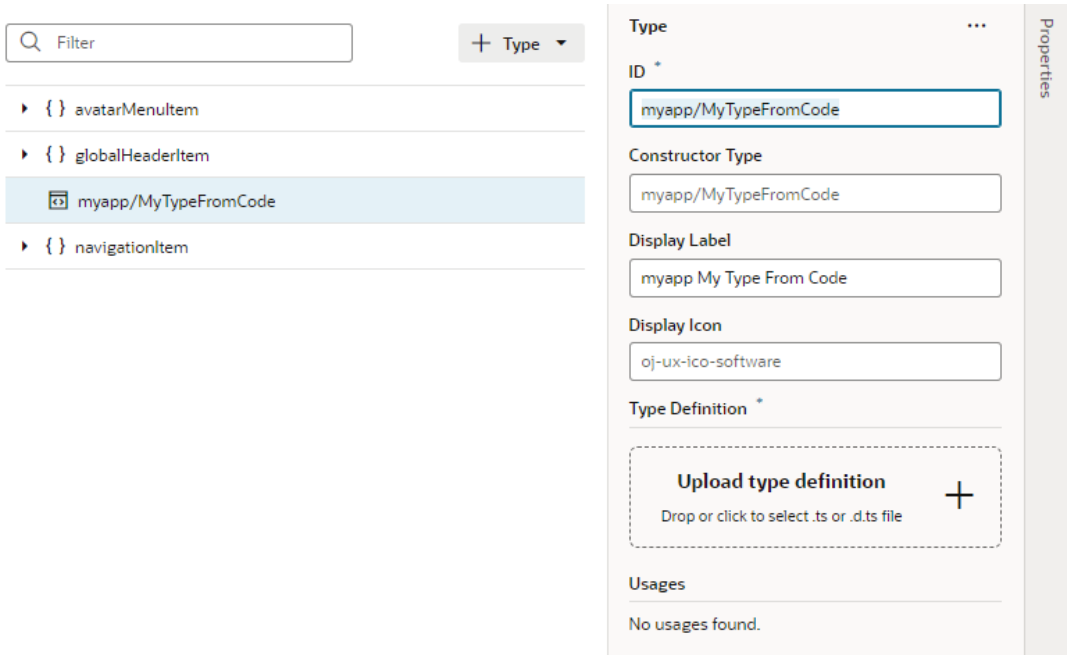
To create a type from code:

1. Select your application (or fragment) in the Navigator.
2. Click the **Types** tab to open the Types editor.
3. Click **+ Type** and select **From Code**. This option is available only at the app (or fragment) level, so make sure you've selected the correct node in the Navigator.

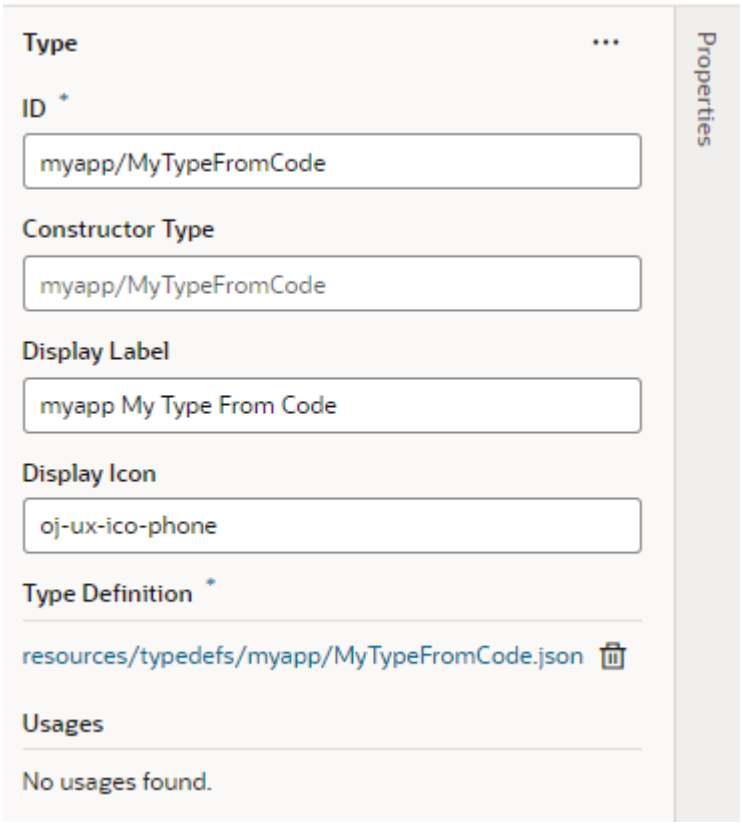


4. Enter the type name using namespaces (for example, `myapp/MyTypeFromCode`) and click **Create**. Namespaces separated by a slash (/) help organize the types.

   An InstanceFactory type is created, with its default icon set to oj-ux-ico-software. A default display label is also generated based on your values, as shown here in the Properties pane:

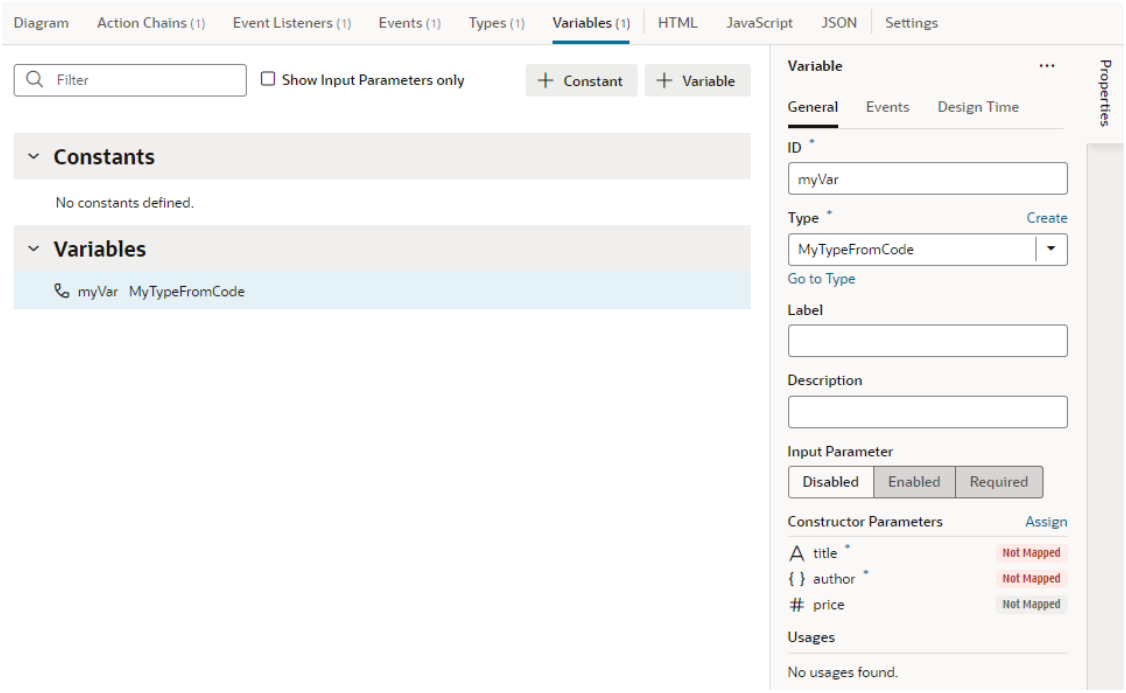


5. If you want, configure the type's properties to use a custom display label and a display icon of your choice. In our example, let's set the display label to `MyTypeFromCode` and the display icon to `oj-ux-ico-phone`.
6. Drag and drop (or click **Upload type definition** to provide) a typescript (`.ts`) or typescript definition (`.d.ts`) file that specifies the type's details:



Visual Builder converts your type definition to a JSON representation, then saves the JSON file under `resources/typedefs/`.

After you've created a type from code, you can create a variable for this InstanceFactory type, just as you would for any other type. For example, here's a `myVar` variable assigned to the `MyTypeFromCode` type:



However, because an InstanceFactory type includes a constructor, initializing InstanceFactory type variables is not the same as other variables. It requires parameters to be mapped to the constructor. To do this, click **Assign** in the variable's Properties pane, then map the constructor parameters.

The shape for the constructor parameters comes from your type definition file. Here's an example type and variable declaration as shown on the JSON tab:

```
"types": {
  "myapp/MyTypeFromCode": {
    "label": "MyTypeFromCode",
    "constructorType": "vb/InstanceFactory<myapp/MyTypeFromCode>",
    "iconClass": "oj-ux-ico-phone",
    "typedef": "resources/typedefs/myapp/MyTypeFromCode.json"
  }
},
"variables": {
  "myVar": {
    "type": "myapp/MyTypeFromCode",
    "constructorParams": [
      "Book Giver",              <<<<< this is title
      {                          <<<<< this is author
        "firstName": "Lois",
        "lastName": "Lowry"
      },
      49.99                    <<<<<<< this is price
```

```
    ]
  }
```

> **Note:**
>
> To make the JavaScript implementation for types like `myapp/MyTypeFromCode` available at runtime, make sure the path to your implementation is correctly mapped in requireJS, for example:
>
> ```
> "requirejs": {
>     "paths": {
>       "myapp": "resources/js/myapp"
>     }
>   },
> ```

One InstanceFactory variable can reference another InstanceFactory variable. In this example, the `incidentsView` variable references `incidentsSDP`, another InstanceFactory variable:

```
"incidentsSDP": {
  "type": "vb/ServiceDataProvider2",
  "constructorParams": [
    {
      "endpoint": "demo-data-service/getIncidents",
      "keyAttributes": "id",
      "itemsPath": "result",
      "uriParameters": "{{ $variables.technicianURIParams }}"
    }
  ]
},
"incidentsView": {
  "type": "ojs/ojlistdataproviderview",
  "constructorParams": [
    "{{ $page.variables.incidentsSDP.instance }}",
    {
      "sortCriteria": [
        {
          "attribute": "priority",
          "direction": "ascending"
        }
      ]
    }
  ]
}
```

Any time the `incidentsSDP` variable changes (that is, a new instance is created), the `incidentsView` variable re-creates a new instance of `ojs/ojlistdataproviderview`. This also means that components bound to either variable are notified of the change.

Service Data Provider

Service Data Provider represents a data provider that provides data by fetching it from a service or endpoint and that can be bound to components. It also allows externalizing fetches through an action chain.

The Service Data Provider can be used to fetch collections of data either implicitly using a configured endpoint, or externally by delegating to an action chain. Additionally, when Service Data Provider uses an Oracle Cloud Applications service, the built-in business object REST API transforms associated with the service automatically enable capabilities such as sorting, filtering, and pagination of the data. When used with endpoints not part of an Oracle Cloud Applications service, it's important for service authors to provide a custom transforms implementation that supports these capabilities. (It's worth noting that some functionality is controlled by the type of endpoint. For example, pagination properties such as `limit` and `offset` are available on a Get Many endpoint, but not a Get One endpoint.)

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.

When the properties of the Service Data Provider variable change, it listens to the variable onValueChanged event, and notifies all its subscribers (such as components) to refresh (by raising a data provider event). Currently, UI components are the only listeners of this event.

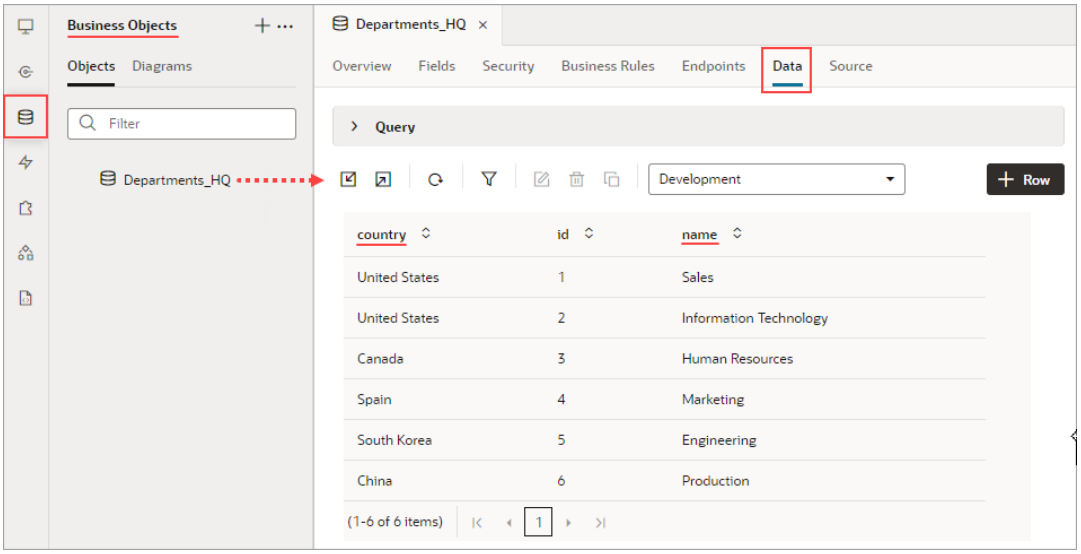
Creating a Custom Fetch Action Chain - An Example

Let's go through an example of creating a custom fetch action chain for an SDP. In this example, we'll add a Single Select component to a page and bind it to an SDP in order to list the department headquarters from a business object. We'll then configure the Single Select component to show additional fields, and create a custom fetch action chain for the SDP to retrieve the data for the additional fields. Each row in the list will show the department headquarters' name, the country that the headquarters is in, and an image of the country's flag.

Before we begin, we'll need to create a Departments HQ business object with a Name and a Country field. We'll also need to add a Single Select component to a page and to use a Quick Start to map it to the Departments HQ business object.

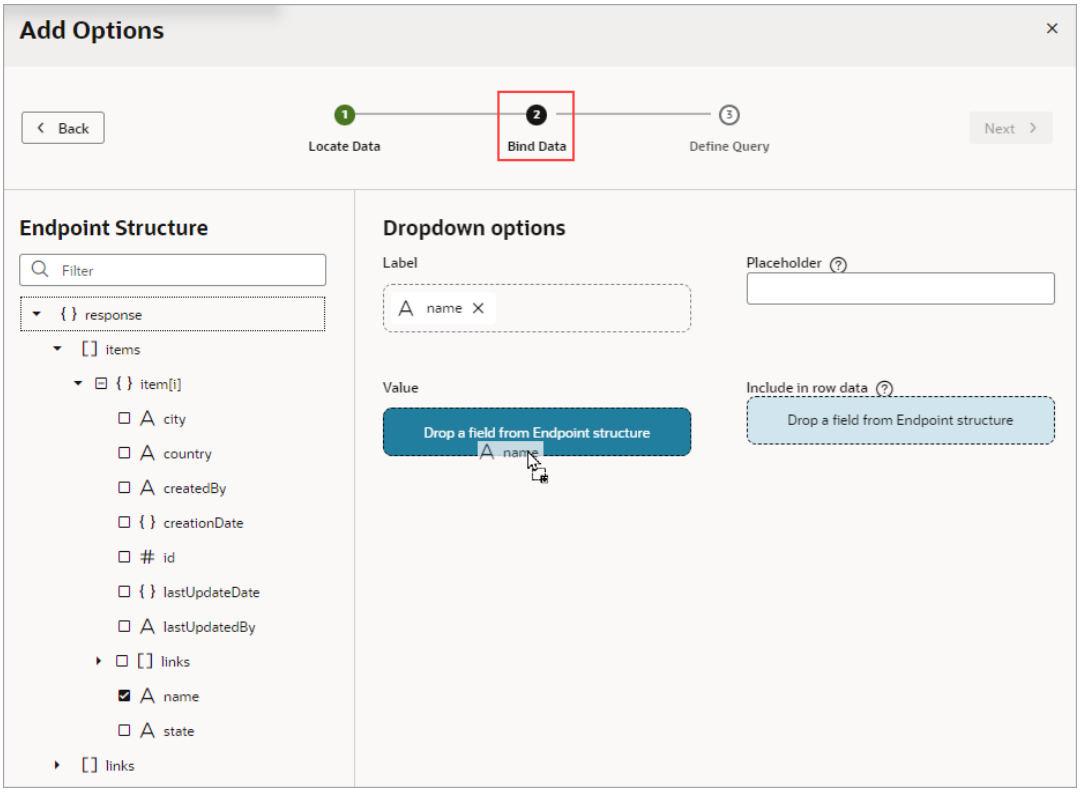
Let's begin by creating the Departments HQ business object:

1. Select the Business Object tab, then click **+** at the top right to create a new business object called Departments_HQ.
2. On the **Departments HQ** tab, select the **Fields** tab and add a `name` (String) and a `country` (String) field.
3. On the **Data** tab, add a few rows for the Select Single component to list:

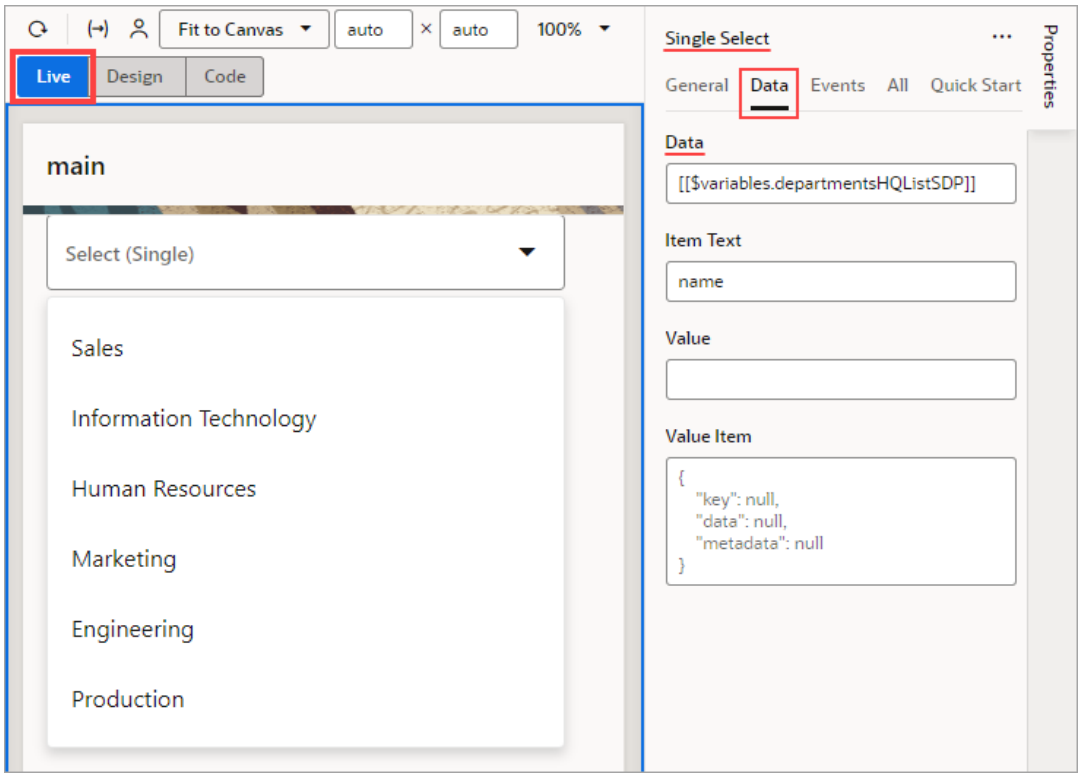


Next, we'll add a Select Single component to a page, and then use its Add Options Quick Start to map it to the Departments HQ business object.

4. Create a new page, then in the Page Designer tab, add a **Select (Single)** component to the page.
5. To map the component to the Department HQ business object, on the Properties pane's **Quick Start** tab, select **Add Options**.
6. For the Locate Data step of the Add Options wizard, under Business Objects, select **Departments_HQ**. Click **Next**.
7. For the wizard's Bind Data step, drag-and-drop the `name` field (department name) into both the **Label** and **Value** box. Click **Next**.



8. For the wizard's Define Query step, click **Finish**.
An SDP is automatically created for Department HQ, which fetches the `name` fields from the business object. You can see the new SDP on the page's Variables tab. You can also see that the Select Single component has automatically been bound to the SDP on the Properties pane's Data tab. If you switch the Page Designer to **Live** mode, you'll see the department names that were fetched by the SDP listed in the Select Single component:

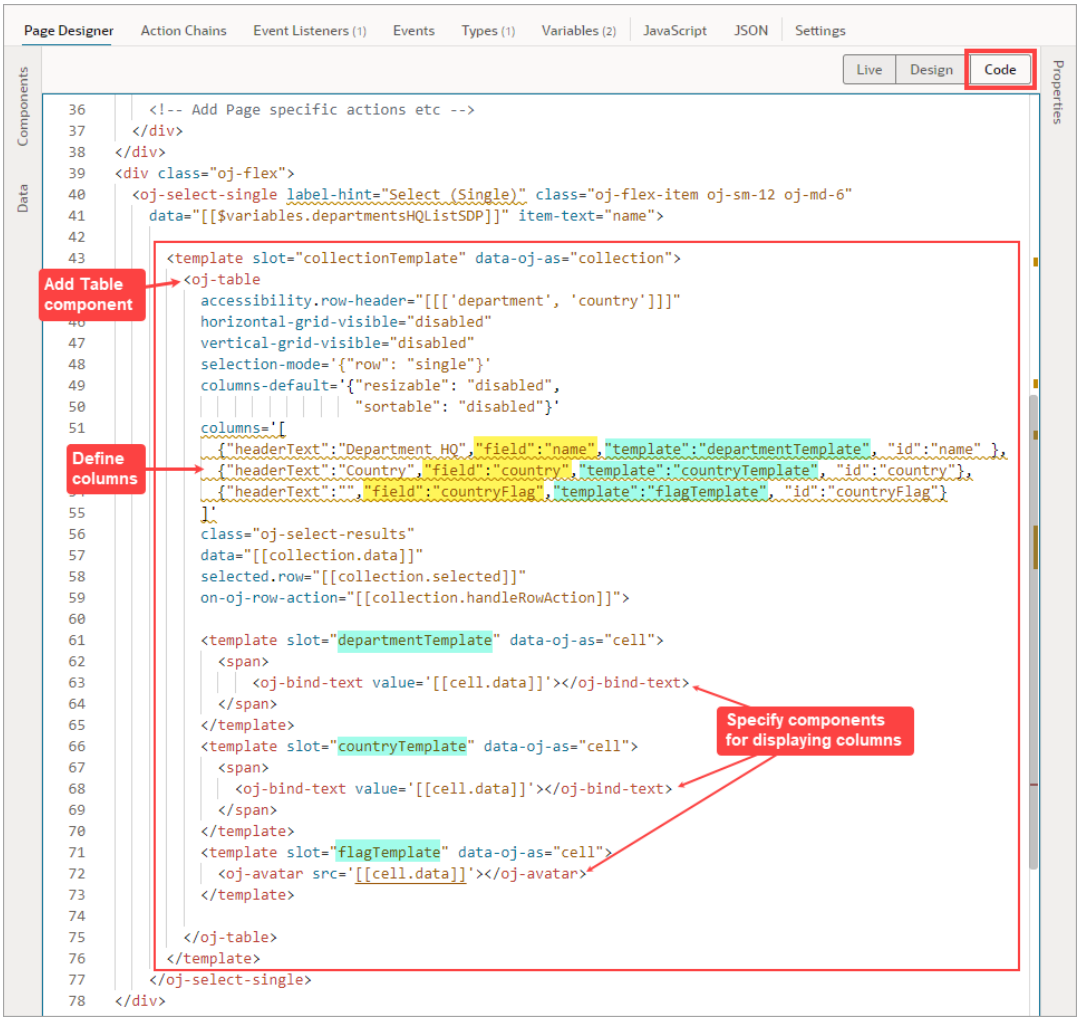


We now need to configure the Single Select component to show additional fields, including an image to show each country's flag.

Click the Page Designer's Code button to edit the page's HTML code. Add the following HTML code to the `oj-select-single` tag, which adds a table to the Single Select component so that it can show additional fields:

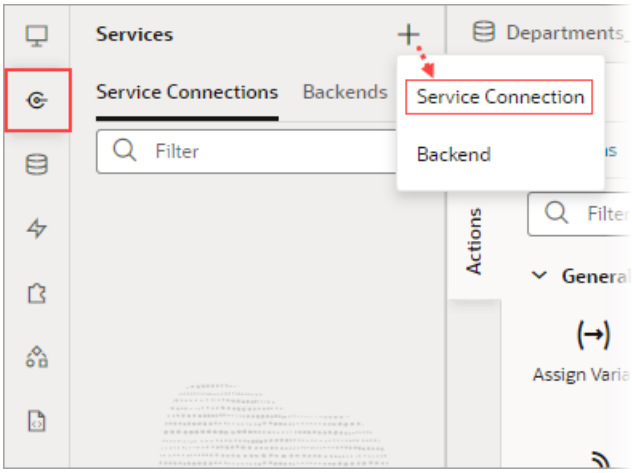
```
      <template slot="collectionTemplate" data-oj-as="collection">
        <oj-table
          accessibility.row-header="[[['department', 'country']]]"
          horizontal-grid-visible="disabled"
          vertical-grid-visible="disabled"
          selection-mode='{"row": "single"}'
          columns-default='{"resizable": "disabled",
                            "sortable": "disabled"}'
          columns='[
            {"headerText":"Department
HQ","field":"name","template":"departmentTemplate", "id":"name" },

{"headerText":"Country","field":"country","template":"countryTemplate",
"id":"country"},
```

```
{"headerText":"","field":"countryFlag","template":"flagTemplate",
"id":"countryFlag"}
          ]'
          class="oj-select-results"
          data="[[collection.data]]"
          selected.row="[[collection.selected]]"
          on-oj-row-action="[[collection.handleRowAction]]">

          <template slot="departmentTemplate" data-oj-as="cell">
            <span>
                <oj-bind-text value='[[cell.data]]'></oj-bind-text>
            </span>
          </template>
          <template slot="countryTemplate" data-oj-as="cell">
            <span>
              <oj-bind-text value='[[cell.data]]'></oj-bind-text>
            </span>
          </template>
          <template slot="flagTemplate" data-oj-as="cell">
            <oj-avatar src='[[cell.data]]'></oj-avatar>
          </template>

        </oj-table>
      </template>
```



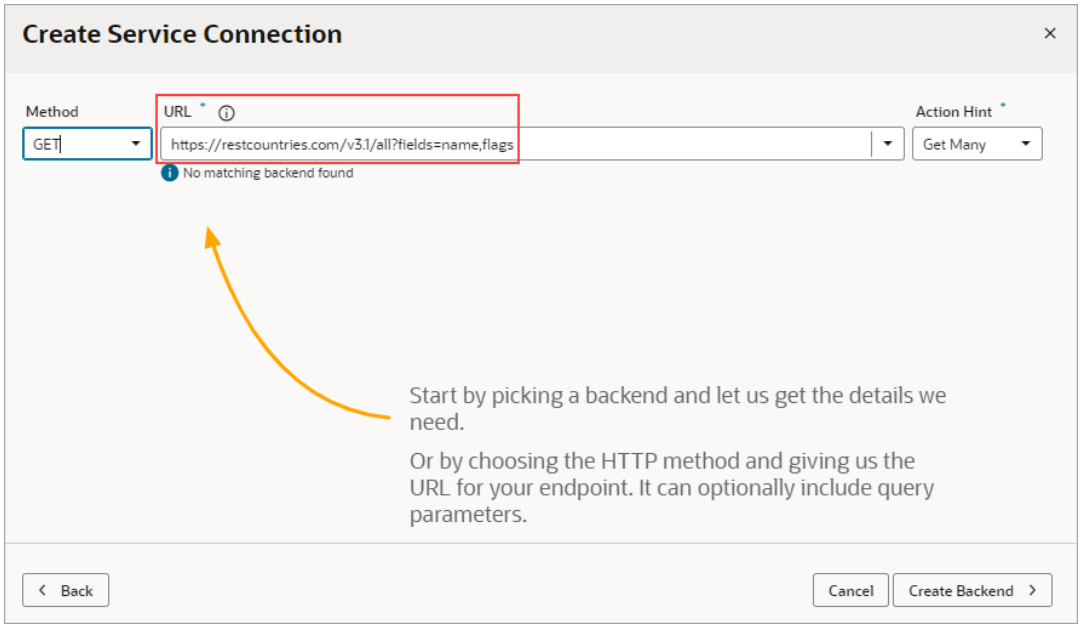
Next, we need to create a service connection for retrieving the flag images.

9. Select the Services pane, then click its plus (+) icon and select **Service Connection**:



10. In the Create Service Connection wizard, under Select Source, select **Define by Endpoint**.

11. In the **URL** field, enter the endpoint `https://restcountries.com/v3.1/all?fields=name,flags` to retrieve the flag images, then click **Create Backend**:



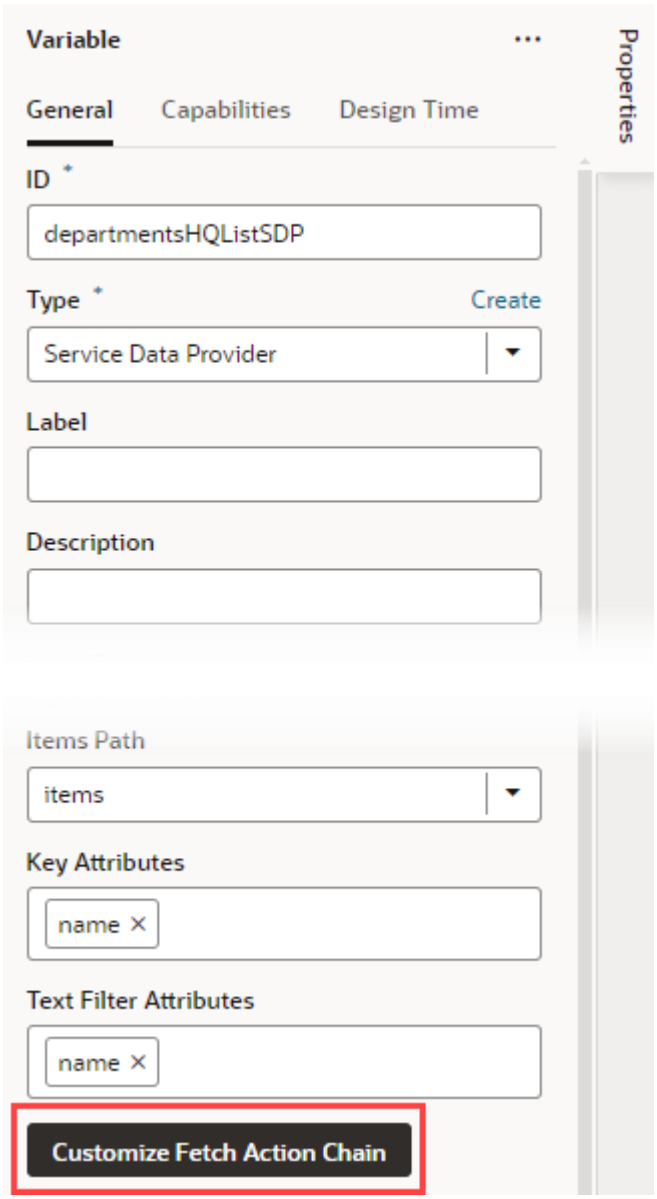
A backend is created for this service connection, which stores the server details. You can use this backend to create related service connections, and to apply endpoint requests and response transform functions to them all.

12. On the Backend Specification step, enter `GetFlagsBackend` as the Backend Name and click **Next**.
13. On the next step, enter `GetFlags` for the Service Name.

Now that the preliminary work has been completed, we can see how to create a custom fetch action chain for an SDP that's bound to a component. We'll first customize the fetch action chain that was automatically created for the SDP when it was mapped to the Departments HQ business object, so that it'll also retrieve flag images. To keep things simple, we won't do any error handling.

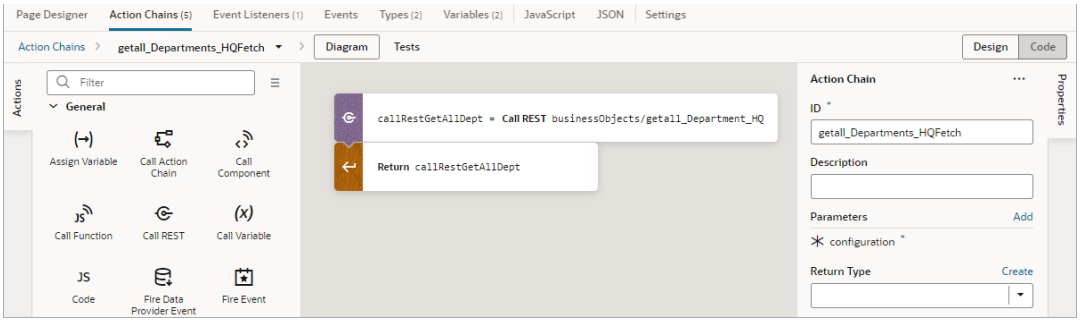
To begin:

1. Go to the page's **Variables** tab and select the SDP that's bound to the Select Single component.
2. In the Properties pane, scroll down to the bottom and click **Customize Fetch Action Chain**:

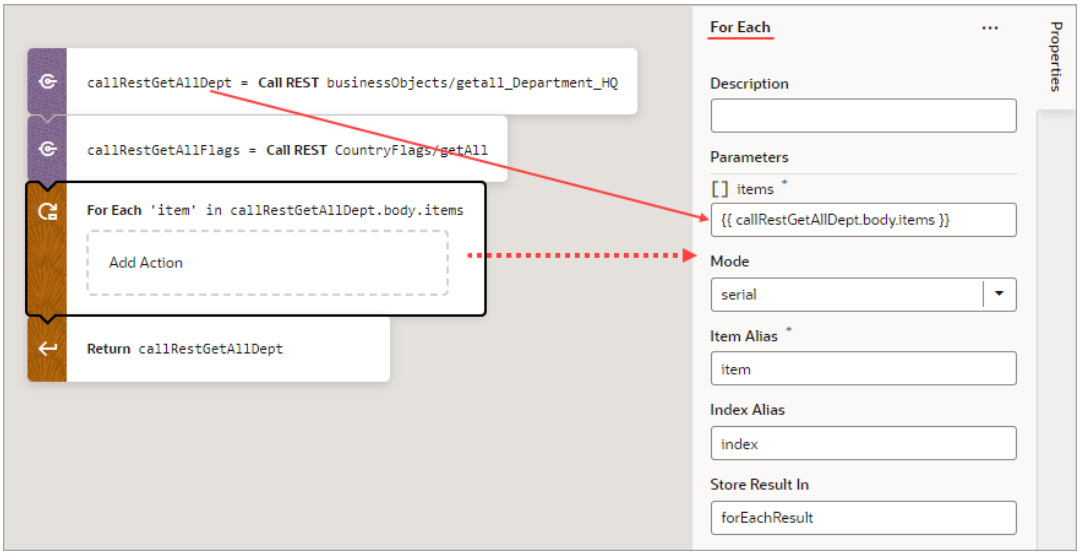


You're taken to the Action Chains editor, where the SDP's fetch action chain is loaded, which has an auto-generated name and a preconfigured Call REST action.

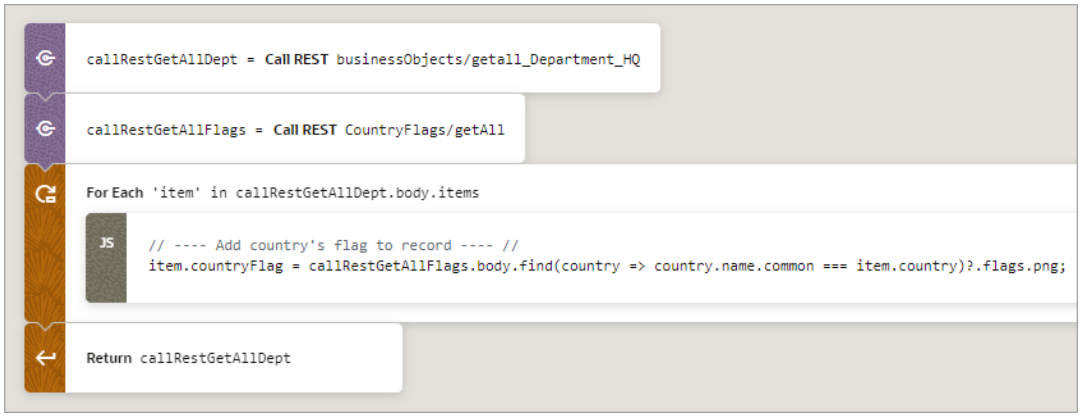
> **Note:**
>
> A `configuration` object has been passed to the action chain, however, it's for internal use only. Don't try to use this object, as it can change in between versions in incompatible ways.



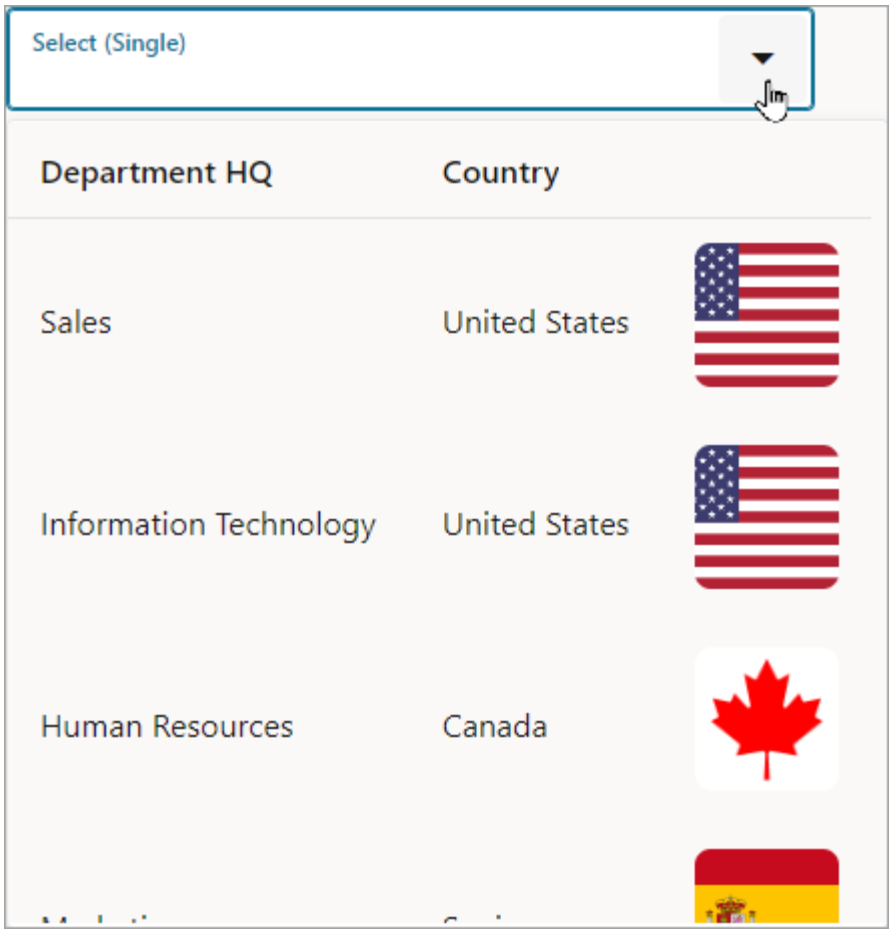
3. Under the provided Call REST action, add another Call REST action.
4. In the Properties pane, click the Endpoint property's **Select** link. In the Select Endpoint dialog, expand the **Services** node, then the **GetFlags** node and select the **GET /all** endpoint. Click **Select**.
5. Add a For Each action under the last Call REST action.
6. In the Properties pane, for the **items** property, enter the location of the returned array of departments using the result from the call to get the departments. For example: `{{ callRestGetAllDept.body.items }}`:



7. Add a JS Code action to the **Add Action** area of the For Each action.
8. In the Properties pane, replace the text in the Code box with this code to add the flag image to the data that's to be returned by the action chain: `item.countryFlag = callRestGetAllFlags.body.find(country => country.name.common === item.country)?.flags.png;`



The custom fetch action chain is complete and ready for you to try out! Now, when you go to view the Select Single component's list, you'll see each department's country and an image of the country's flag:



Delay Display of SDP Data

To improve the performance of your visual application, you can delay fetching of SDP data until it's requested by the user.

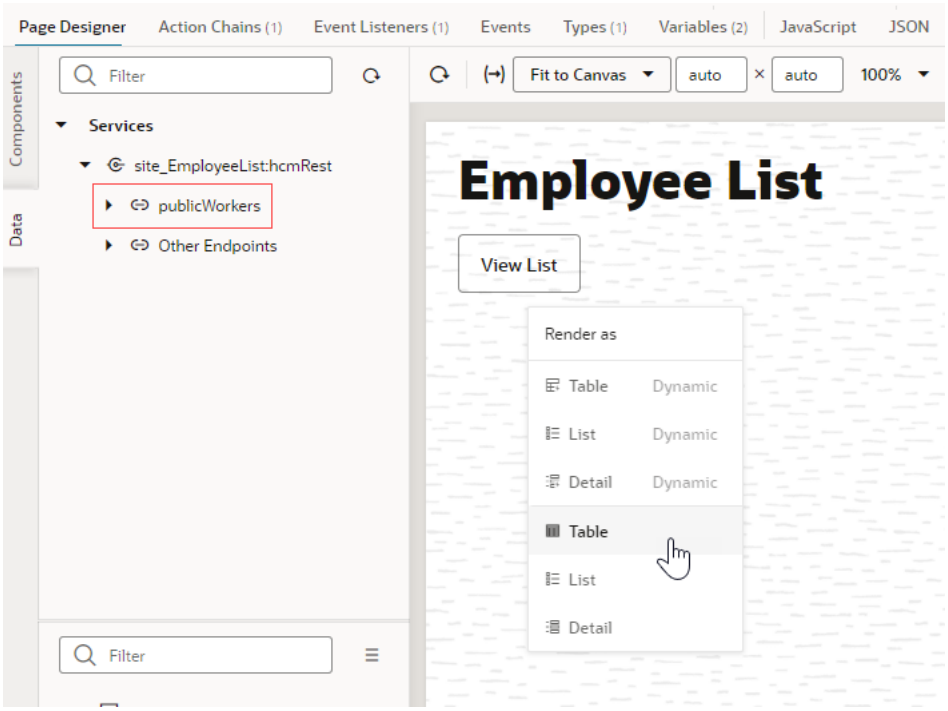
An SDP automatically executes the REST point that it's bound to when the associated UI component is first shown on the page, so if we want to delay the execution of the SDP call, we need to hide the UI component. For example, you can delay display of table data on a page until the user clicks a button.

In this procedure, we use an `oj-bind-if` component to hide a table, then we add an `ojAction` event to a button on the page. When the button is clicked, the variable controlling the `oj-bind-if` component is updated, and the REST call is executed to fetch and display the data in the table. For more information on the `oj-bind-if`component, see Use Conditions to Show or Hide Components.

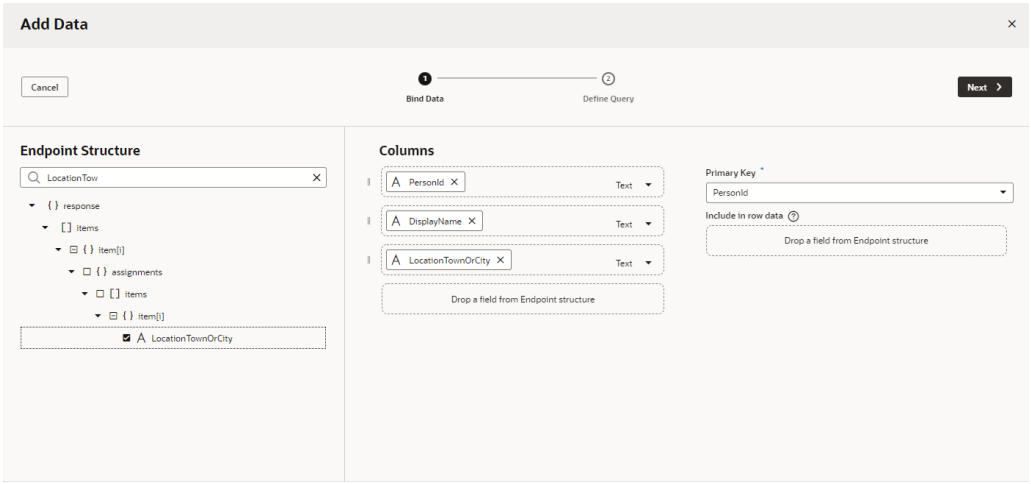
An SDP is bound to REST endpoints that fetch many records, which can come from a service or a business object.

In this example, we've set up a service connection using the **Create Service Connection** wizard to create a Human Capital Management service connection from the catalog and chose the **publicWorkers** object. For more information, see Create Service Connections from the Oracle Cloud Applications or Integration Applications Catalog.

1. In your visual application Page Designer, drag a **Button** component to the canvas and add a label for it in the Properties pane, for example, `View List`.
2. Now create a table using a service connection.
    a. From the Page Designer **Data** tab, expand **Services** and drag an object (for example, publicWorkers) to the canvas. Choose the second **Table** item from the **Render as** list.



b. In the **Bind Data** page, search for and select the endpoints that you want to add as table columns (for example PersonID, DisplayName, and LocationTownOrCity). Click **Next**.



c. In the **Define Query** page, click **Finish**.

3. Create a boolean variable to control the table display.

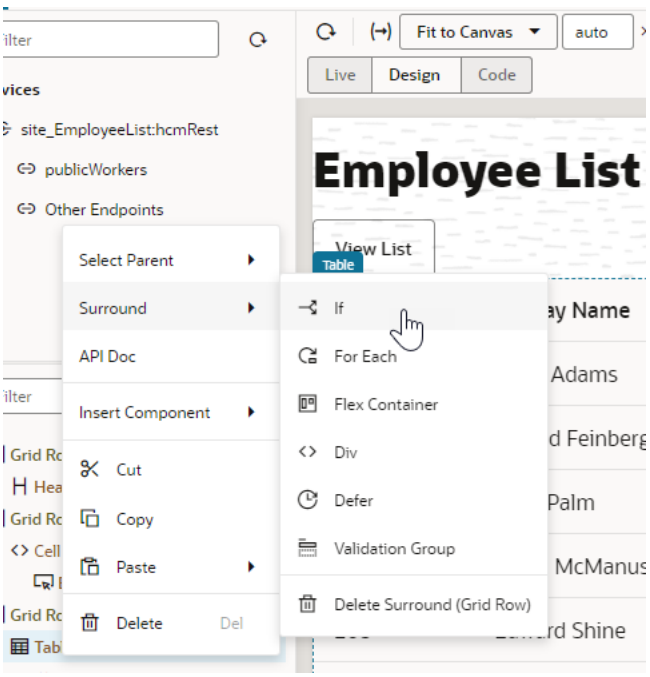
a. In the Variables tab, click **+ Variable**.

b. Update the **ID** to (for example) `EmpList` and choose **Boolean** in the **Type** field.

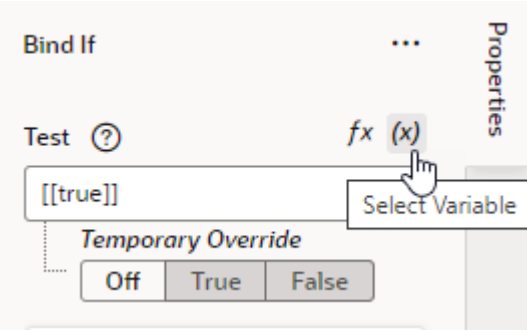
c. Select **false** as the **Default Value**.

4. Use the `oj-table-bind` component to hide the table.

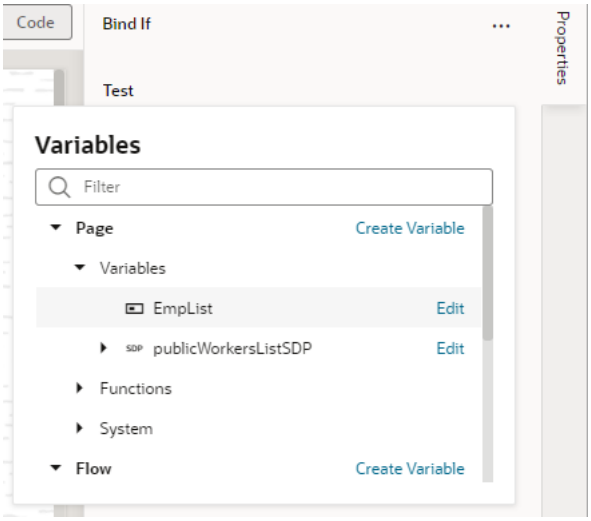
a. In the Structure tab, right-click the **Table** component and select **Surround**, then **If**.



b. In the Structure tab, select the **Bind If** component. In the Properties pane, hover over the **Test** field and click to open the variables picker.



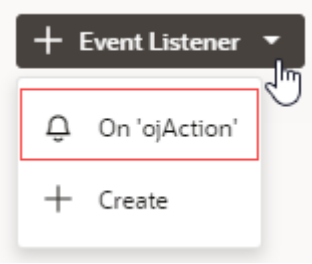
c. Select **EmpList** from the Variables list.



The table is hidden.

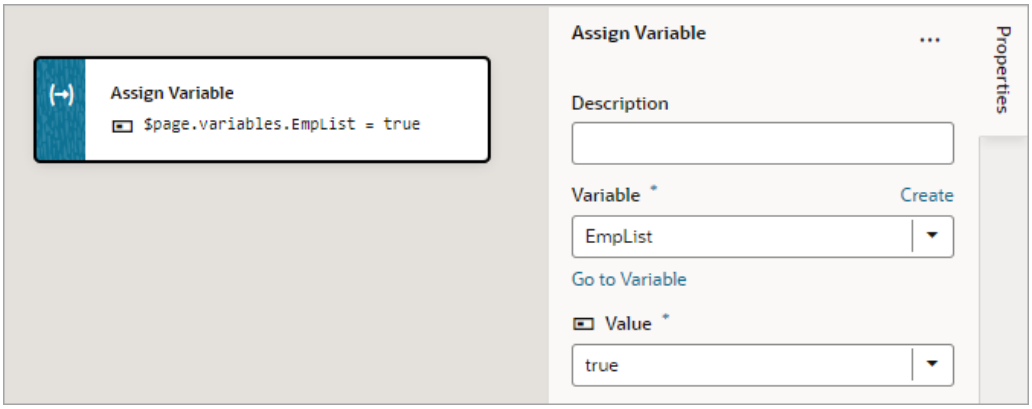
5. Create an `oj-action` event for the button.

a. Select the button component in the Designer, then in the **Events** tab of the Properties pane, click **+ Event Listener** and select **On 'ojAction'**.



You're taken to the Action Chain editor.

b. Add an **Assign Variables** action to the canvas. In the action's Properties pane, select **EmpList** in the **Variable** list and **true** in the **Value** list.



Now when users reach the page, they will need to click the button to view the table.

Speed Up Sorts on Business Object Fields

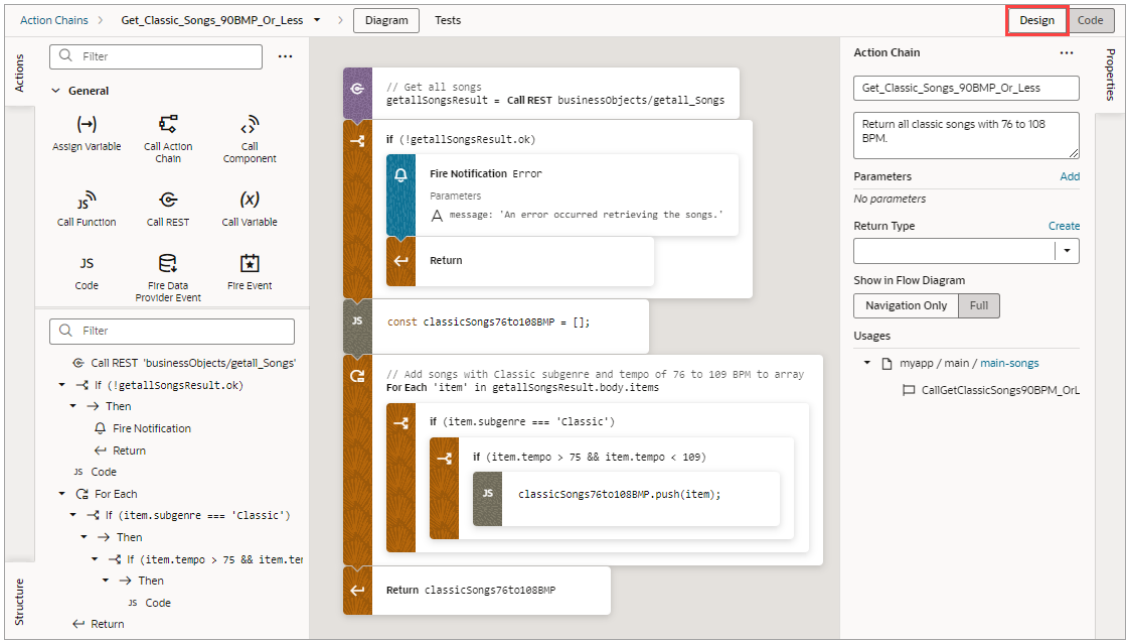
If your SDP is pulling data from a business object, and you're having performance issues with a sort operation (using the SDP's `orderBy` or `sortCriteria` parameter) on a business object's field, index the field as explained here.

10

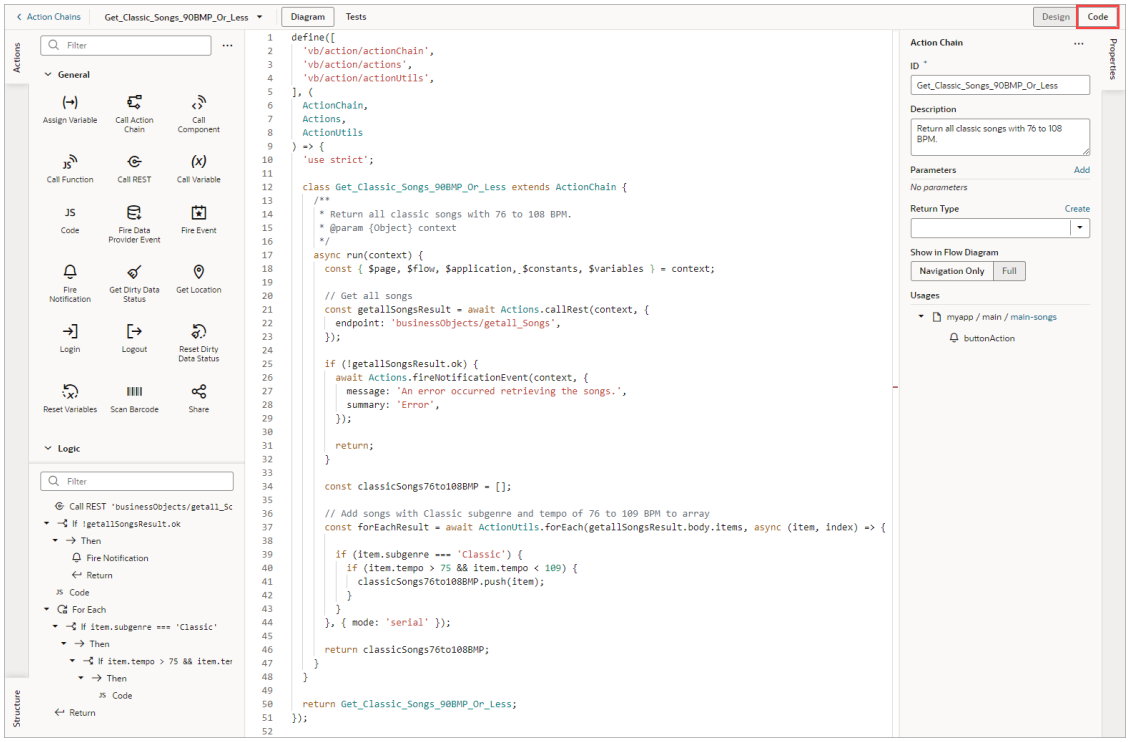
Work With JavaScript Action Chains

A JavaScript *action chain* is a sequence of actions started by an *event*. When a given event occurs in a page, the *event listener* listening for that event kicks off the action chain.

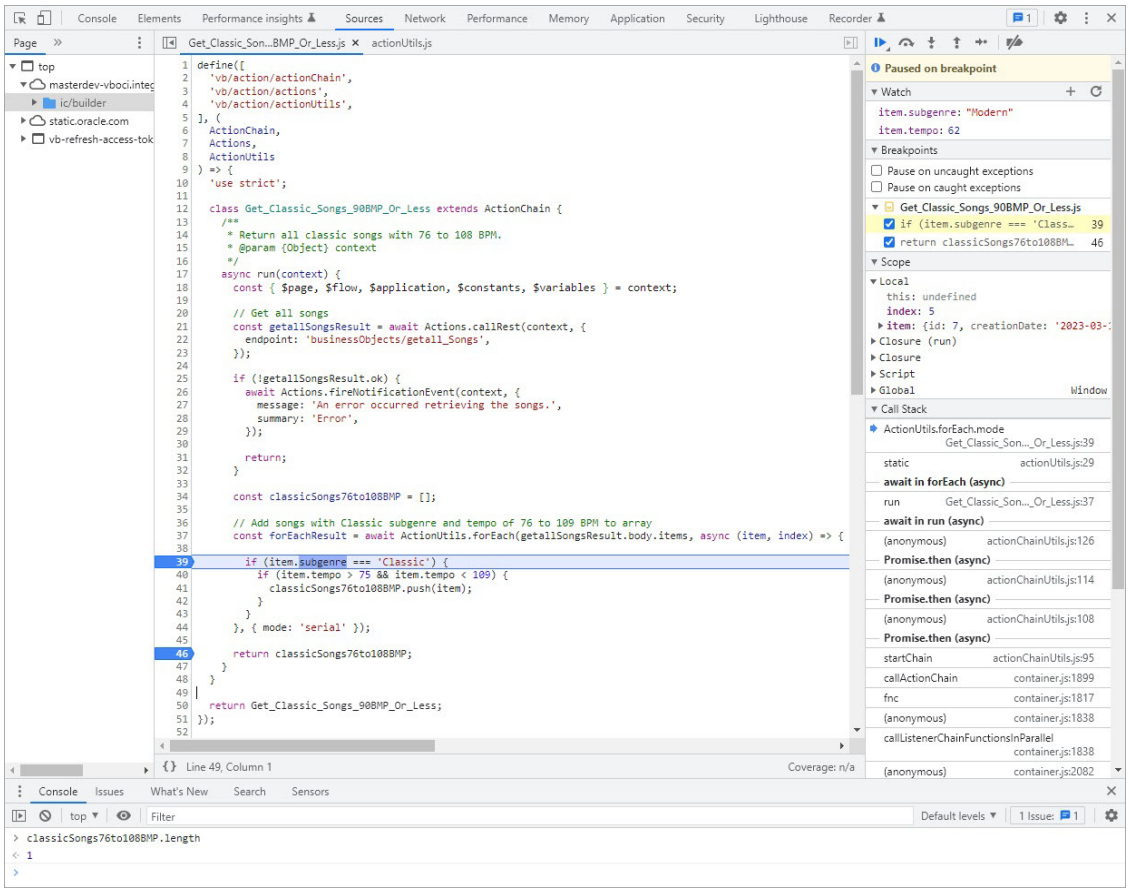
You implement action chains using either the visual Action Chain editor or through code, using JavaScript. Here's an example of an action chain built with the Action Chain editor. The action chain first calls a REST endpoint to get a full set of songs, then checks to see if any errors occurred. As long as things remain error-free, the action chain loops through the songs and adds those with a "Classic" subgenre and a tempo of 76 to 108 beats per minute to an array. The array is then returned:



Here's the same action chain in the code editor:



You can also debug JavaScript action chains using your browser's Developer tools:



About Action Chains

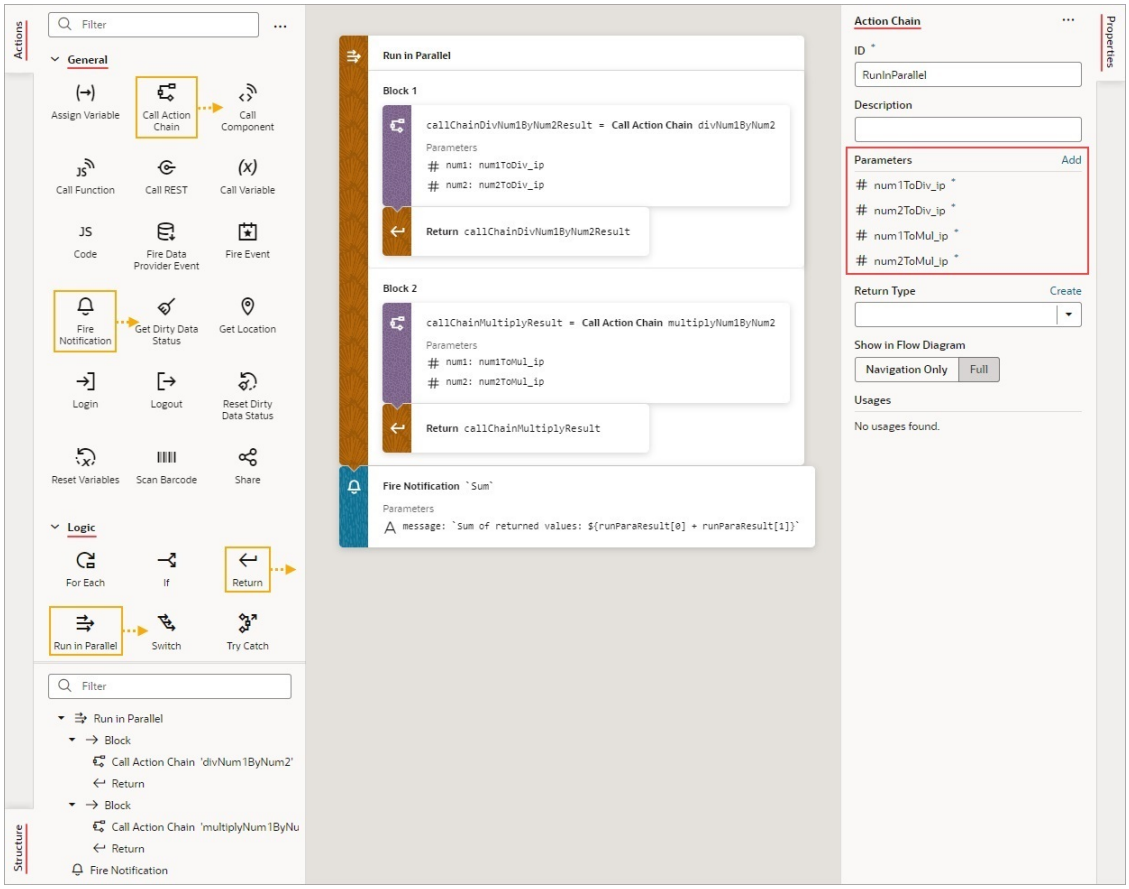
An action chain drives a series of actions in response to a lifecycle event from the user interface. Events are what start them, and there are many types of events, such as:

- `vbEnter`: triggered when a page starts and can be used to fetch data
- `ojAction`: triggered when a button component is clicked
- `onValueChange`: triggered when the value stored in a variable changes

No matter the type of event, every action chain must be bound to an event listener to be able to run it. Sometimes the event listener is created automatically, but sometimes you must create it explicitly. For example, if you accept the event that Visual Builder suggests (say, the `onValue` event suggested for an Input Text component), the event listener is created for you, which will trigger an action chain when the component's value changes.

Creating an action chain involves using the Action Chain editor to assemble predefined (built-in) actions into a sequence that performs the required task. If you need an action that isn't available, you can either use the Code action to add your own block of code, or you can create a custom action if you think you might need it again.

Here's an example of an action chain that runs two action chains asynchronously, and then uses the result from each to create a combined result. Through input parameters, the action chain receives four numbers, as shown in the Properties pane. Using the **Run in Parallel** action, one `asyn()` method is used to call an action chain that returns the quotient of two numbers, and another `asyn()` method is used to call an action chain that returns the product of two numbers. The **Run in Parallel** action returns an array (`runParaResult`, in this example), with the first element containing the value from the first `asyn()` method and the second element containing the value from the second `asyn()` method. The sum of the values is then displayed using a **Fire Notification** action:



Here's the action chain's code:

```
      const runParaResult = await Promise.all([
        async () => {

          const callChainDivNum1ByNum2Result = await
Actions.callChain(context, {
            chain: 'divNum1ByNum2',
            params: {
              num1: num1ToDiv_ip,
              num2: num2ToDiv_ip,
            },
          });

          return callChainDivNum1ByNum2Result;
        },
        async () => {

          const callChainMultipleNum1ByNum2Result = await
Actions.callChain(context, {
            chain: 'multipleNum1ByNum2',
            params: {
              num1: num1ToMul_ip,
              num2: num2ToMul_ip,
            },
          });
```

```
        return callChainMultipleNum1ByNum2Result;
      },
    ].map(sequence => sequence()));

    await Actions.fireNotificationEvent(context, {
      message: `Sum of returned values: ${runParaResult[0] +
runParaResult[1]}`,
      summary: `Sum`,
    });
```

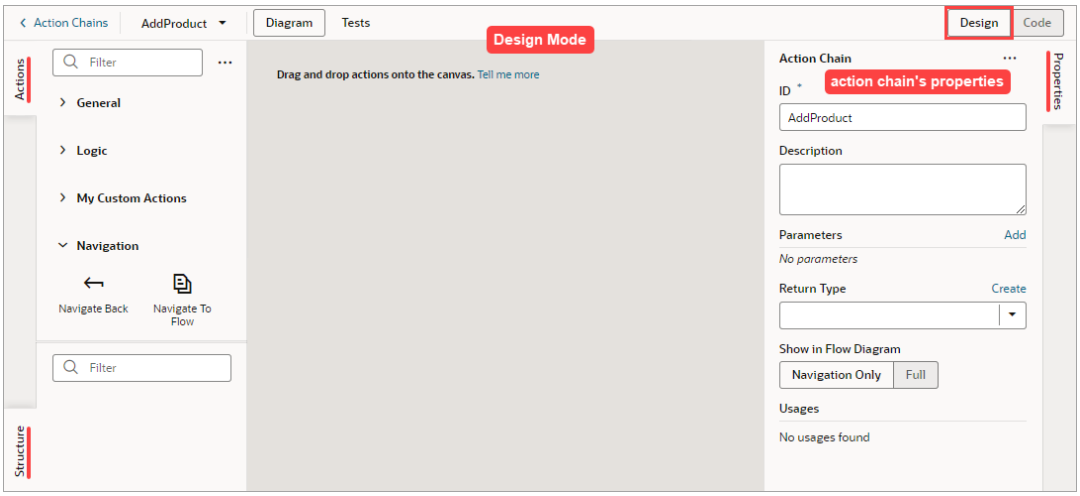
When creating action chains, keep in mind that each action chain has a scope that depends on where it's defined: at the application, flow, page, or fragment level. An action chain defined at the application level can be called from any flow or page, but a page-level action chain can only be called from that page— however, the chain itself can access variables defined on the page, parent flow, or application. The same goes for flow-level action chains. A fragment-level or layout-level action chain can only be called from that fragment or layout, and the chain can only refer to variables defined in that fragment or layout.

While actions within a particular chain run serially, you can run multiple action chains concurrently by configuring the event listener to start multiple chains.

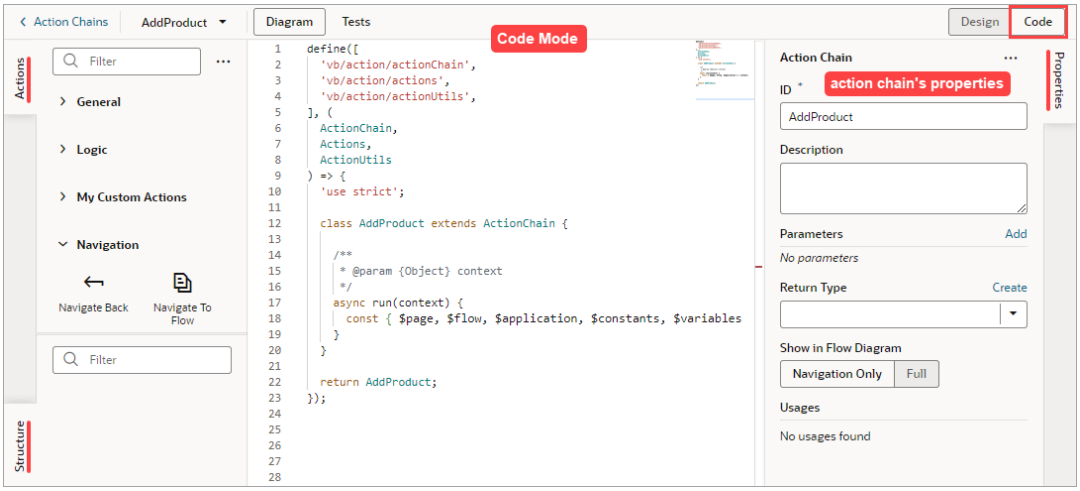
About the Action Chain Editor

The Action Chain editor has two modes for creating an action chain, which you can seamlessly switch between, as changes in one are immediately reflected in the other:

- Design mode is used to visually create an action chain:



- Code mode is used to create an action chains with code:

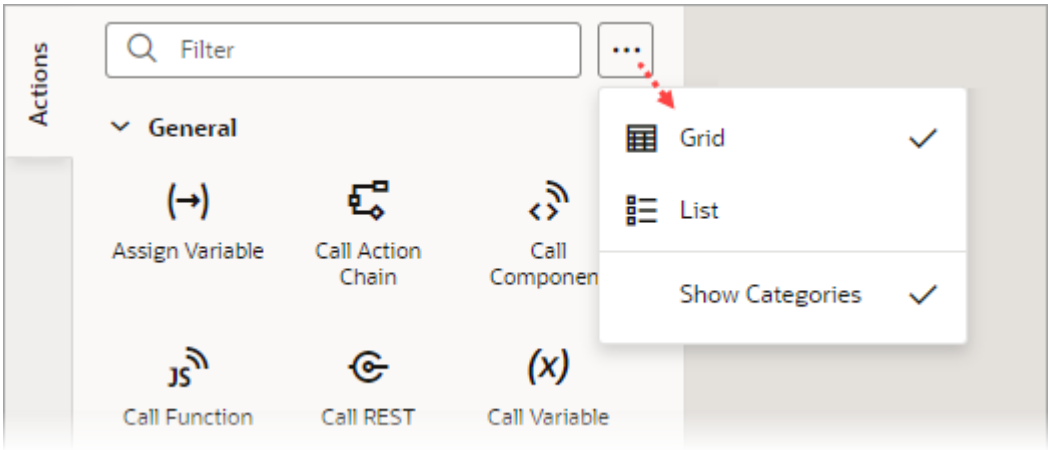


Both modes have an Actions palette, a Structure pane, and a Properties pane:

- **Actions Palette**:
  The Actions palette provides built-in actions, organized into categories, for creating actions chains. As mentioned, if none of the actions meet your need, you can use the Code action to add your own block of code, or you can create a custom action if a future need warrants it.

  When a local function is added, it's added to the Actions palette, under a newly added Local Functions category. Use it to quickly add a call to the local function.

  To customize the Actions palette, click its Menu ☰:



  You can choose to view the actions in a grid or list; and **Show Categories** groups the actions into categories when selected, and lists them alphabetically otherwise. These preferences are saved for each action chain.

- **Properties Pane**:
  The Properties pane provides an easy way to define an action chain's input parameters and return object, and an action's properties.

  When entering text in the Properties pane, an entry is considered a string unless it's wrapped with double curly brackets, like this {{2+3}}, in which case it's considered a direct expression.

  If a local function is added, you can select it and use the Properties pane to view and modify its properties.

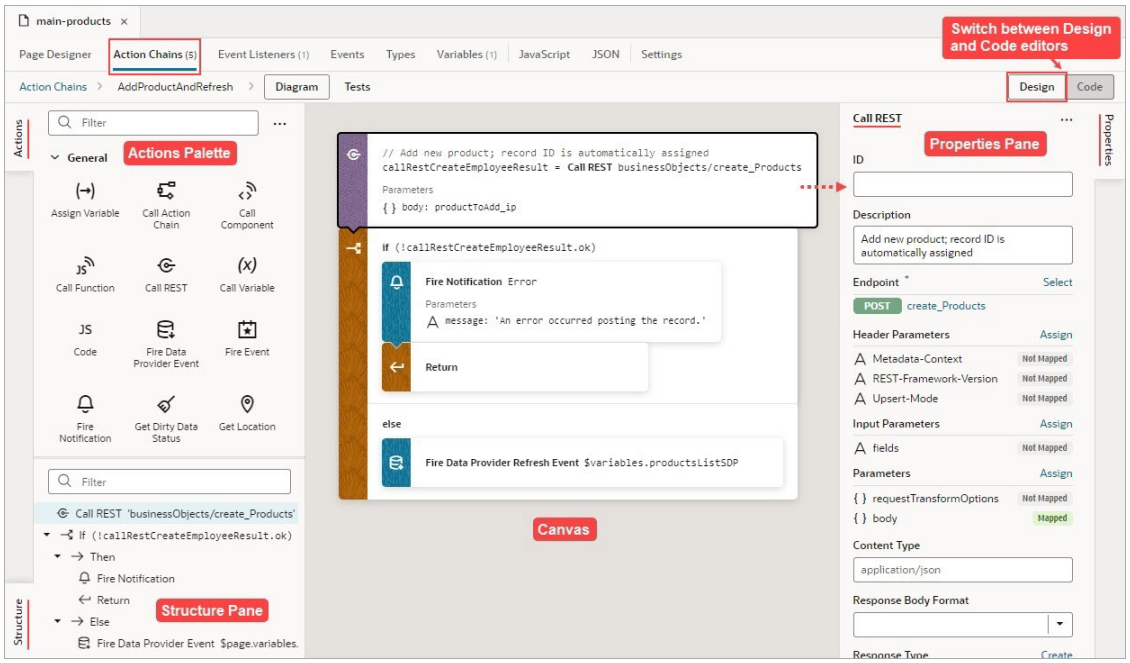
- **Structure Pane**:
  The Structure pane provides a compact view of the actions, and has a filter to quickly find and select an action. Also, syntax errors detected by the JavaScript parser are shown, if they can be handled, otherwise a message states that the Structure pane can't be displayed due to errors.

  When a local function is added, the Structure pane gets organized by functions and their actions.

You can hide any of these panes by clicking their tab.

Create Action Chains in Design Mode

When you use the Actions palette, Properties pane, and Structure pane in Design mode, VB Studio writes the corresponding JavaScript behind the scenes. You can edit this code directly at any time by switching to Code mode, which opens a code editor.

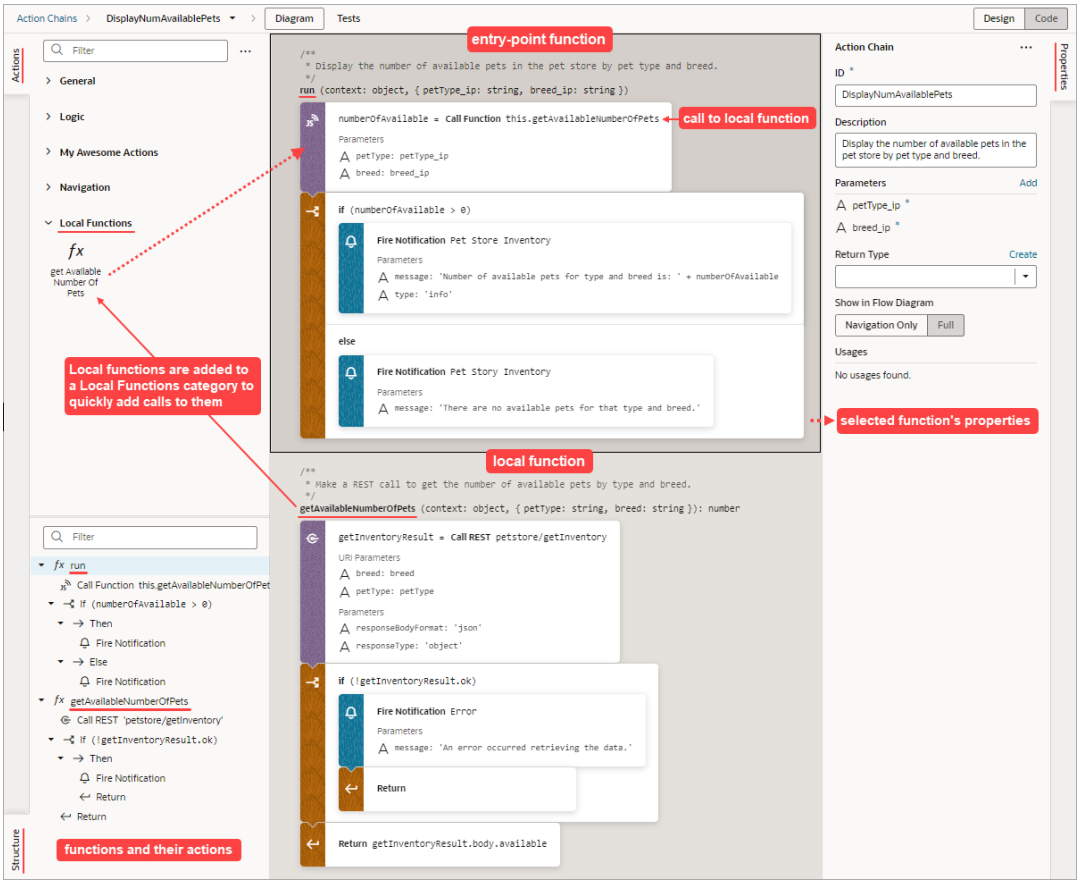


Right-click an action to display a context menu with the following options:

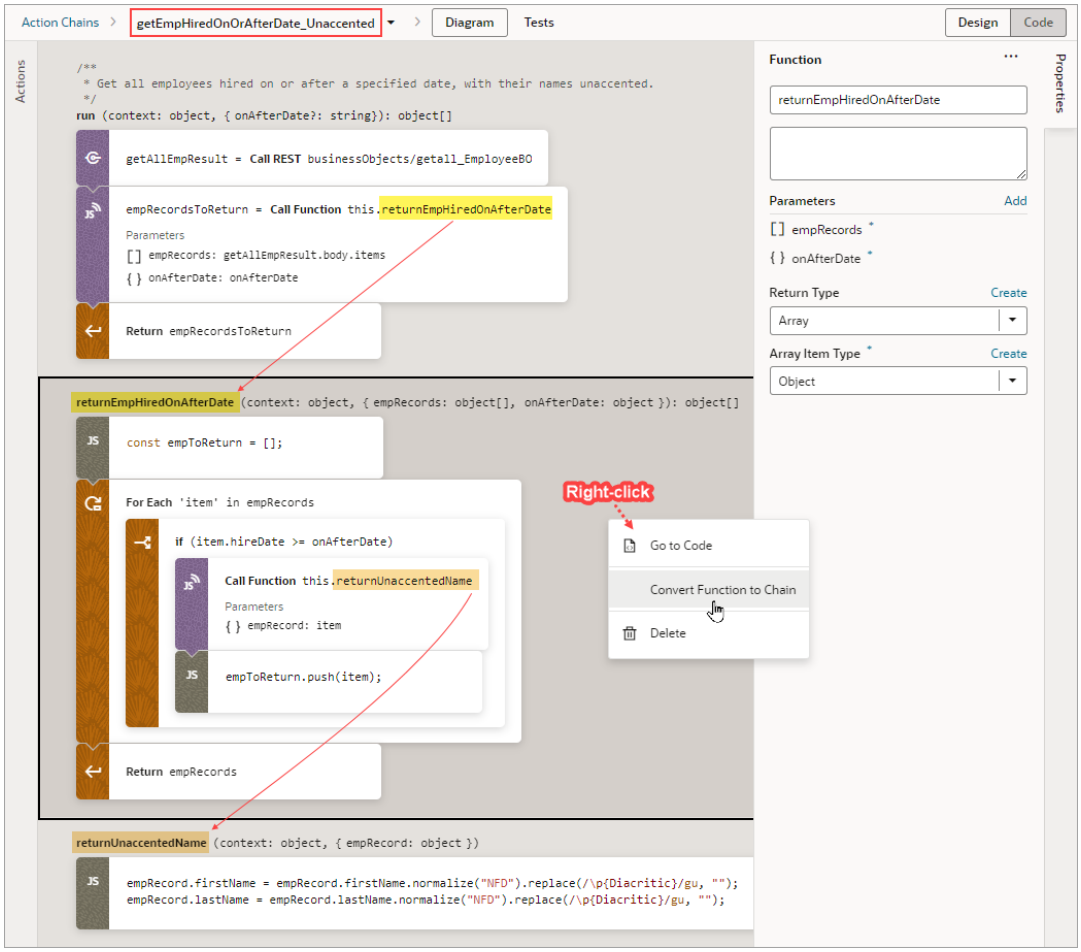
- **Go to Code**: Go to the action's code in the code editor.
- **Surround with If**: Surround the action with an If action.
- **Surround with Try-Catch**: Surround the action with a Try-Catch action.
- **Duplicate**: Duplicate an action, or a code block within an If, Switch, or Run in Parallel action. For an If action, you can’t duplicate an Else block, and for a Switch action, you can’t duplicate the Default block.
- **Delete**: Delete the action.

Here are some helpful tips for working with action chains:

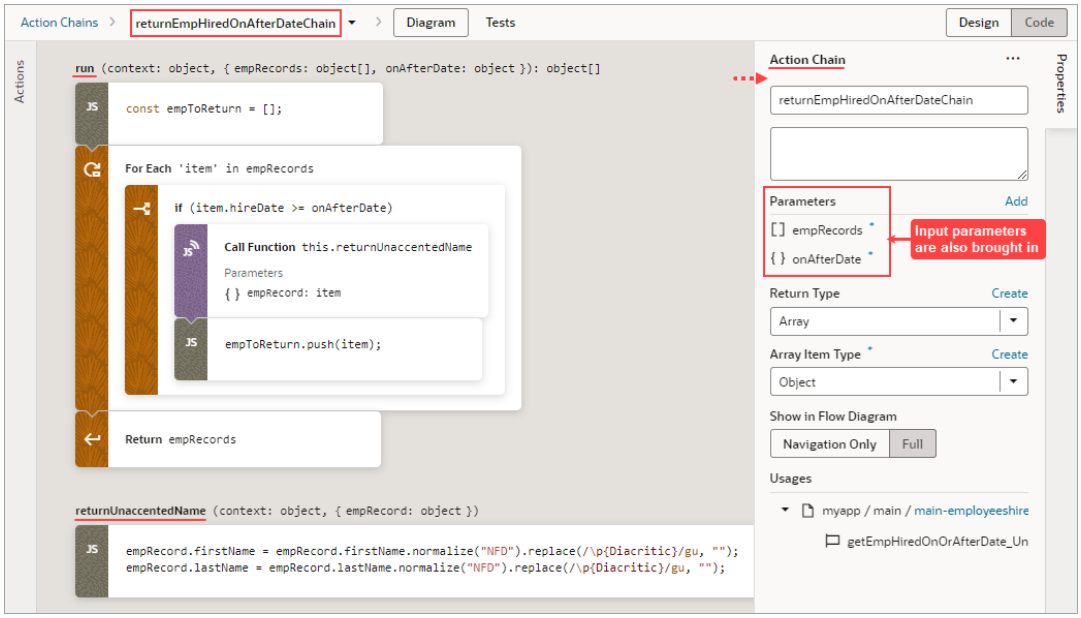
- When you add a local function, the function and its actions are added to the Structure pane for quick navigation; the function’s properties appear in the Properties pane; and the local function is added to the Actions palette, under a newly created Local Functions category:



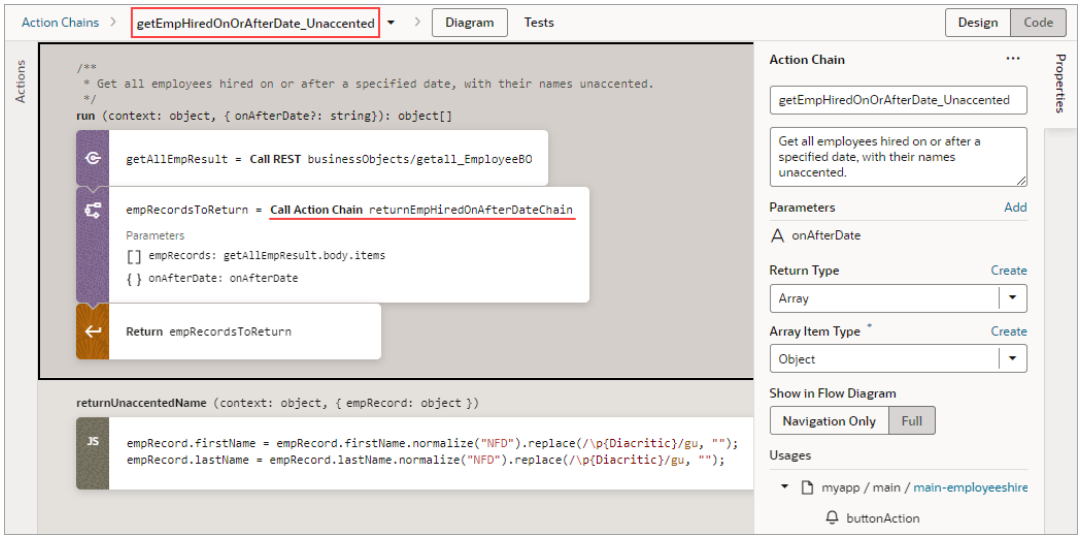
- To convert a local function to an action chain so that it can be used by other action chains, right-click the local function and select **Convert Function to Chain**:



The local function, as well as any local functions it uses, is converted to an action chain:

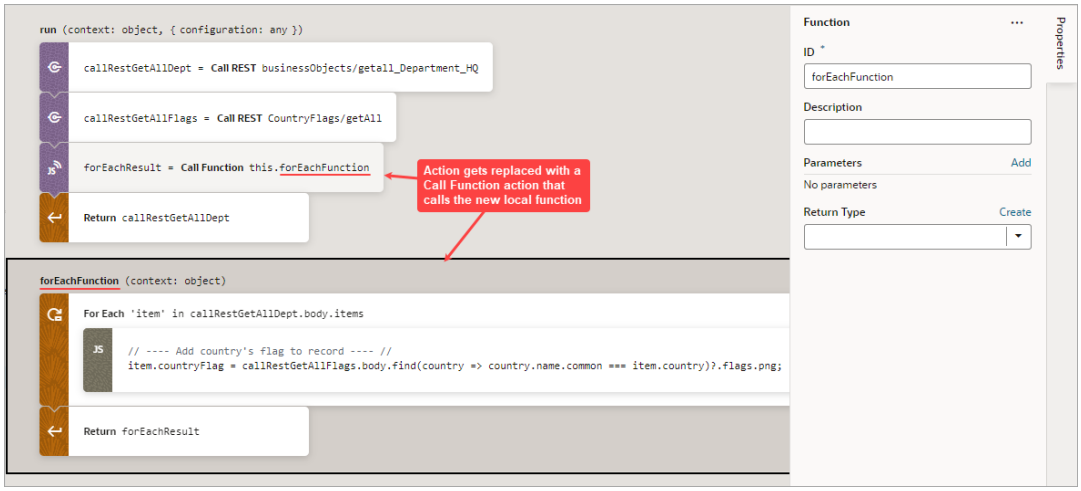


In the original action chain, the Call Function action that called the local function is converted to a Call Action Chain action that calls the new action chain:

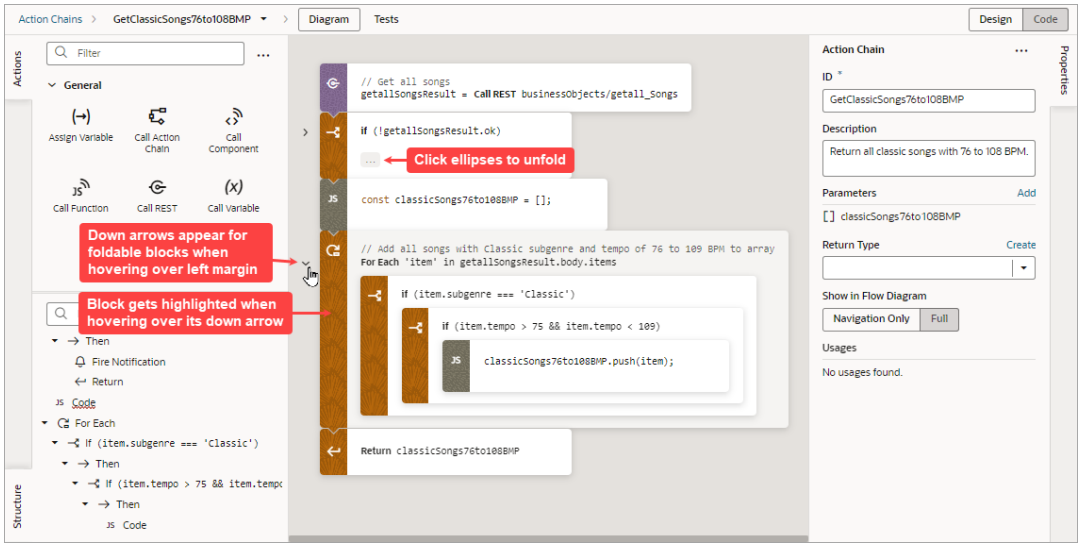


- You can also create a local function from an action on the canvas. Just drag an action from the canvas onto the **Create Function** button that appears on the bottom right of the canvas, or onto the green line that appears before or after a local function. In this example, we create a local function by dragging a For Each action onto the Create Function button:

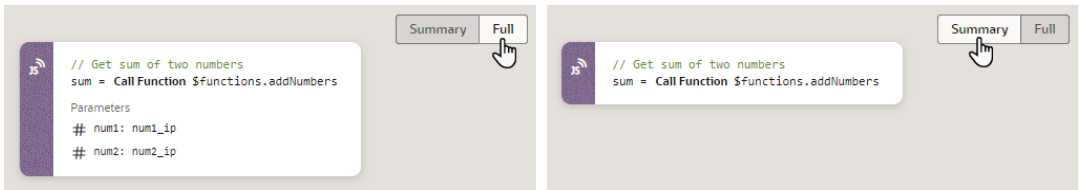
In the `run()` entry point function, the For Each action was replaced with a Call Function action that calls the new local function:



- To visually simplify large action chains, you can fold the code blocks for If, Switch, Run in Parallel, and For Each actions. To do so, hover over the canvas's left margin and click the down arrows for the blocks of code to fold. To unfold a block, click the corresponding right arrow or ellipses on the action card:



You can further visually simplify an action chain by hovering over the canvas's upper-left corner and clicking the Summary button that appears. The Summary button hides the input parameter details for each action, except the Assign Variable and Reset Variables actions. The Full button switches back to displaying them:



- If you prefer to construct your action chain using code, click the **Code** button at the top-right of the screen to open a code editor. Typically, one uses Design mode to visually add

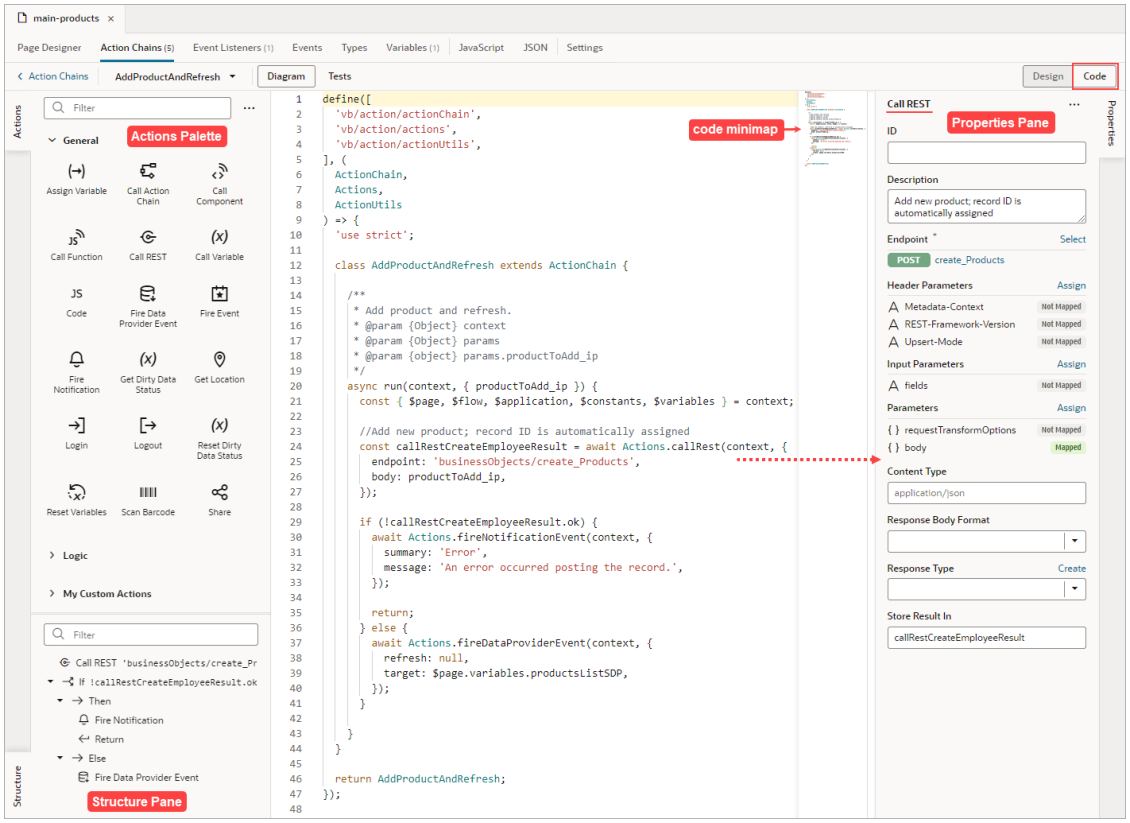
and configure an action, then switches back to Code mode to work directly with the code, as needed.

Create Action Chains in Code Mode

You use the Code mode to create an action chain using JavaScript code. The code editor may be familiar to you if you've used VS Code, as both are based on the Monaco Editor.

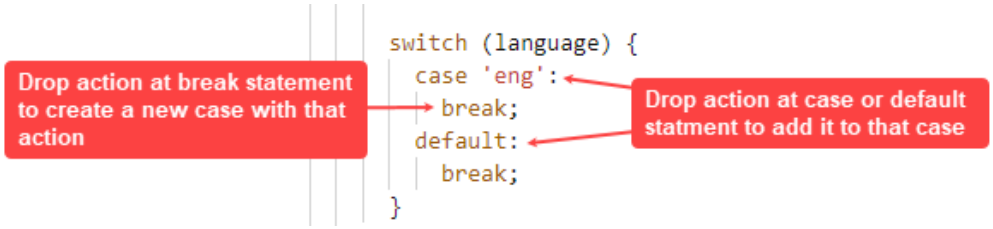
To help you write the code, use the editor's Actions palette, Properties pane, and Structure pane like so:

- **Actions Palette**: Add an action's code by dragging and dropping it from the Actions palette onto the desired place in the editor.
- **Properties Pane**: Define an action chain's input parameters and return object, and an action's properties (code is updated accordingly).
- **Structure Pane**: Quickly find and select an action.



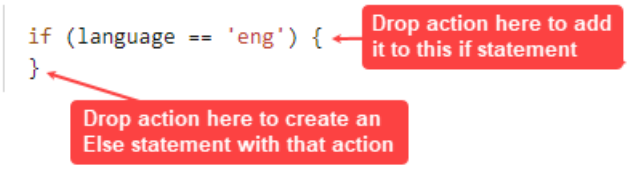
For the Switch and If actions, you can add actions to a clause, or create a new one, depending on where you drop the action:

- **Switch**:

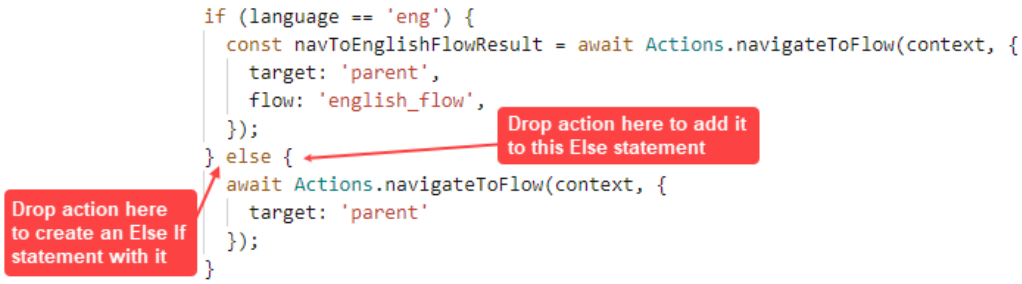


- **If**:

  –

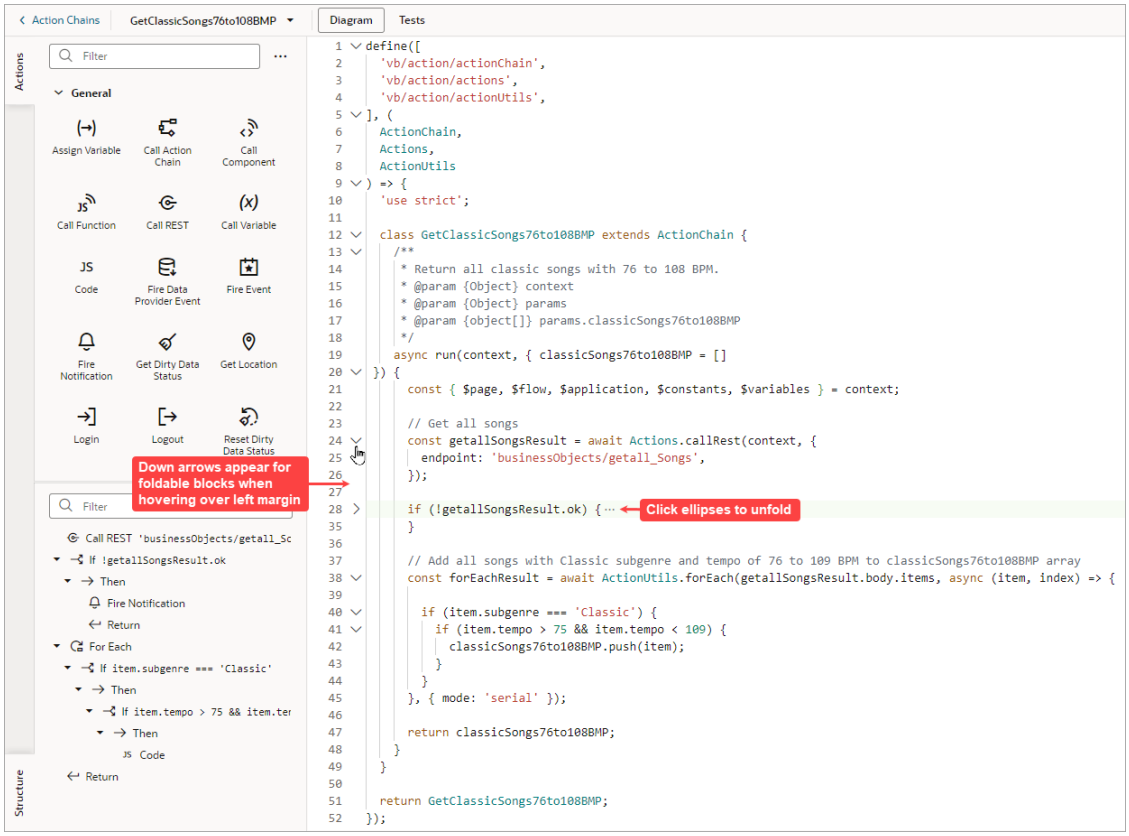


  –



If you add a local function, the function and its actions are added to the Structure pane, and you can view and modify the properties for a selected function in the Properties pane. The local function also gets added to the Actions palette, under the newly created Local Functions category, for you to quickly add a call to it.

To help you focus on currently relevant code blocks, you can fold the code blocks for If, Switch, Run in Parallel, and For Each actions. You can also fold block comments. To do so, hover over the code editor's left margin and click the down arrows for the blocks of code that you want to fold. To unfold a block, click the corresponding right arrow or ellipses:

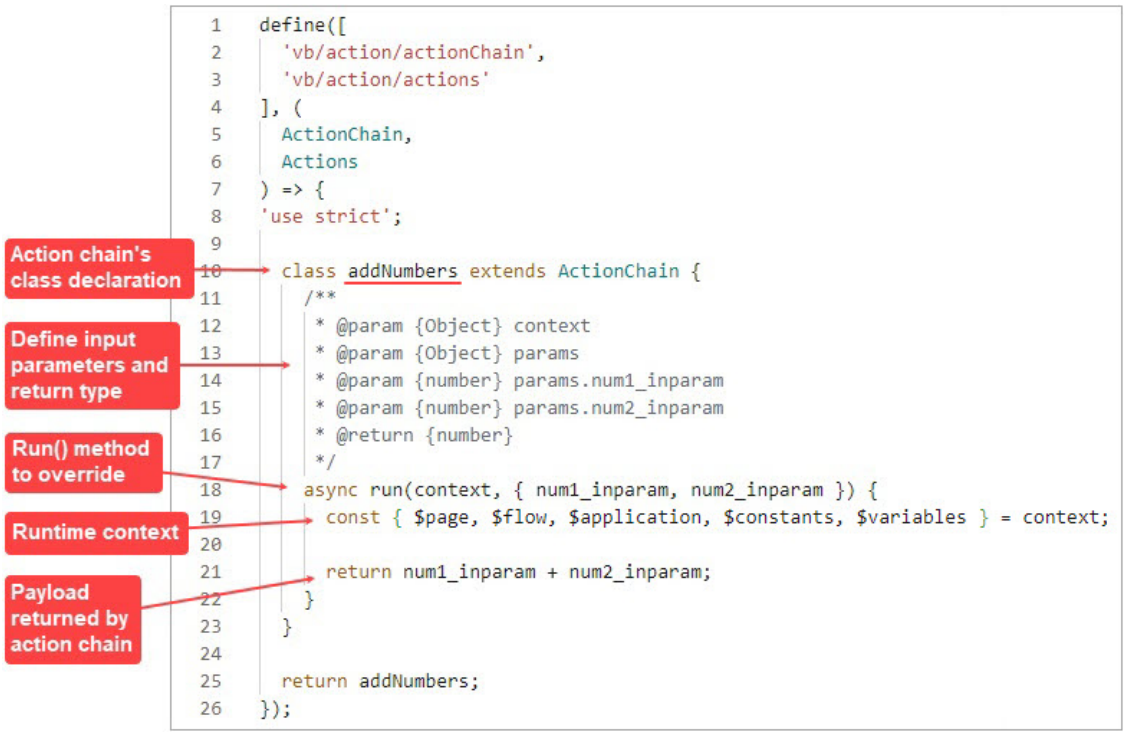


If you'd like to change the editor's appearance, right-click on the editor and select **Editor Settings** from the context menu. You'll be taken to the `settings.json` file where you can customize Visual Builder's code editors. To disable the code minimap, add this entry to the `settings.json` file: `"editor.minimap.enabled": false`. For further details, see Manage Code Editor Settings.

About the Action Chain Code

When you create a new action chain, Visual Builder creates a code file with the basic class declaration for your new action chain. All you need to do is specify the input parameters and return payload types, if any, and to override the `run()` function. You can do all this through the code editor, visually through the Action Chain editor, or both. You can also create local functions, as needed.

Here's an example of a simple action chain that returns the sum of its two input parameters:



The availability of a variable within an action chain depends on the variable's scope. Variables defined in a subscope aren't available in a higher scope. For example, if an action chain is at the flow level, page-scoped variables (`$page`) aren't available.

When referencing a variable, constant, or function, the scope must be specified, unless it's defined in the current scope. If the scope isn't specified, it means it's the current scope. For example, on a page, the page's variables and functions would be referenced as `$variables.myVar` and `$functions.myFunc` instead of `$page.variables.myvar` and `$page.functions.myfunc`.

> **Note:**
>
> It is strongly recommended that you do not use reassignments of context variables (example: `const page = $page` or `const pageVariables = $page.variables`), since audits and action chain tests rely on detecting usages of variables using string searches. You should always reference objects fully, for instance: `$page.variables.var1`.

To call a built-in action, use this format:

```
Actions.<actionName>(context, {
   param1: val1,
   param2: val2,
});
```

To call a custom action, use this format, where the `module` parameter specifies the custom action's ID:

```
Actions.runAction(context, {
   module: '<custom-action-ID>',
   parameters: {
      param1: val1,
      param2: val2,
   },
});
```

Here are details about the parameters for these APIs:

API Part	Details
<actionName>	Name of action.
Context	The runtime context.
parameters	Action-specific parameters object.
<custom-action-ID>	Custom actions ID, as set in the custom action's JSON file.
options	Optional; Object that holds the action's properties for testing or tracing purposes. Currently, it can contain the action's ID.

For details about the API parameters for each built-in action, see JavaScript Actions in the *Oracle Visual Builder Page Model Reference*.

Local Functions

Should the need arise to break up the `run()` entry point function into modular parts, you can create local functions:

```
class DisplayNumAvailablePets extends ActionChain {

  /**
   * Display the number of available pets in the pet store by pet type and breed.
   * @param {Object} context
   * @param {Object} params
   * @param {string} params.petType_ip
   * @param {string} params.breed_ip
   */
  async run(context, { petType_ip, breed_ip }) {
    const { $page, $flow, $application, $constants, $variables } = context;

    const numberOfAvailable = await this.getAvailableNumberOfPets(context, { petType: petType_ip, breed: breed_ip });

    if (numberOfAvailable > 0) {
      await Actions.fireNotificationEvent(context, {
        summary: 'Pet Store Inventory',
        message: 'The number of available pets for this type and breed is: ' + numberOfAvailable,
        type: 'info',
      });
    } else {
      await Actions.fireNotificationEvent(context, {
        summary: 'Pet Story Inventory',
        message: 'There are no available pets for that type and breed.',
      });
    }
  }

  /**
   * Make a REST call to get the number of available pets by type and breed.
   * @param {Object} context
   * @param {Object} params
   * @param {string} params.petType
   * @param {string} params.breed
   * @return {number}
   */
  async getAvailableNumberOfPets(context, { petType, breed }) {
    const { $page, $flow, $application, $constants, $variables } = context;

    const getInventoryResult = await Actions.callRest(context, {
      endpoint: 'petstore/getInventory',
      uriParams: {
        breed: breed,
        petType: petType,
      },
      responseBodyFormat: 'json',
      responseType: 'object',
    });

    if (!getInventoryResult.ok) {
      await Actions.fireNotificationEvent(context, {
        message: 'An error occurred retrieving the data.',
        summary: 'Error',
      }, { id: 'callRest_PetStoreError' });

      return;
    }

    return getInventoryResult.body.available;
  }
}
```

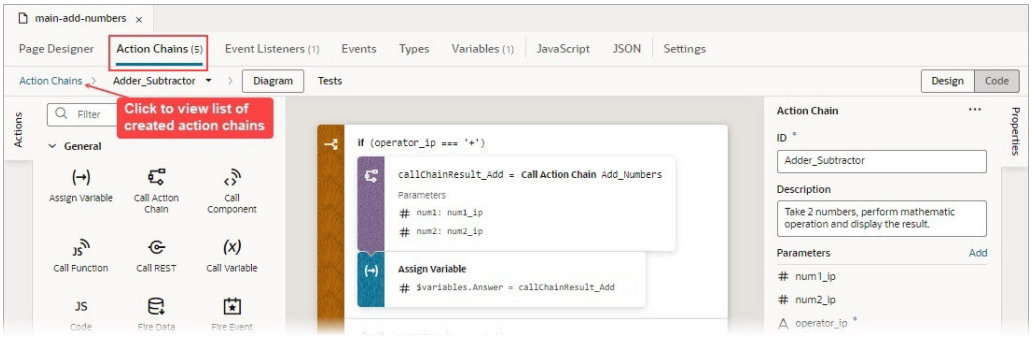

Visually Create an Action Chain

Here, we'll use the Action Chain editor's Design mode to assemble built-in actions into a sequence that performs a task. Each action performs a specific function and returns results that can serve as inputs for subsequent actions.

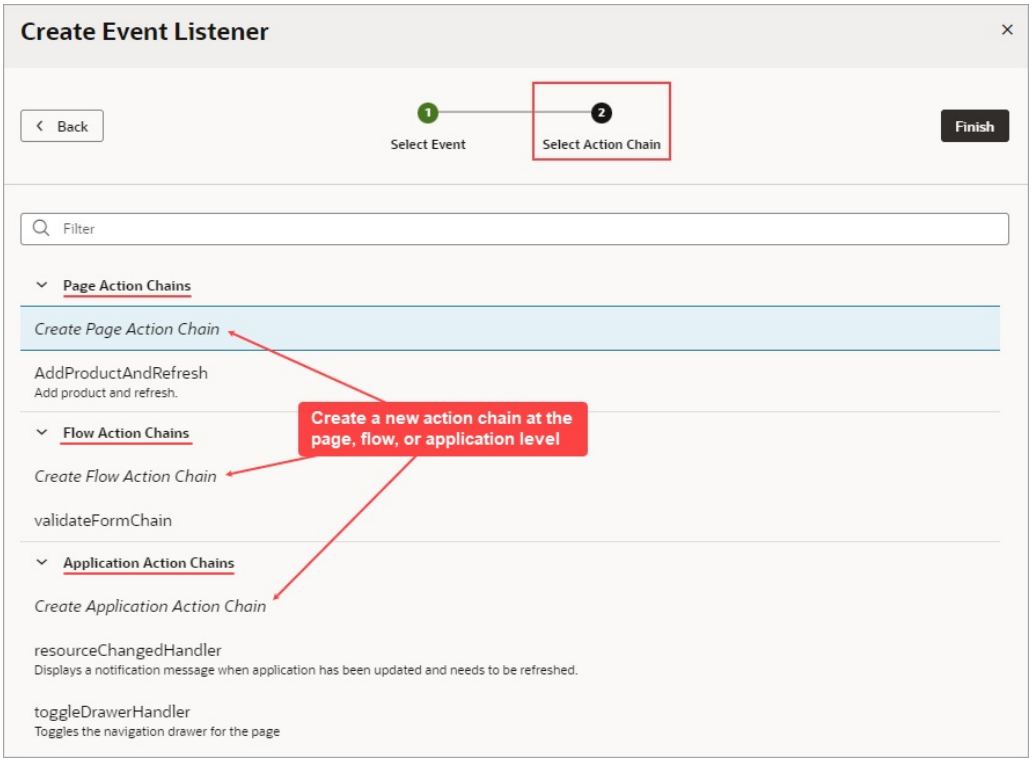
To visually create an action chain:

1. Navigate to where you want to initiate the creation of the action chain, depending on your preference and how you want it triggered:

   - **Actions Chains Tab**:
     If you prefer to go straight to creating an action chain and later assigning it to an event listener, component event, or variable event, go to the relevant **Actions Chains** tab at the application, flow, or page level. On the Actions Chains tab, click the **+ Action Chain** button. If an action chain is displayed on the tab instead, click the **Action Chains** link in the Action Chain editor to get to the list of created action chains and the button for creating a new action chain:



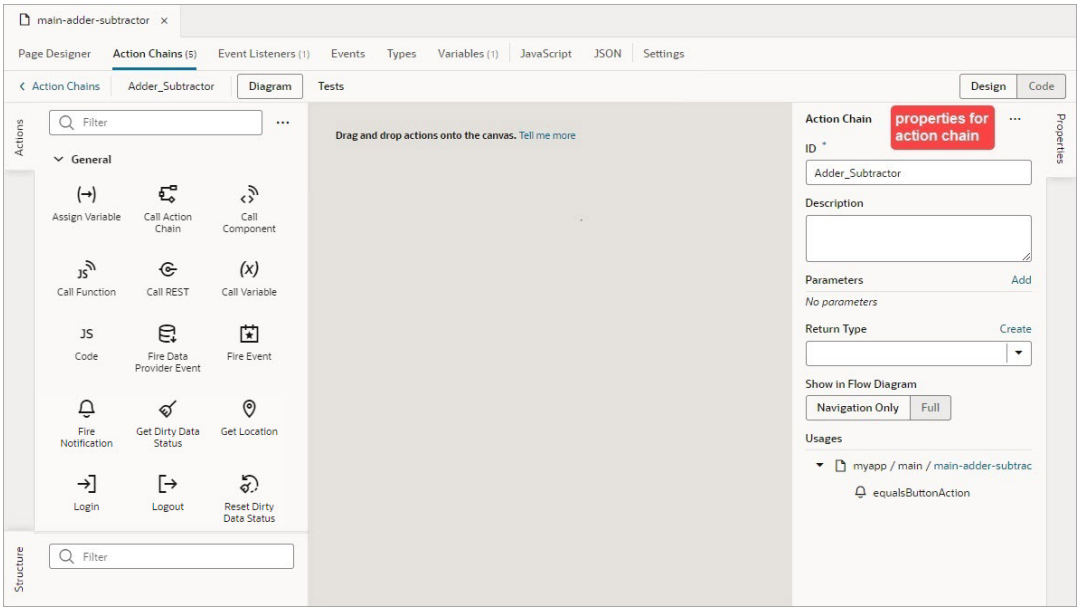
   - **Event Listeners Tab**:
     To have your new action chain started by a lifecycle, application, flow, or page event (`vbBeforeEnter`, `vbEnter`, `vbAfterNavigate`...), select the **Event Listeners** tab and click **+ Event Listener**. In the Create Event Listener wizard, select the event and click **Next**. On the wizard's **Select Action Chain** step, select the create action chain option at the appropriate level (page, flow, or application) and click **Finish**. For further details, see Start an Action Chain From a Lifecycle Event.



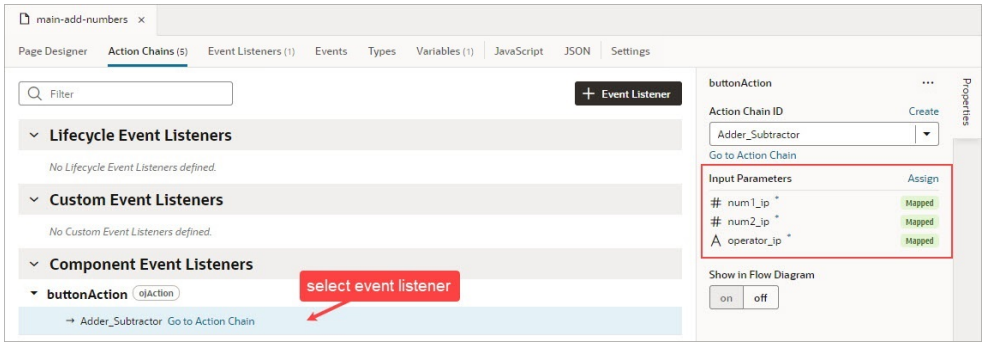
- **Custom Event**:
  To have your new action chain started by a custom event that's triggered by a Fire Event action in another action chain, see Add a Fire Event Action.
- **Component**:
  To have your new action chain started by a component event, select the component on the Page Designer's canvas, and in the Properties pane use the **Events** tab to create a new event, event listener and action chain for the component. For further details, see Start an Action Chain From a Component and Start an Action Chain By Firing a Custom Event.
- **Variable**:
  To have an action chain started when a variable's value changes, open the relevant **Variables** tab, at the application, flow, or page level, and select the variable. In the Properties pane, select the **Events** tab and click **+ Event Listener**. A new `onValueChanged` event is automatically created for the variable, and you're presented with a window for you to either select an existing action chain or create a new one. For further details, see Start an Action Chain When a Variable Changes.

For more about events and event listeners, refer to Work With Events and Event Listeners .

2. Enter a name for the action chain in the **ID** field, and if you like, a description. The new action chain opens in the Action Chain editor:

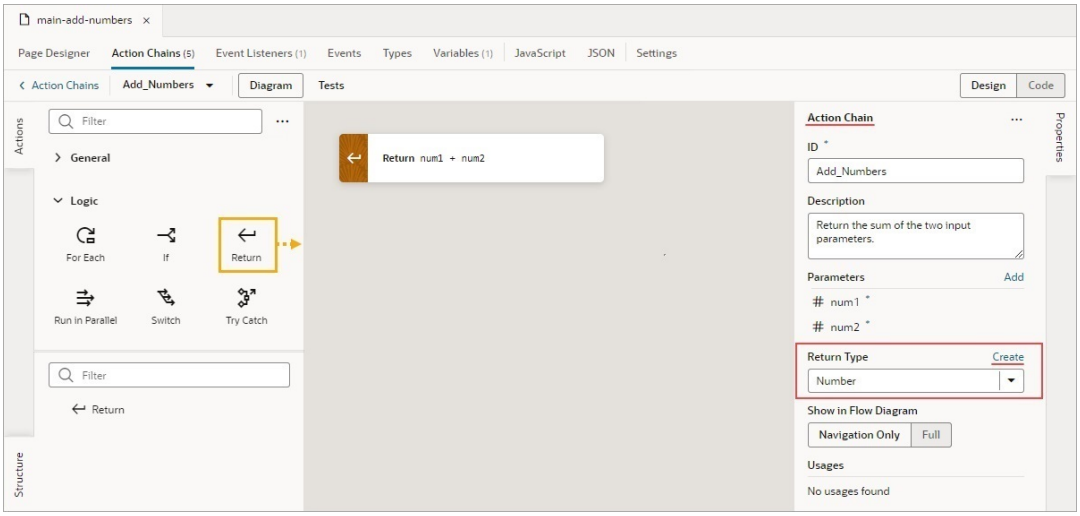


3. If your new action chain needs input parameters:
   a. Define the input parameters using the **Add** link for the Parameters property in the Properties pane.
   b. Pass the input parameter values to the action chain:
      - If your action chain is to be started by another action chain, the input parameters are passed through the call to your new action chain.
      - If your new action chain is to be started by an event listener, open the relevant **Event Listeners** tab (application, flow, or page level), select the event listener, and use the **Assign** link for the **Input Parameters** property in the Properties pane:

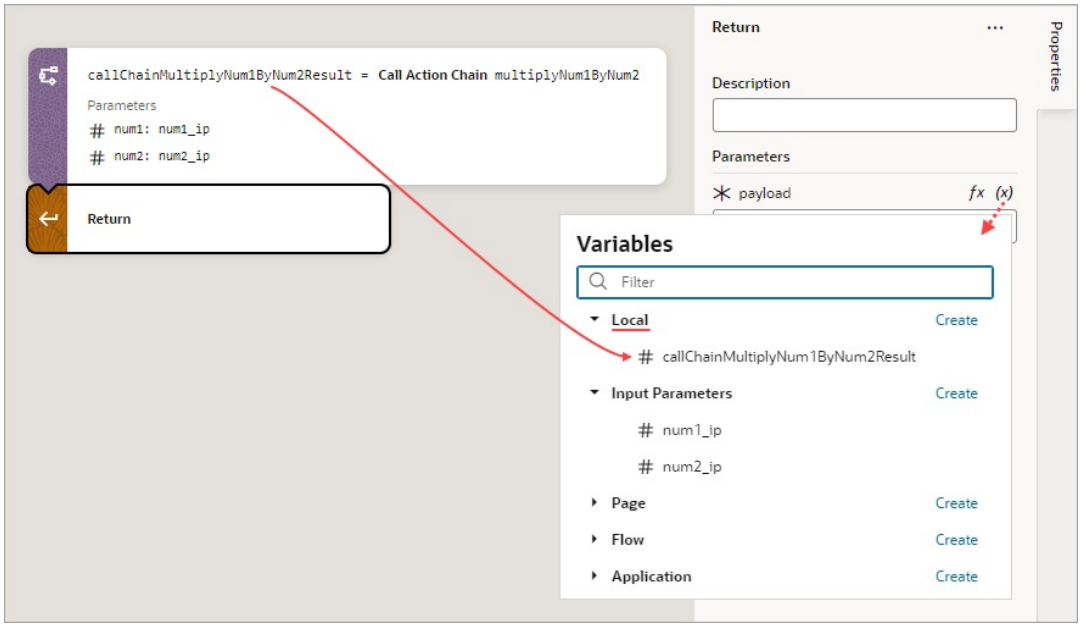


- If your action chain is to be started by a component, select the component on the Page Designer's canvas and in the Properties pane select the **Events** tab. Use the **Assign** link for the **Input Parameters** property.
- If your action chain is to be started by a variable, open the relevant **Variables** tab (application, flow, or page level), and in the Properties pane select the **Events** tab. Use the **Assign** link for the **Input Parameters** property.

4. If your action chain needs to return a payload, click the canvas to bring up the action chain in the Properties pane. For **Return Type**, select the type, or click the **Create** link to create a return type:



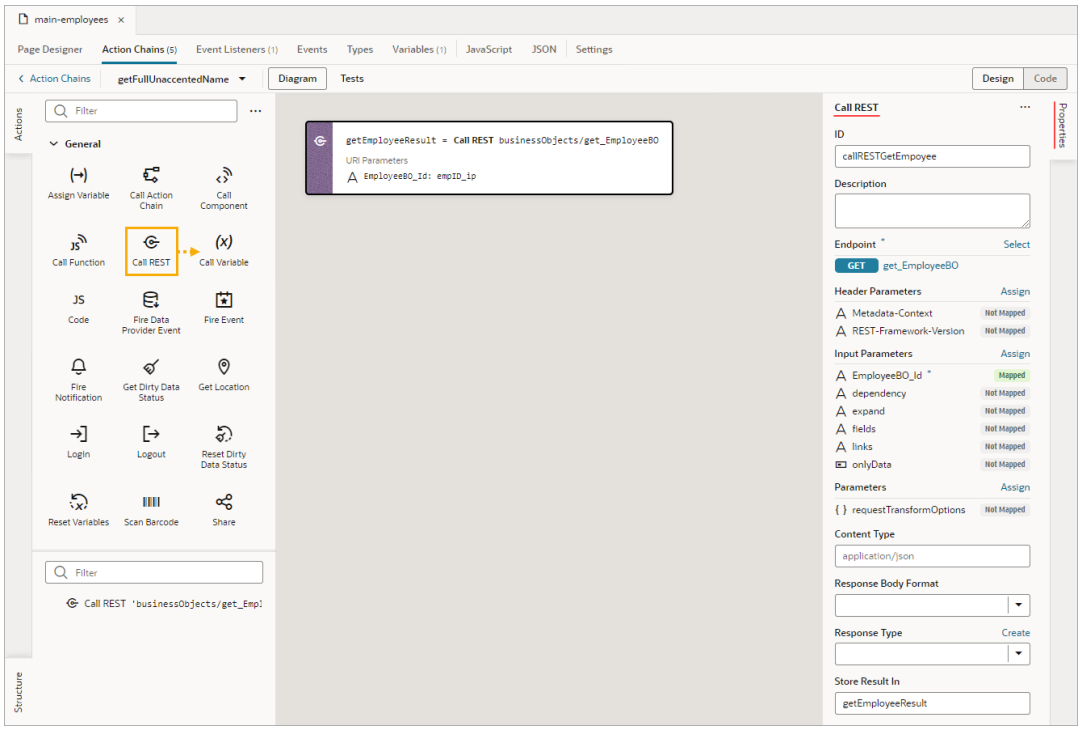
As shown in this example, in which the variable for the Return action to return is selected, the result returned by an action chain is available in the Variables picker, under the **Local** node, :



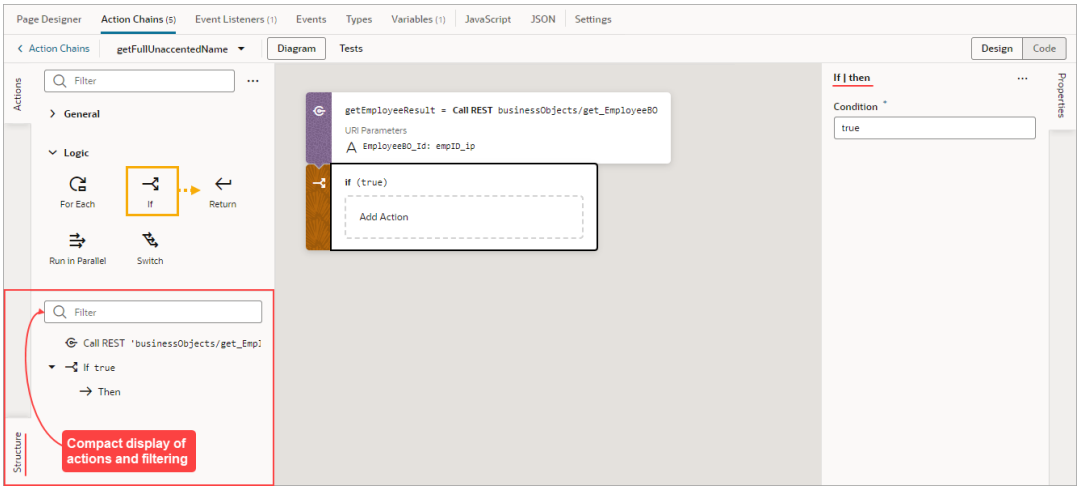
You are now ready to add the actions that will compose the action chain.

5. If an action you need isn't available in the Actions palette, add the **Code** action to add your own block of code, or create a custom action if you think you'll need it in the future. For details about how to create a custom action, see Custom Actions.

6. From the Actions palette, double click an action or drag and drop it onto the canvas. The new action is added to the chain and is selected by default. The Properties pane displays the properties that you can specify for the action, and the action's card on the canvas displays the specified values. For example, here's a Call Rest action with its properties set in the Properties pane:

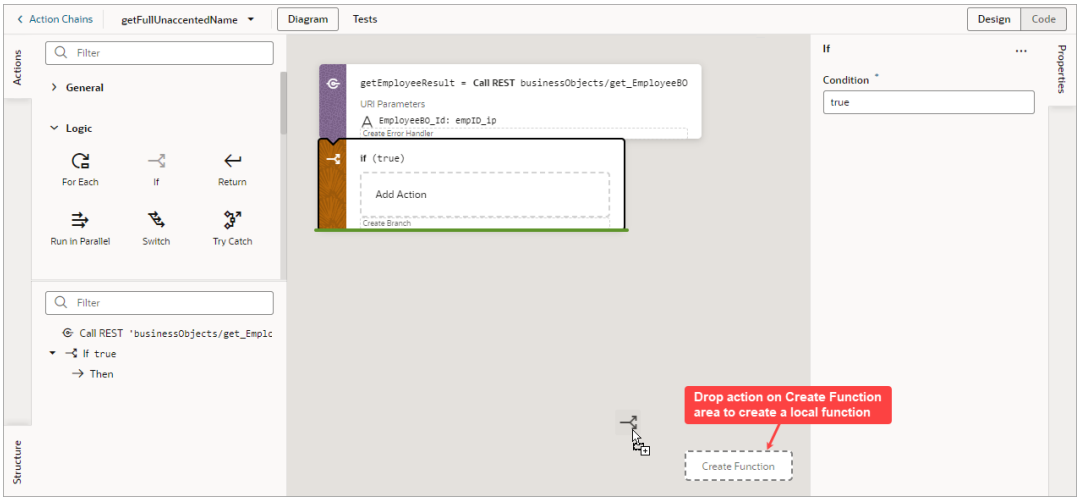


7. Double click or drag and drop the next action from the Actions palette onto the bottom edge of the action that it follows. Configure the action in the Properties pane or through code. To add an action before another action, drop it on the top edge of the action it is to precede.

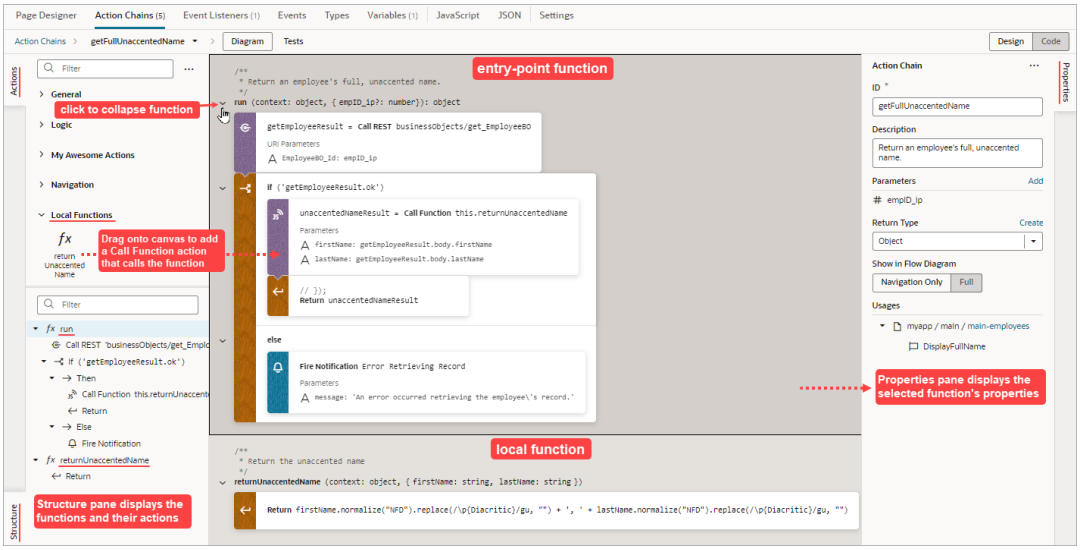


As shown above, the Structure pane displays a compact view of the actions and it provides a filter to quickly find and select an action.

8. If you want to create a local function for your action chain, for the sake of modularity, drop the function's first action on the **Create Function** area that appears when you drag an action over the canvas:

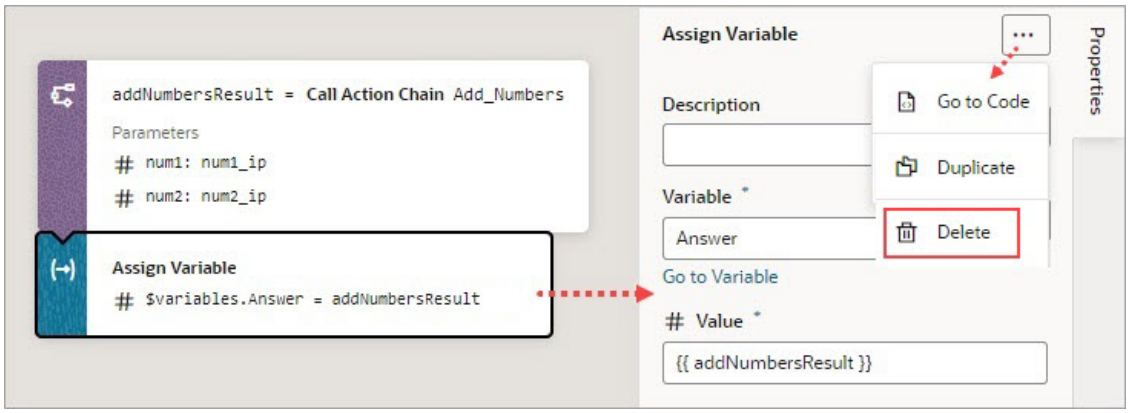


When a local function is created, it gets its own space on the canvas, the Property pane displays its properties, and the Structure pane displays its actions. Also, the local function is added to the Actions palette, under the Local Functions category, for you to quickly add a call to it.

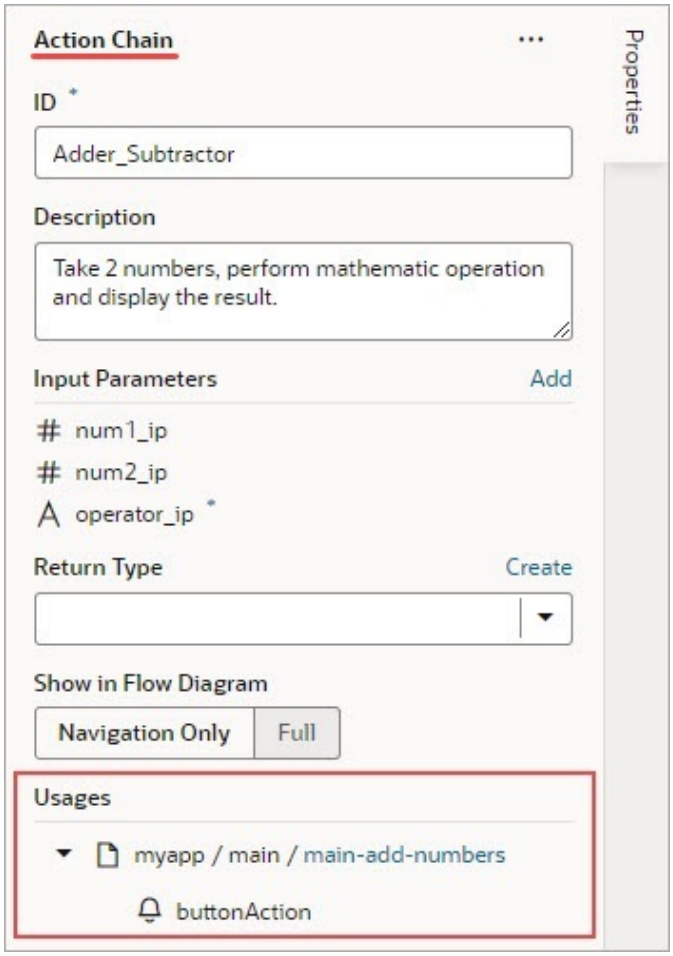


9. Continue to add and configure actions until your action chain is complete. The action chain is saved automatically as you make changes.

If you need to remove an action from the chain, right-click the action on the canvas and select **Delete** or press Delete on your keyboard. You can also delete the action in the Properties pane using its options menu:

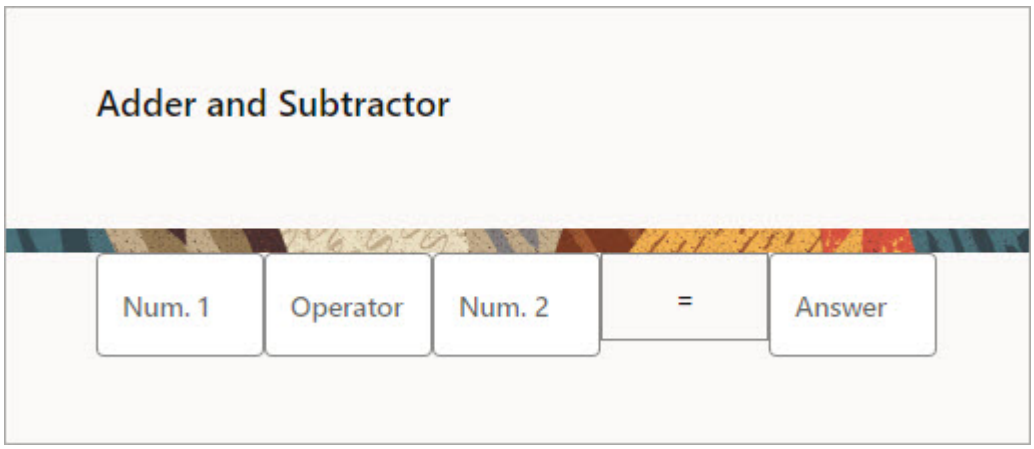


To view usage details for your action chain, such as which pages use it, click an empty space on the canvas to select the action chain and look under Usages in the Properties pane. Click a usage to navigate there. The event listener tied to the event that calls the action chain is also listed, as shown here:



Example of How to Create an Action Chain

In this example, we implement this Adder and Subtractor interface by creating an action chain that either adds or subtracts two numbers and displays the result:

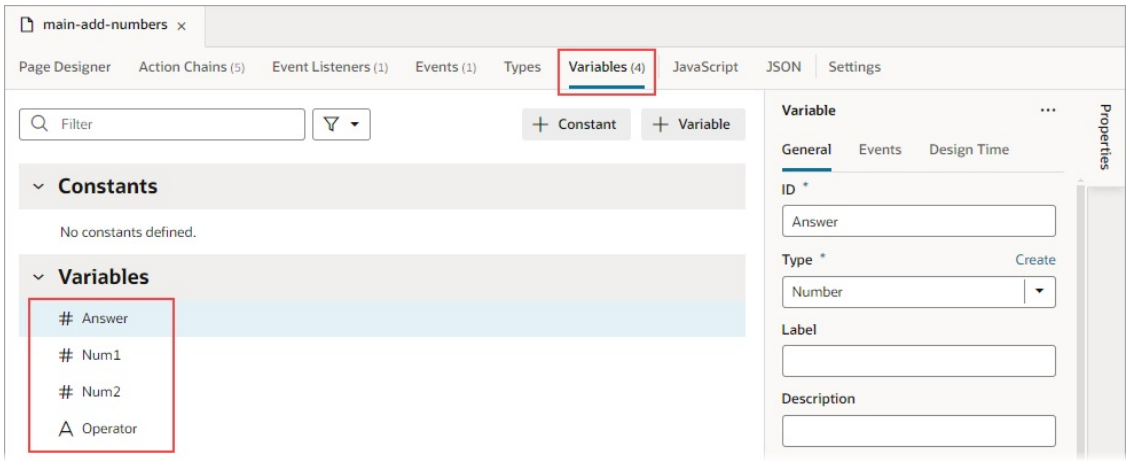


The interface has:

- Four text components:
  - Two for entering the numbers to add or subtract (Num. 1, Num. 2)

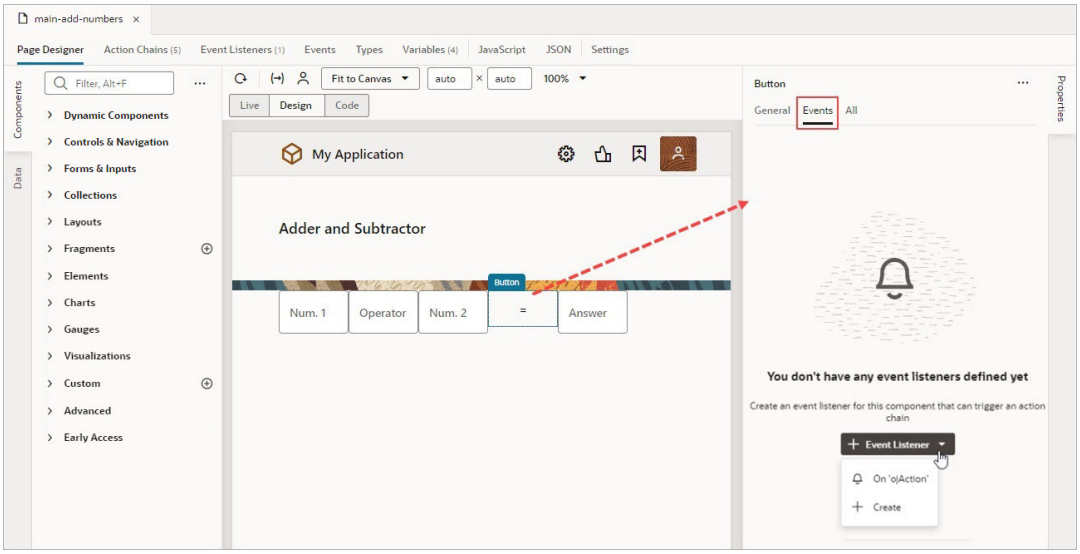
  - One for entering either a plus or minus sign (Operator)
  - One to display the result (Answer)
- One button (=) that triggers the action chain that performs the operation and displays the result

Each of the four text components is bound to a page variable:



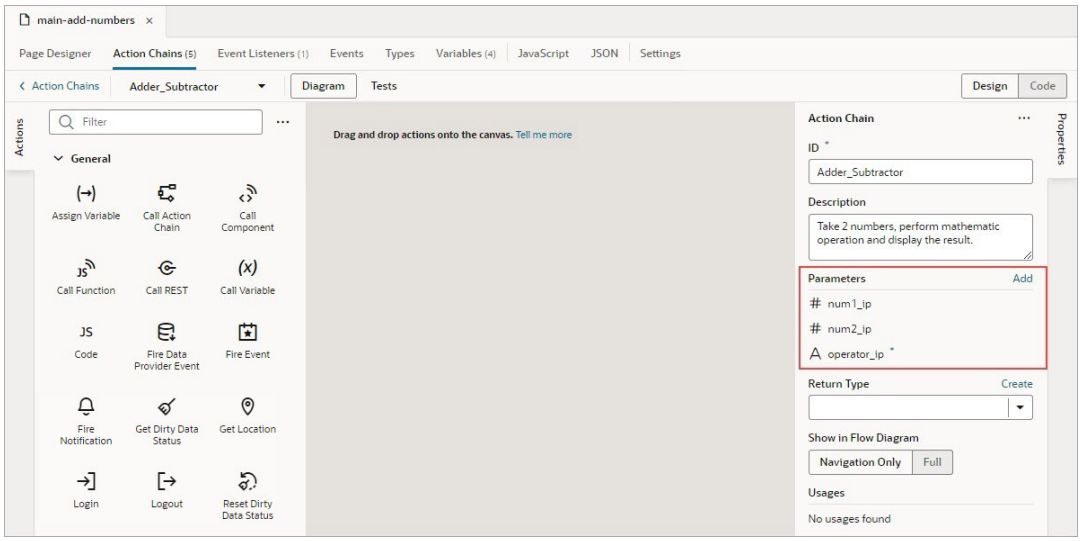
Here, we create the action chain that either adds or subtracts two numbers and displays the result:

1. We want the action chain to be triggered by clicking the equals button, so select the equals button on the Page Designer's canvas, then select the **Events** tab on the Properties pane:



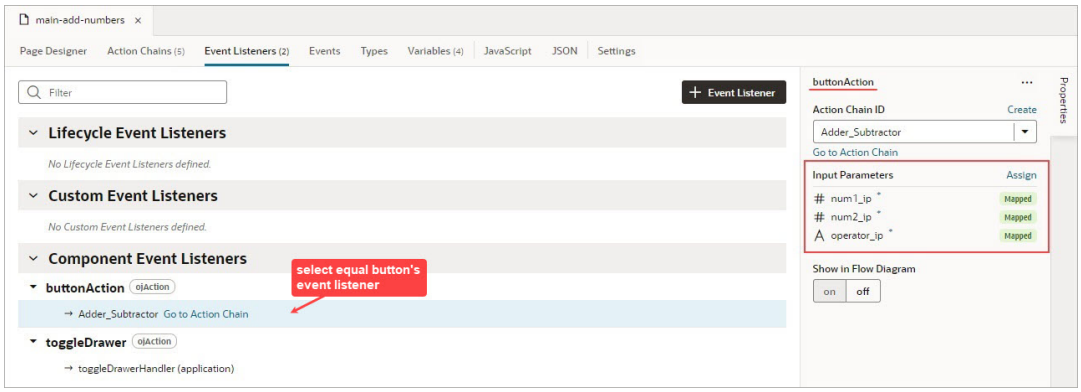
2. Click **+ Event Listener**, then **On 'ojAction'** to create an event that's triggered by clicking the button, as shown above.
   The new action chain opens in the Action Chain editor.
3. Using the Properties pane, enter `Adder_Subtractor` for the action chain's **ID** field, and optionally a description.

4. Since three input parameters are needed, two for the numbers and one for the operator, you need to add them using the **Add** link next to the Parameters property in the Properties pane:

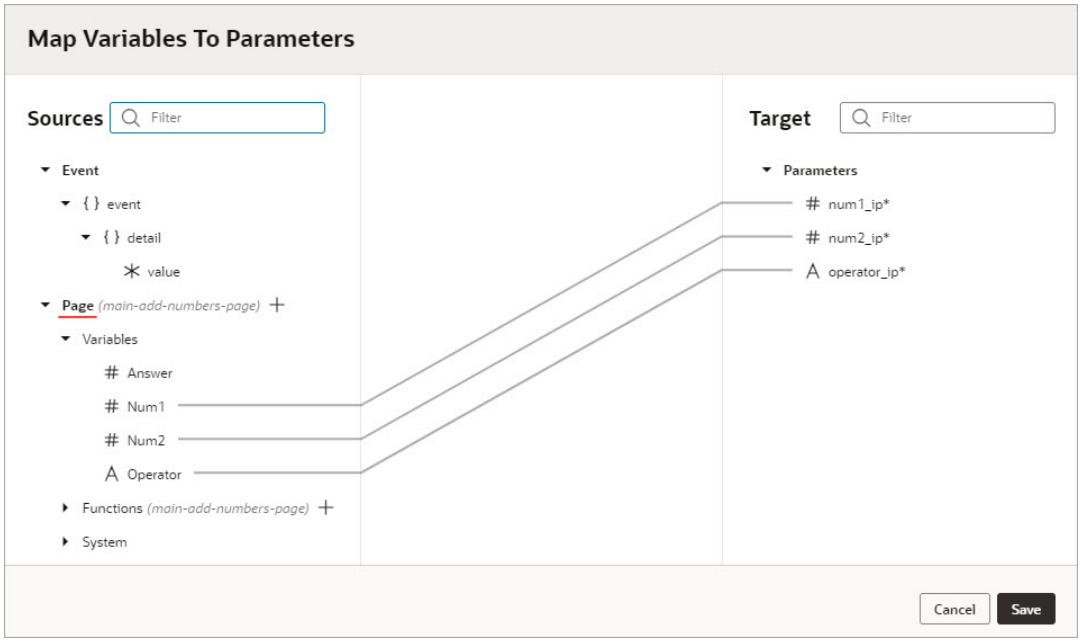


Nothing needs to be returned by this action chain, so we don't need to define a return type.

5. Next, you need to provide the values for your input parameters. Open the **Event Listeners** tab and select the equal button's event listener. In the Properties pane, click the **Assign** link for the **Input Parameters** property:

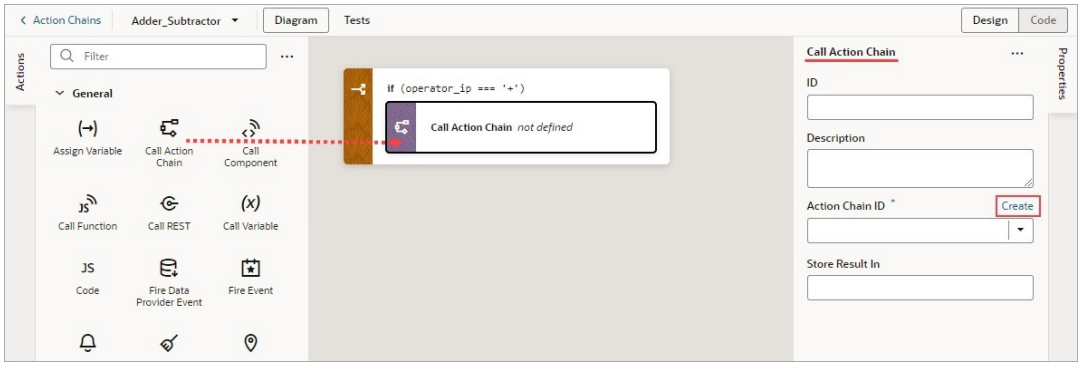


6. In the mapper, map the page variables that were bound to the text components for the numbers and the operator to the action chain's input parameters:



To implement this action chain, we need to handle each possible entry for the operator field: a plus, a minus and an invalid entry.

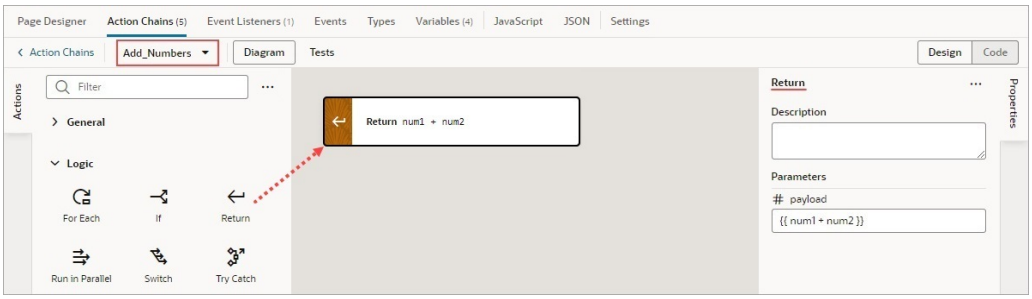
7. Start by dragging and dropping the **If** action from the Action Palette onto the canvas.
8. Select the `If` condition on the canvas, and in the Properties pane enter `operator_ip === '+'` in the **Condition** field to check if the user entered a plus sign.
   To handle this case, let's add a call to a simple action chain, which we'll create, that returns the sum of two numbers.
9. Drag and drop the **Call Action Chain** action from the Actions palette onto the `If` condition.



10. To create the action chain that adds two numbers, in the Properties pane, click the **Create** link for the Action Chain ID property (shown above). In the dialog box, enter `Add_Numbers` in the **ID** field, and optionally, enter a description. Click the **Create** button.
    The action chain has been created and set for the Action Chain ID property.
11. We now need to implement the Add_Numbers action chain by clicking the Action Chain ID property's **Go to Action Chain** link.
    The Actions Chain editor is now loaded with the Add_Numbers action chain.
12. To implement the Add_Numbers action chain:
    a. In the Properties pane, click the **Add** link for the Parameters property and add two input parameters for the numbers to add: `num1` and `num2`.

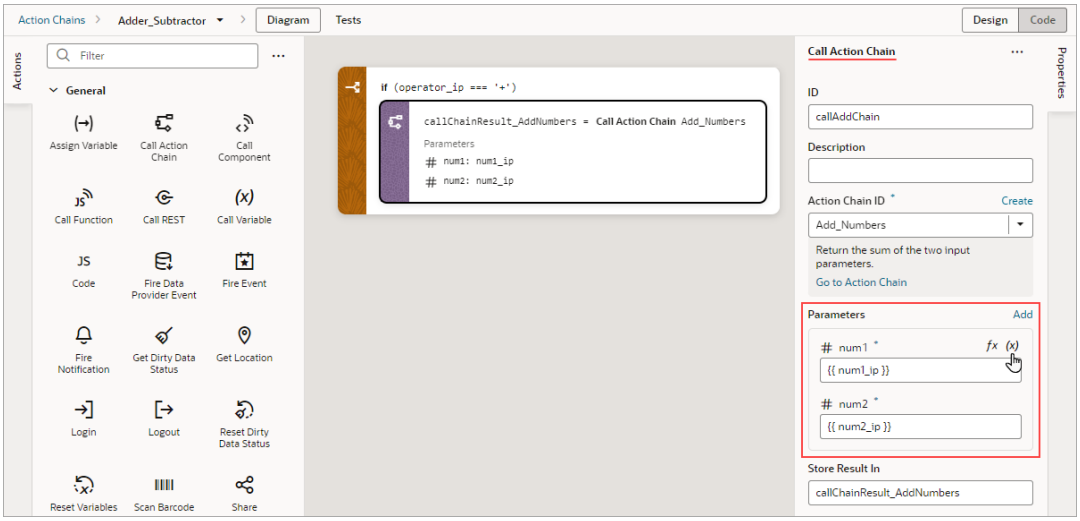
b. Since the action chain needs to return a number, define its return type by selecting **Number** for **Return Type**.

c. Drag and drop the **Return** action from the Actions palette onto the canvas. In the Properties pane, enter `{{ num1 + num2 }}` for the **Payload** property to return the sum of the two input parameters. Recall, wrapping the expression with double curly brackets indicates that it's a literal expression and not a string:



13. Navigate back to the Adder_Subtractor action chain by clicking the **Action Chains** link at the top-left of the editor and selecting the action chain.
The editor is now loaded with the Adder_Subtractor action chain.

14. The two numbers that were passed as input parameters to the Adder_Subtractor action chain now need to be passed to the Add_Numbers action chain. Select the Call Action Chain action on the canvas, and in the Properties pane, for the **Parameters** property, select the number input parameters:

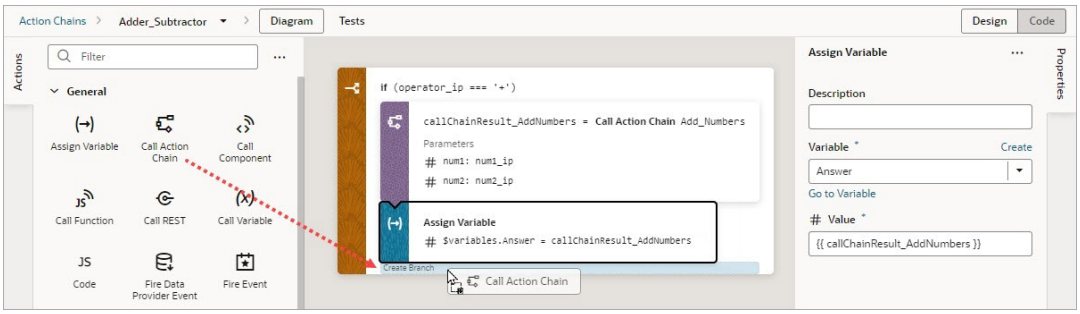


You now need to assign the result from the Add_Numbers action chain to the page variable that's bound to the text component that displays the answer.

15. Drag and drop the **Assign Variables** action to the bottom edge of the **Call Action Chain** action on the canvas. For the **Variable** property, in the Properties pane, select the page variable that's bound to the text component displaying the answer, and for the **Value** property, select the result from the Add_Numbers action chain:

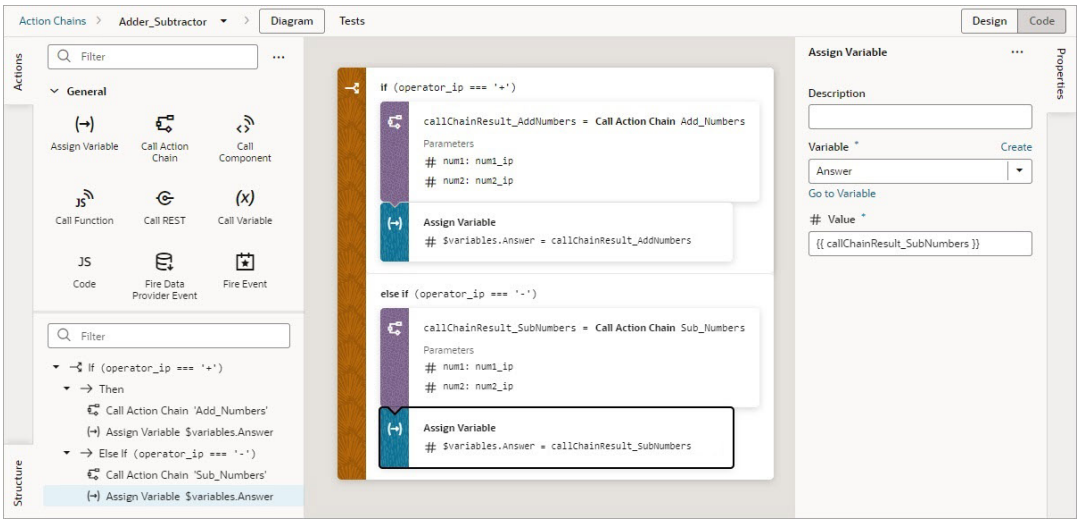
Next, we need an `Else If` condition to handle the case when a user enters a minus sign for the operator.

16. Drag and drop the **Call Action Chain** action onto the **Create branch** area at the bottom of the `If` condition:



17. Change the `Else` condition into an `Else If` by entering `operator_ip === '-'` for the **Condition** field, in the Properties pane.

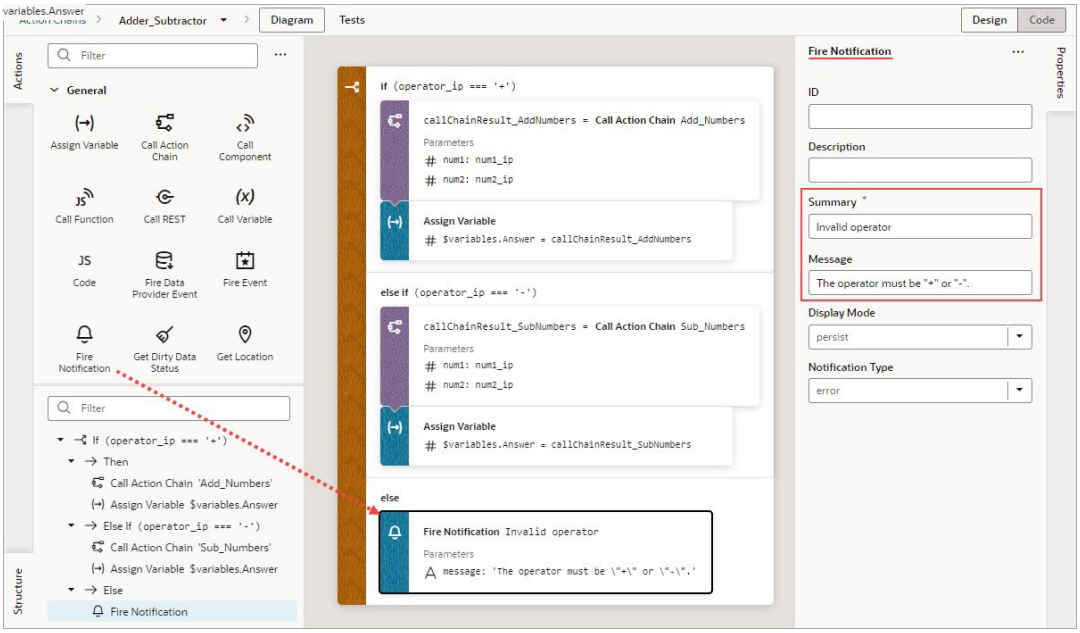
18. Complete this `Else If` condition by following the previous instructions on how to handle the plus sign case.
Here's the completed `Else If` condition:



19. Lastly, we need to handle the case when a user doesn't enter a plus or a minus sign for the operator. Drag and drop the **Fire Notification** action from the Actions palette onto the `Else`

condition. In the Properties pane, enter `Invalid operator` for the **Summary** property, and for the **Message** property, enter:

```
The operator must be "+" or "-".
```



Our action chain is now complete.

20. At this point it makes sense to create some unit tests to test your new action chain. For details on how to do so, refer to Test Action Chains.

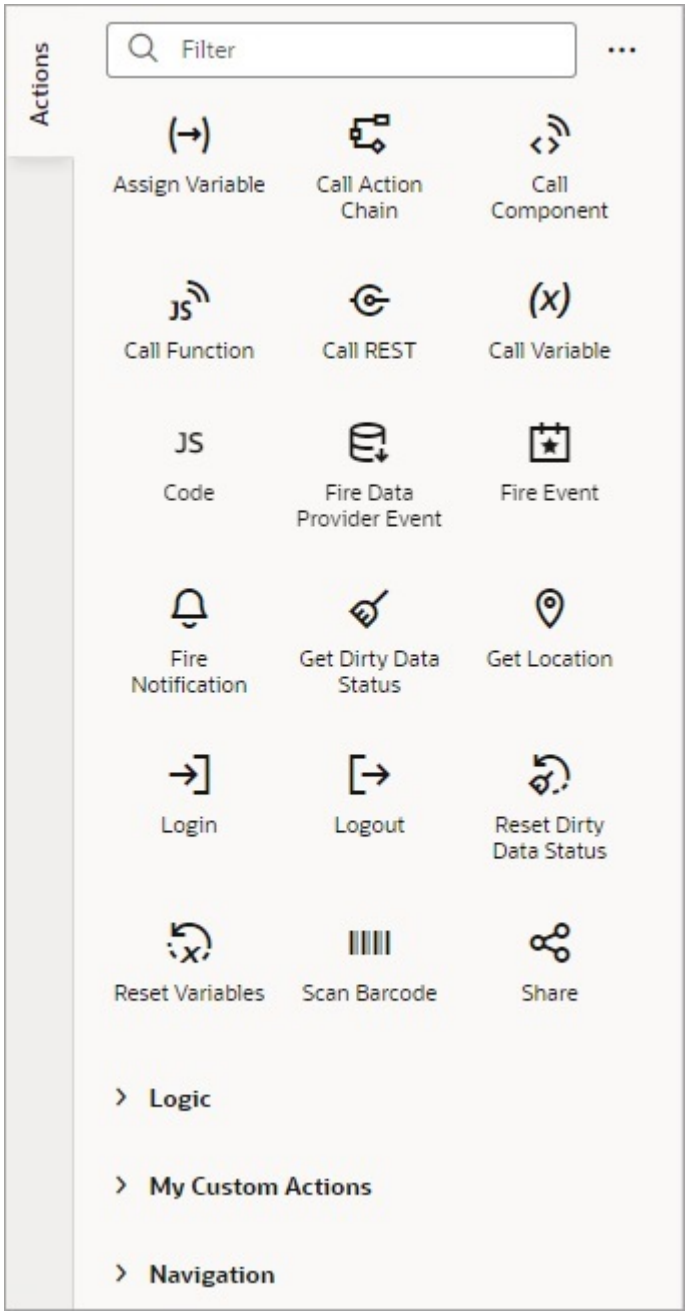
Here's the completed code for the action chain:

```
define([
  'vb/action/actionChain',
  'vb/action/actions'
], (
  ActionChain,
  Actions
) => {
  'use strict';

  class Adder_Subtractor extends ActionChain {

    /**
     * Take 2 numbers, perform mathematic operation and display the result.
     * @param {Object} context
     * @param {Object} params
     * @param {number} params.num1_ip
     * @param {number} params.num2_ip
     * @param {string} params.operator_ip
     */
    async run(context, { num1_ip = '0', num2_ip = '0', operator_ip }) {
      const { $page, $flow, $application, $functions } = context;
```

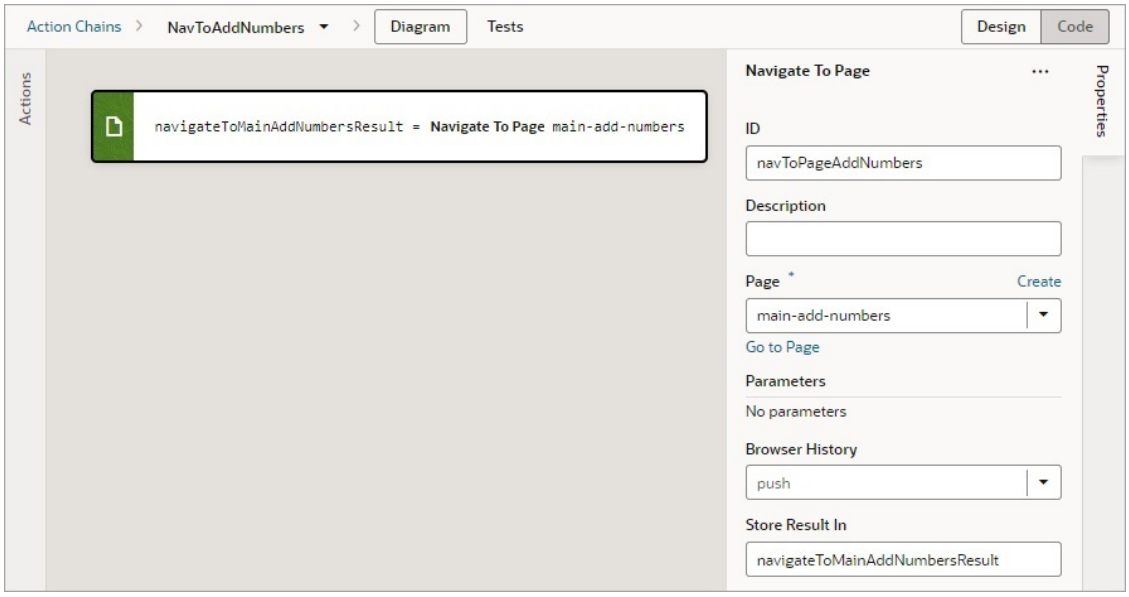
```
    if (operator_ip === '+') {
      const callChainResult_AddNumbers = await Actions.callChain(context, {
        chain: 'Add_Numbers',
        params: {
          num1: num1_ip,
          num2: num2_ip,
        },
      }, { id: 'callAddChain' });

      $variables.Answer = callChainResult_AddNumbers;

    } else if (operator_ip === '-') {

      const callChainResult_SubNumbers = await Actions.callChain(context, {
        chain: 'Sub_Numbers',
        params: {
          num1: num1_ip,
          num2: num2_ip,
        },
      });

      $variables.Answer = callChainResult_SubNumbers;
    }
    else {
      await Actions.fireNotificationEvent(context, {
        summary: 'Invalid operator',
        message: 'The operator must be "+" "-".',
      });
    }
  }
}

  return Adder_Subtractor;
});
```

Built-In Actions

Visual Builder provides a set of built-in actions that you use to create your action chain. If an action you need isn't available in the Actions palette, use the **Code** action to add your own block of code, or if a future need warrants it, create a custom action that can be reused.

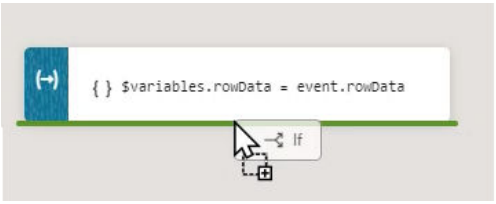


Each action performs a specific function and requires you to set different properties. For example, when you add the Call REST action to your action chain, you need to specify the endpoint and other details about the response to the Call REST action. Similarly, when you add a Navigate To Page action, you'll need to select a page in your current application to navigate to:



You can add an action in one of three ways, depending on your preference and where you want it added:

- Drag and drop the action from the Actions palette onto the bottom edge of the action it's to follow, or onto the top edge of the action it's to precede.



- Double-click the action in the Actions palette to add it to an empty canvas or to the end of an action chain.
- Select the action on the canvas that you want the new action to follow, then double-click the new action in the Actions palette.

If you need more details about an action than are provided in this section, refer to the JavaScript Actions section in the *Oracle Visual Builder Page Model Reference*.

Add an Array Operation Action

Includes

You use an Includes action to check if an array includes a specific value, returning true or false.

After you add the action, select the array using the Variable property. For the Search Element parameter that appears, specify the value to search for, and for the From Index parameter, select the index to start the search from.

Pop

You use a Pop action to remove and return the last element from an array, which decreases the array's length by one.

After you add the action, select the array using the Variable property. The last element is returned by the variable shown by the Store Result In property.

Push

You use the Push action to add an element to the end of an array. This action also returns the new length of the array.

After you add the action, select the array using the Variable property. The new length of the array is stored in the variable shown by the Store Result In property.

Slice

You use the Slice action to return a copy of an array, by specifying the indexes to start and end the copying. The array is copied from the start to the end indexes you provide.

After you add the action, use the Variable property to select the array for the operation. For the Start Index and End Index parameters that appear, enter the indexes to start and end the copying. The copy of the array is returned in the variable shown by the Store Result In property.

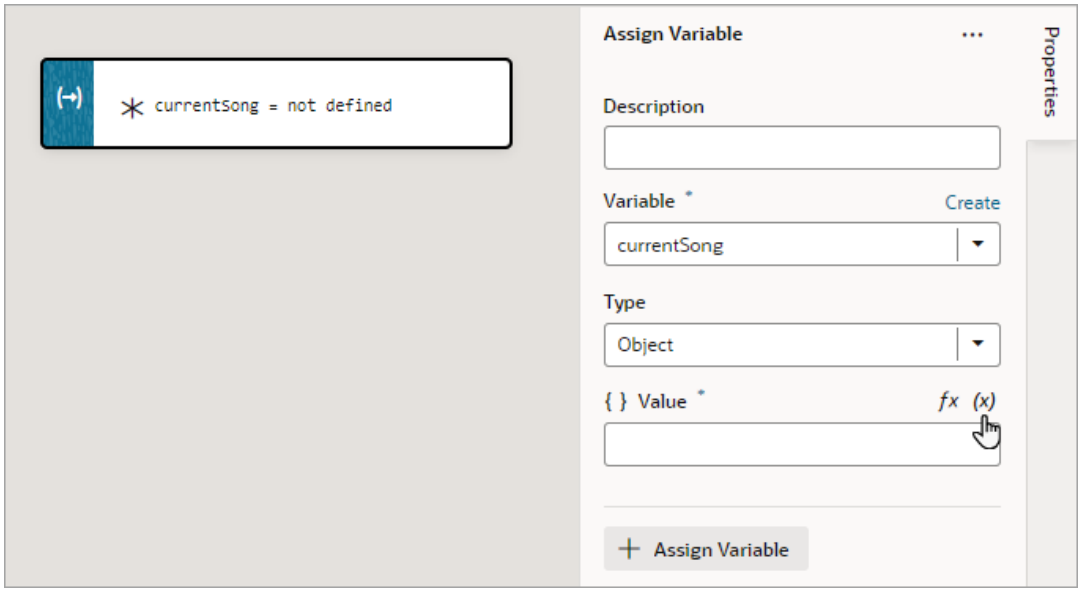
Add an Assign Variable Action

You use an Assign Variable action to assign a local, page, flow, or application variable a value. This action can also be used to create a local variable.

For example, if your action chain sends a request to a GET endpoint, you can use the Assign Variable action to map the response to a page variable that's bound to a page component. Or, suppose 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.

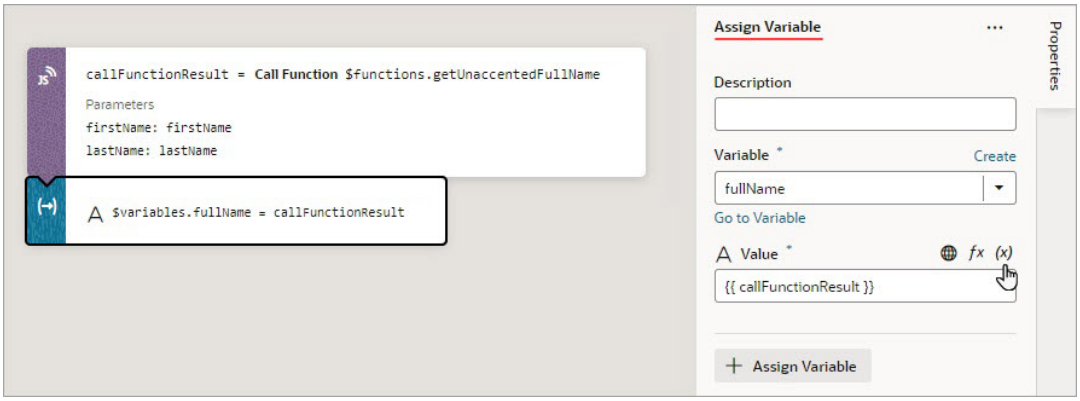
To use an Assign Variable action to create a local variable:

1. For **Variable**, enter its name and hit Enter on your keyboard.
2. For **Type**, select its data type.
3. If necessary, use the **Value** field to assign it a value.

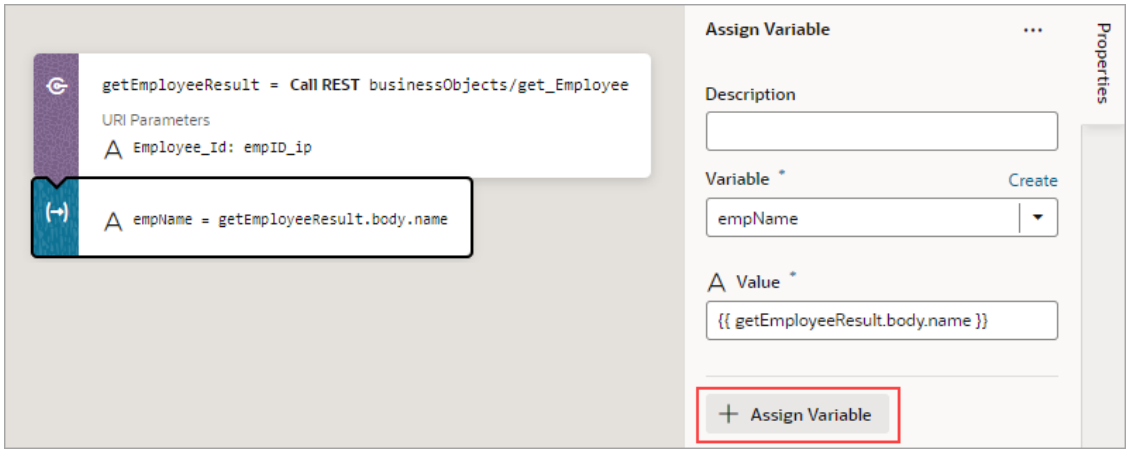


To use an Assign Variable action for a value assignment:

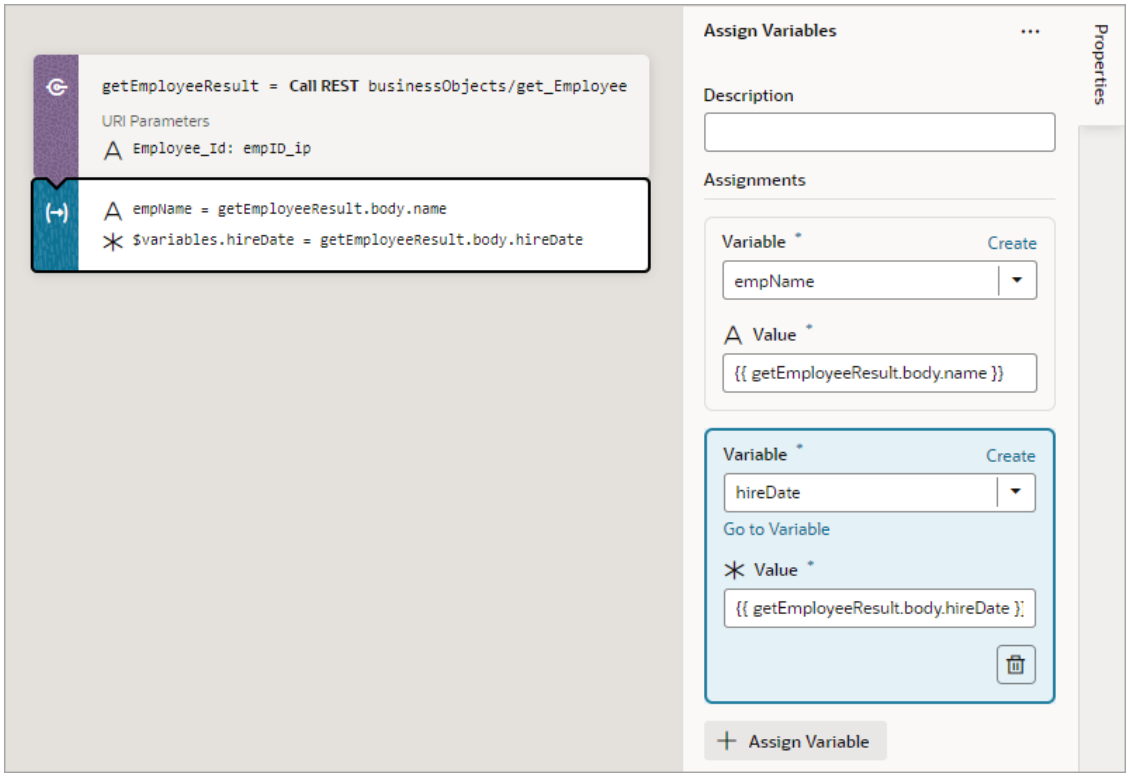
1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. If you need to create a variable for the assignment, click the **Variable** property's **Create** link, otherwise, start to type the variable's name in the field and select it when it appears. You could also select the variable from the list.
3. To set the variable's value, hover over the far-right side of the **Value** property and click to choose the variable that holds the value, or click ***fx*** to create an expression for the value.
   In this example, the page variable `fullName` is assigned the result from a module function:



If you need to do another assignment, click the **+ Assign Variable** button in the Properties pane:



Then make the assignment using the Variable and Value fields that appear for the new variable assignment:

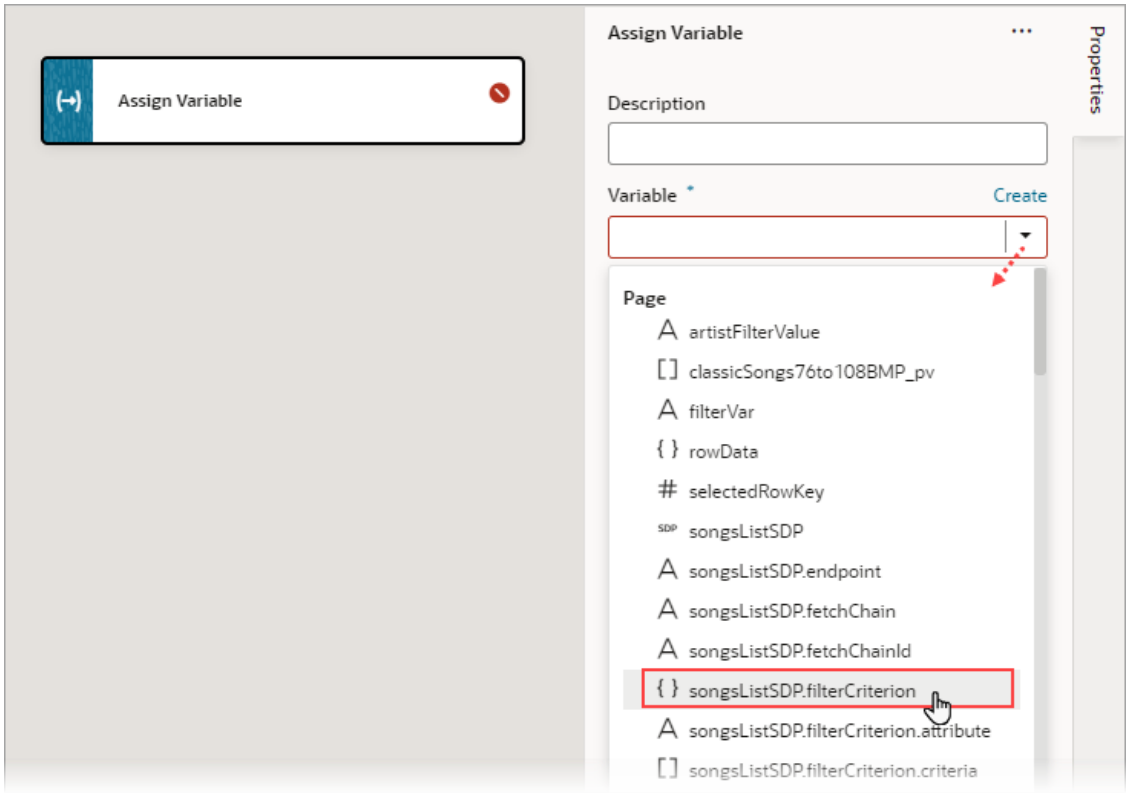


If you'd like to move a variable assignment to a different position, in the Properties pane, click and hold an assignment, then move it to its new position.

Use Filter Builder to Create Filter Criteria for an SDP

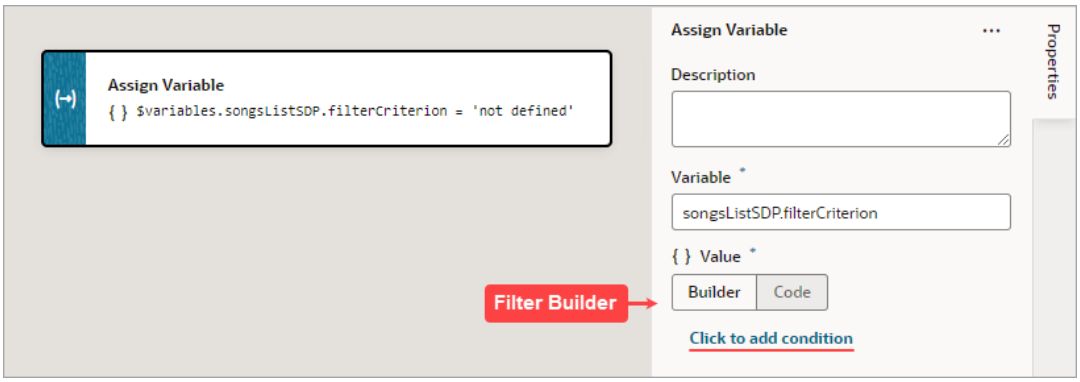
If you're using an SDP to provide a table or list's data, and you'd like to filter out rows, you can use the Assign Variable action to create and assign the filter criteria to the SDP's `filterCriterion` property. For further details about using an SDP to filter a table or list's rows, see Filter Data by Filter Criteria.

When the Assign Variable action's Variable property is set to an SDP's `filterCriterion` property, the Filter Builder appears under the Variable property for you to create the filter criterion. To directly work with the code, click the **Code** button. For details, see Filter Builder's Code Editor.

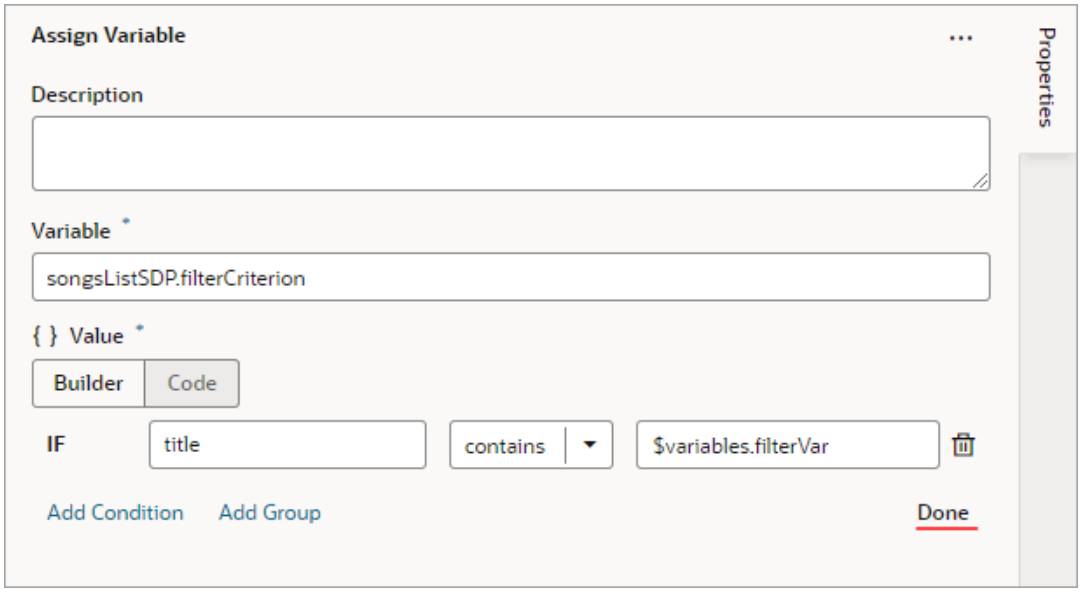


To use the Assign Variable action's Filter Builder to create the filter criterion for an SDP:

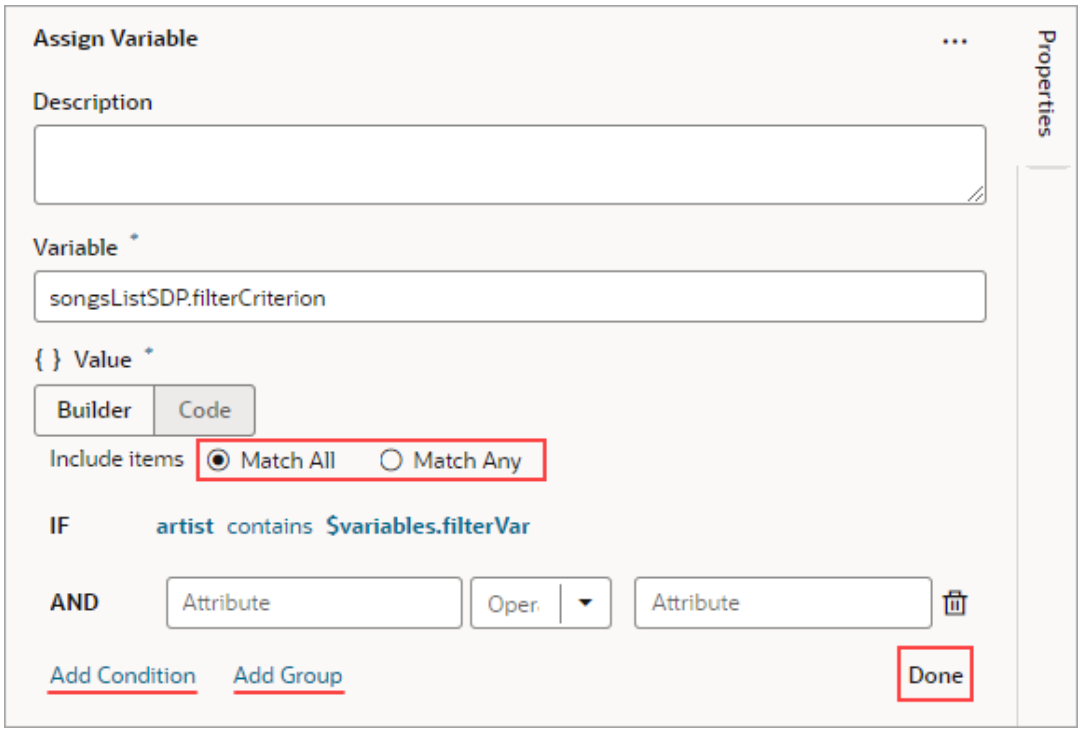
1. Click the Filter Builder's **Click to add condition** link:



2. For the first Attribute textbox, enter the name of the column (field in record, like "city") that you want compared against a specific value (like "Tokyo").
3. For the Operator list, select the operator for the criterion.
4. For the second Attribute textbox, enter the specific value to compare against, or select the variable that contains the value. For instance, the value could be stored by a page variable that was bound to an Input Text component for a user to enter the value.



5. To add another condition, click the **Add Condition** link to add one with an AND or OR operator, or click the **Add Group** link to add a group of conditions that are to be evaluated together (conditions enclosed in brackets). To combine conditional expressions with the AND operator, select **Match All**, and to combine them with the OR operator, select **Match Any**:



6. Click **Done** when you're finished.

Filter Builder's Code Editor

You can use the Filter Builder's Code tab to view and edit the filter's code. After defining a condition on the Builder tab, you will see that the Code tab contains an `attribute`, `op` and `value` property.

Here's an example of a filter with two conditions combined by an AND operator:

```
{
 "op": "$and",
 "criteria": [
  {
   "op": "$eq",
   "attribute": "name",
   "value": "{{ $variables.filterVar }}"
  },
  {
   "op": "$eq",
   "attribute": "id",
   "value": "{{ $variables.idVar }}"
  }
 ]
}
```

In this example:

- The Oracle JET operator is "`$eq`" (it must include the dollar sign ("`$`")).
- The `attribute` property is set to the name of the field (column) that you want to be evaluated against the `value` property.
- The `value` property (`$variables.customerListSDP.filterCriterion.criteria[0].value`) is mapped to a page variable (`$variables.filterVar`) that holds the value to be evaluated against each field (column) value.

Add a Call Action Chain Action

You add a Call Action Chain action to start an action chain. This action can call action chains defined in the same page, flow, or application.

> **Note:**
>
> Using this JavaScript action, you can call a JSON action chain, however, you can't call a JavaScript action chain from a JSON action chain.

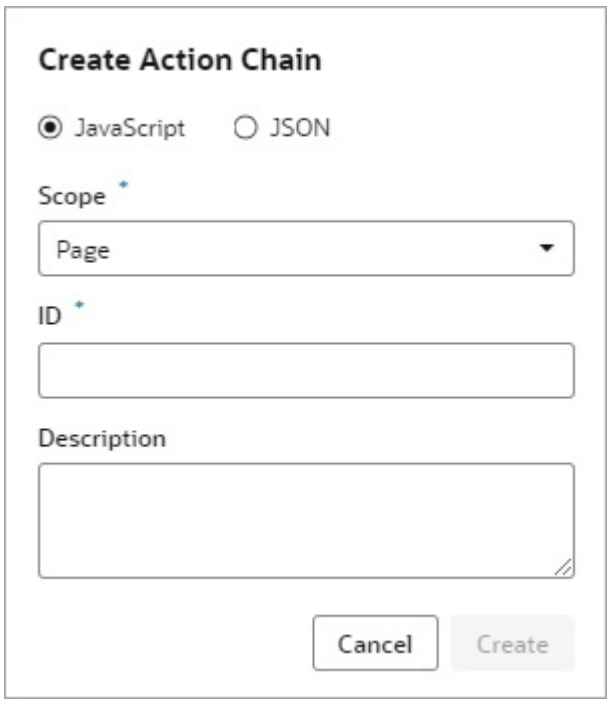
For API information about this action, see Call Action Chain in the *Oracle Visual Builder Page Model Reference*.

To use a Call Action Chain action:

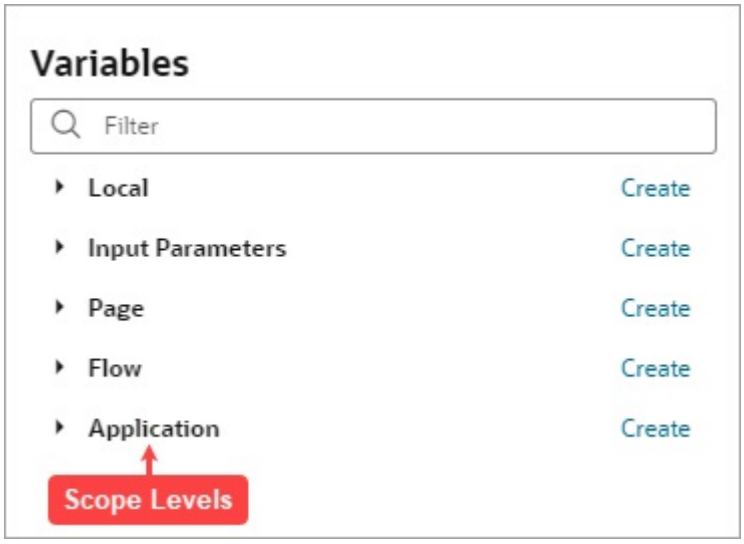
1. Add the action in one of three ways, as explained at the end of Built-In Actions.

2. Select an existing action chain from the **Action Chain ID** list, or click **Create** to create a new action chain.

   The dialog lets you choose between a new JavaScript or JSON action chain, and has a list for you to choose the action chain's scope (page, flow, or application). Depending on where you are creating the action chain, the list might have entries for action chains defined in the page, in the current flow, or in the application. If you're creating an action chain in a flow, you can only select other action chains defined in the same flow or in the current application, and you won't see an entry for page level action chains.

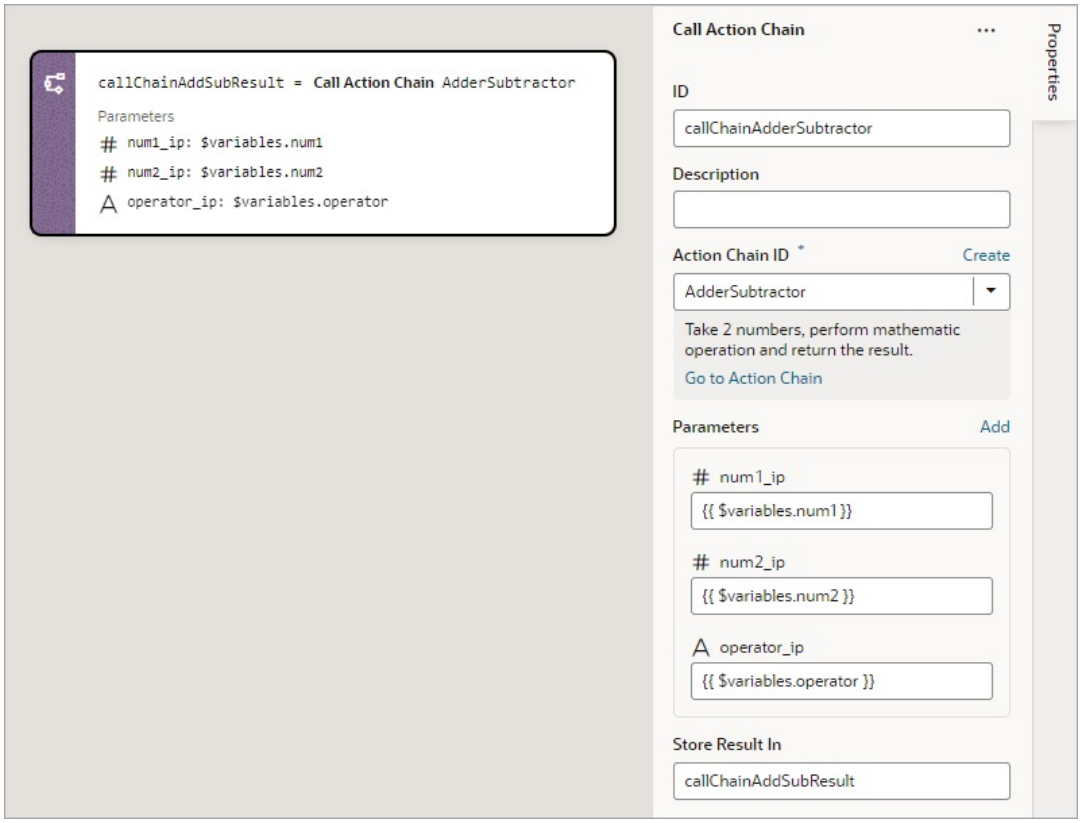


3. If the called action chain requires input parameters, the input parameters will be listed under the **Parameters** section of the Properties pane. For each input parameter, hover over the far-right side of the parameter and click to choose its source. If you need to create a variable, use the appropriate **Create** link in the Variables dialogue to create it at the appropriate scope level.



If a value is returned by the action, it is assigned to the auto-generated variable shown by the Store Result In property.

Here's an example of a completed Call Action Chain action with specified input parameters:



Add a Call Component Action

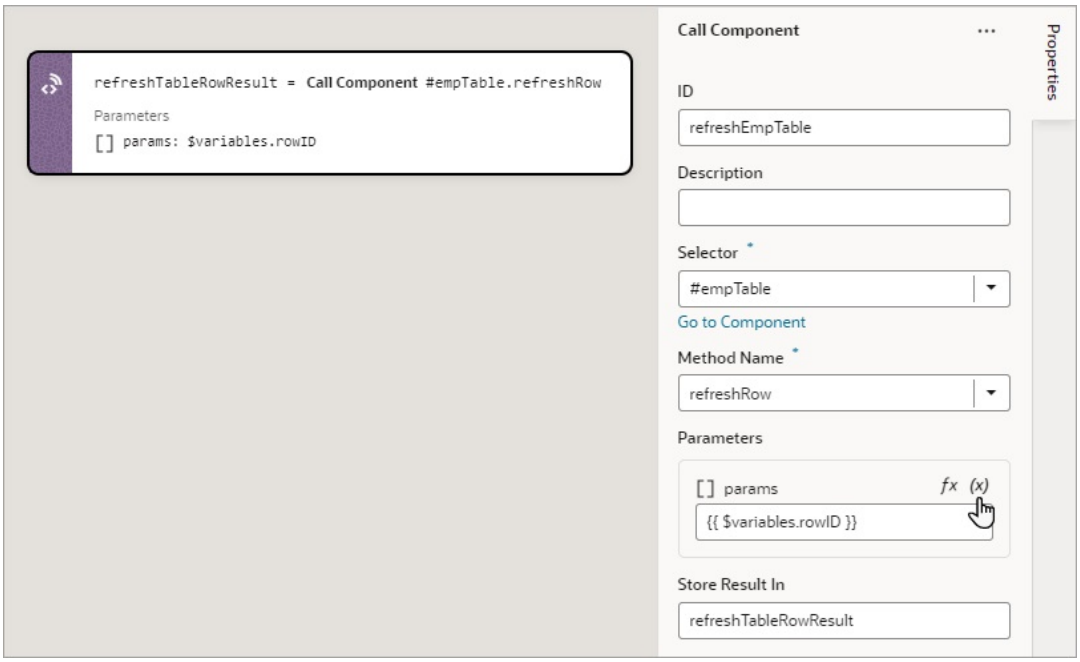
You add a Call Component action to call a method on a component.

For API information about this action, see Call Component Method in the *Oracle Visual Builder Page Model Reference*.

To use a Call Component action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. In the Properties pane, select the component name from the **Selector** list, which is only populated with components that have their ID properties specified.

   For example, if your page contains three buttons whose IDs are `Create`, `Update`, and `Save`, you'll see those options available for selection in the list.
3. With the component selected, select or enter the **Method Name** to call:



4. If the method requires input parameters, hover over the far-right side of the parameters under the Parameters section and click to choose their source.

If a value is returned by the action, it is assigned to the auto-generated variable shown by the Store Result In property.

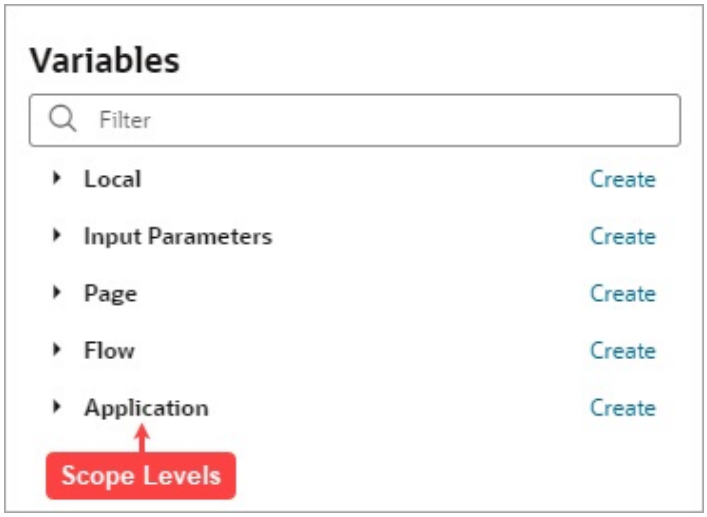
Add a Call Function Action

You add a Call Function action to call a function defined for the current page, flow, or application. Functions for a page, flow and application are created using their JavaScript editor.

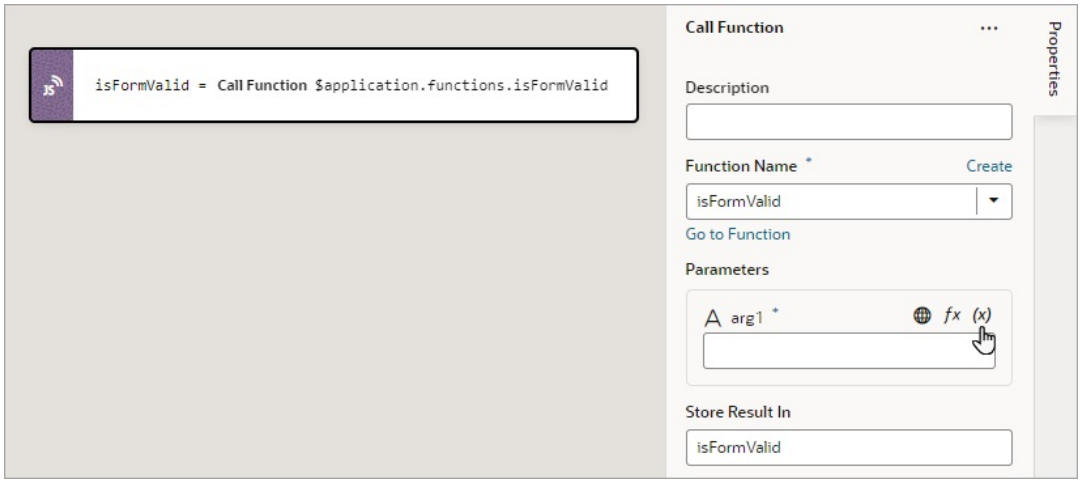
For API information about this action, see Call Function in the *Oracle Visual Builder Page Model Reference*.

To use a Call Function action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. In the Properties pane, select an existing function from the **Function Name** list, or click **Create** to create a new function. You can select or create a function that is defined for the current page, flow, or application.
3. If you want to view or modify the function's code, click **Go to Module Function** to go to the JavaScript editor to do so.
4. If the function requires input parameters, they'll be listed under the Parameters section of the Properties pane. For each input parameter, hover over the far-right side of the parameter and click to choose the variable that holds the value, or click ***fx*** to create an expression for the value. If you need to create a variable, use the appropriate **Create** link in the Variables dialogue to create it at the appropriate scope level.



Here's an example of a Call Function action, with its input parameter, `arg1`, needing specification:



If a value is returned by the action, it is assigned to the auto-generated variable shown by the Store Result In property.

Add a Call REST Action

A Call REST action is used to call a REST API endpoint to create, update, delete or display records.

For API information about this action, including details about error handling and its return object, see Call REST in the *Oracle Visual Builder Page Model Reference*.

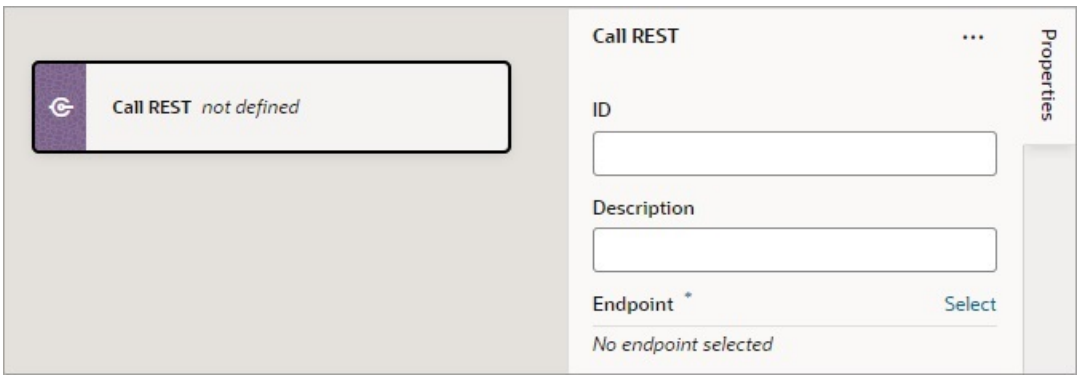
After you add a Call REST action, you need to specify the endpoint for the request. Depending on the endpoint, you might also need to provide input parameters, such as an ID to identify a record.

This table lists the parameters that you typically need to provide for a Call REST action, for each type of endpoint. For a code example of a call to each endpoint type, see Call REST in the *Oracle Visual Builder Page Model Reference*. Regarding the action's returned result, it's assigned to the automatically generated variable set for the Store Result In property.

Type of Endpoint	Use	Typical Requirements
POST	Add a new record.	• Provide the new record: In the Parameters section of the Properties pane, assign the variable containing the data to the body property. • Provide endpoint's input parameters, if any: If the endpoint requires input parameters, use the Input Parameters section in the Properties pane to provide the required input parameters. Required input parameters are marked with an asterisk.
GET	Get one or many records.	To get single record, provide the record's ID: In the Input Parameters section of the Properties pane, provide the record's ID using the input parameter for the record's ID.
DELETE	Delete a record.	• Provide the record's ID: In the Input Parameters section of the Properties pane, provide the record's ID using the input parameter for the record's ID.
PATCH	Update a record.	• Provide the updated data: In the Parameters section of the Properties pane, assign the variable containing the updated data to the body property. • Provide the record's ID: In the Input Parameters section of the Properties pane, provide the record's ID using the input parameter for the record's ID.

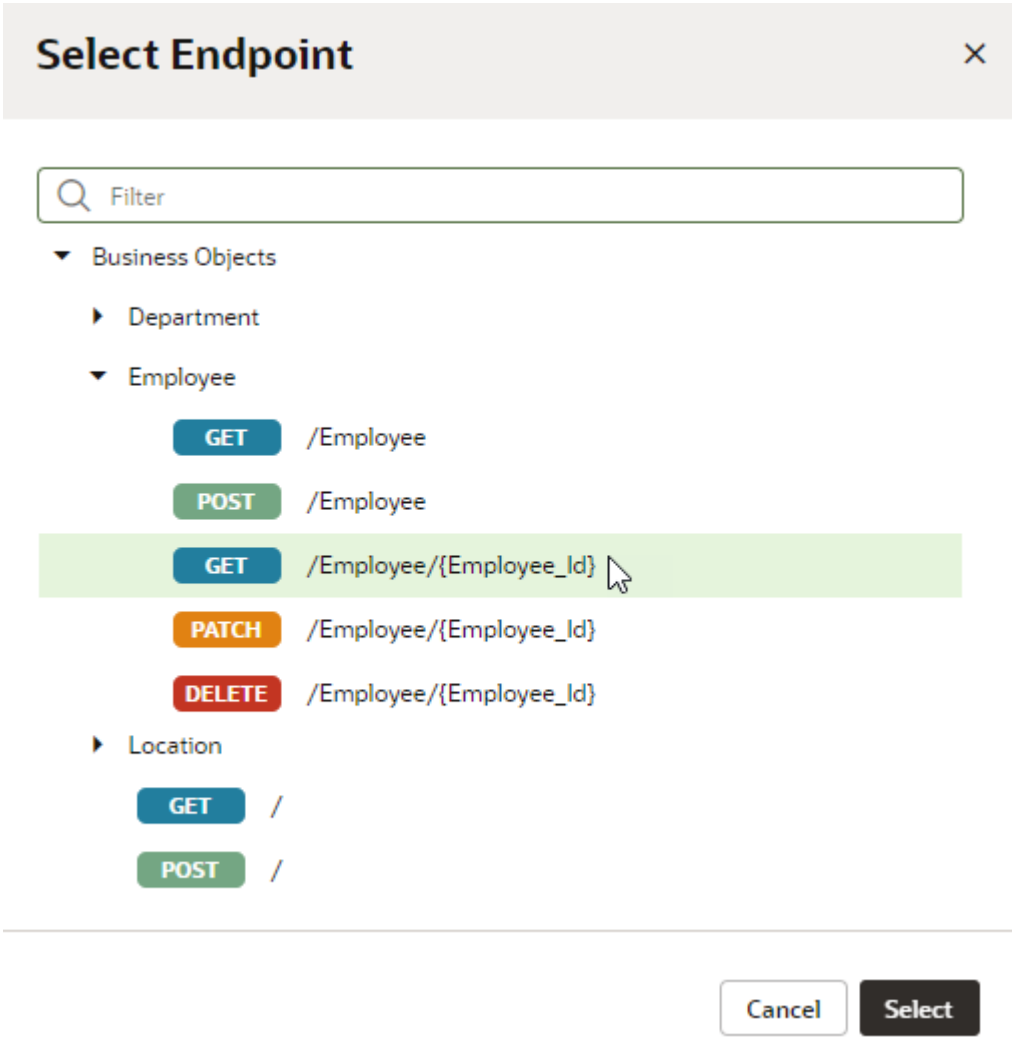
To use a Call REST endpoint:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



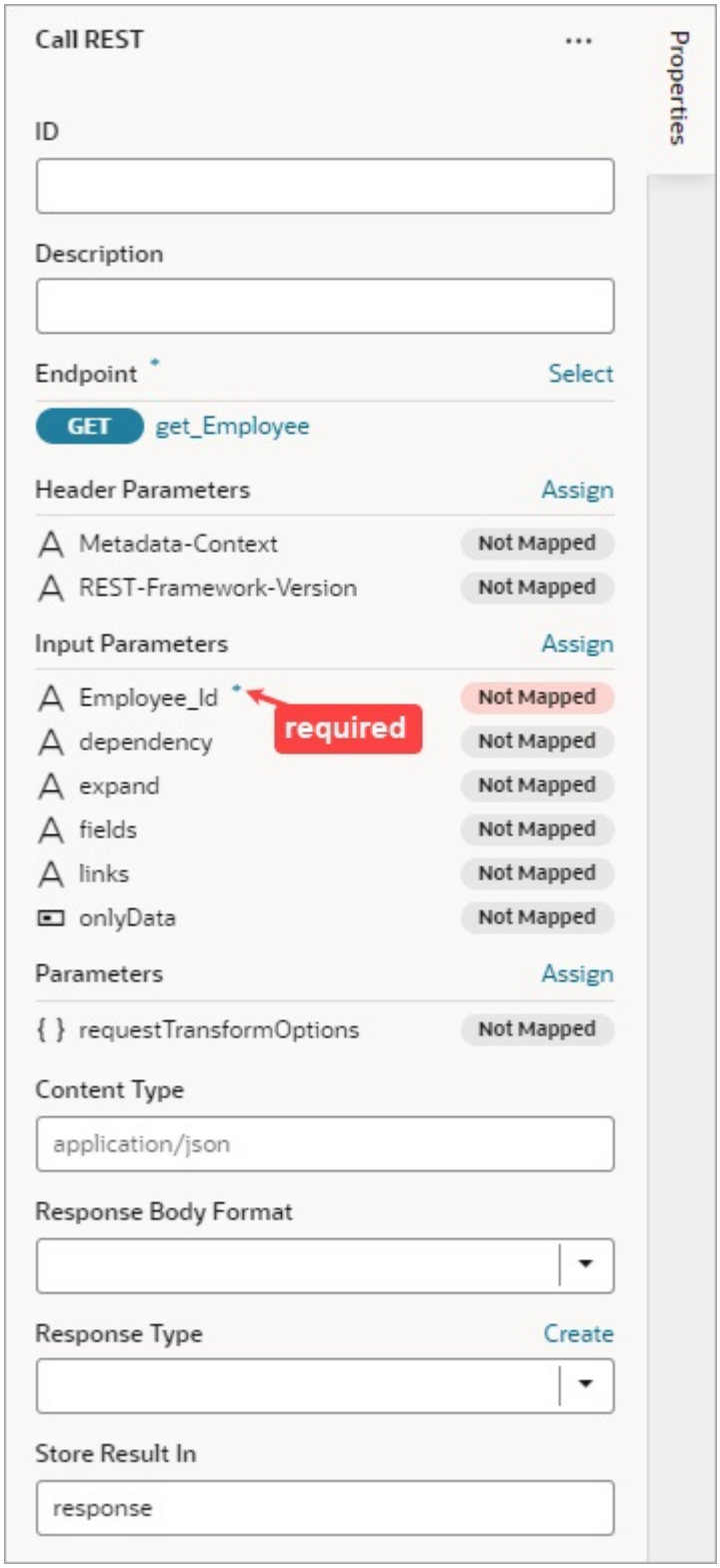
2. Click **Select** beside **Endpoint** in the Properties pane.

   The Select Endpoint window displays a list of endpoints that are available in your application. Each business object and service usually exposes multiple endpoints, each one for a different purpose. For instance, you can have an endpoint for getting multiple records, one for getting a single record, one for updating a record, and one for deleting a record. Each endpoint has different properties that you need to specify. For instance, for an endpoint that retrieves a single record, the record's ID must be provided.



3. Select an endpoint from the list and click **Select**.

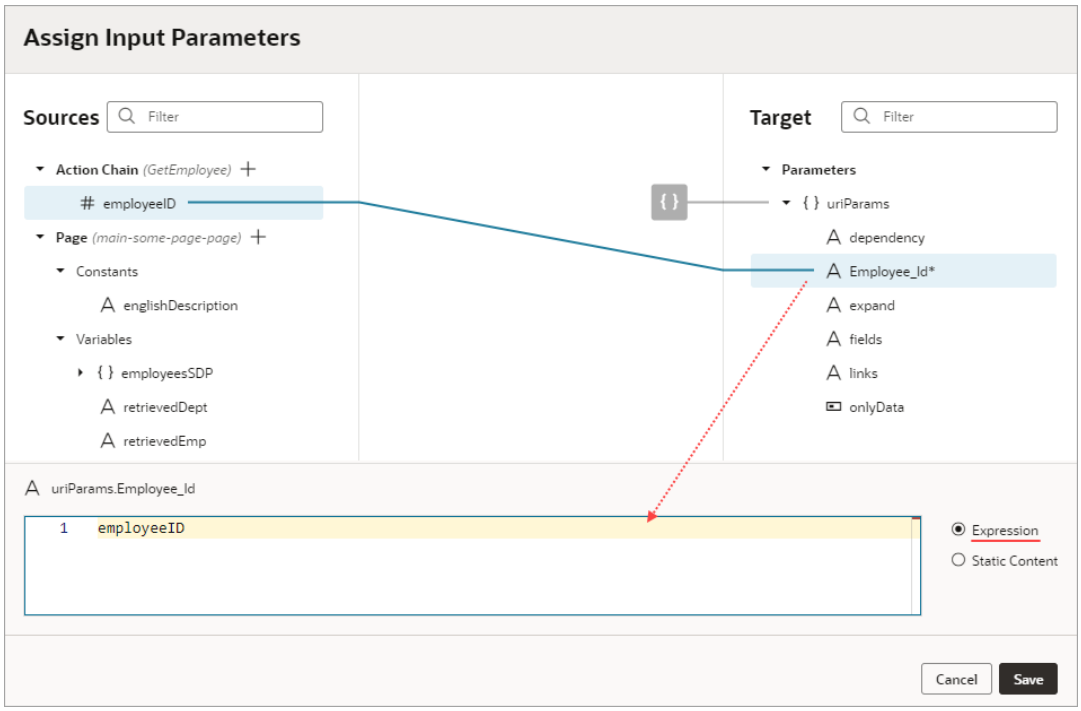
   The properties for the REST call are displayed in the Properties pane, with required properties marked with an asterisk (*):



4. If the REST call requires header or input parameters, click the associated **Assign** or **Not Mapped** link and use the assignment window to specify the value for the property. Click **Save**.

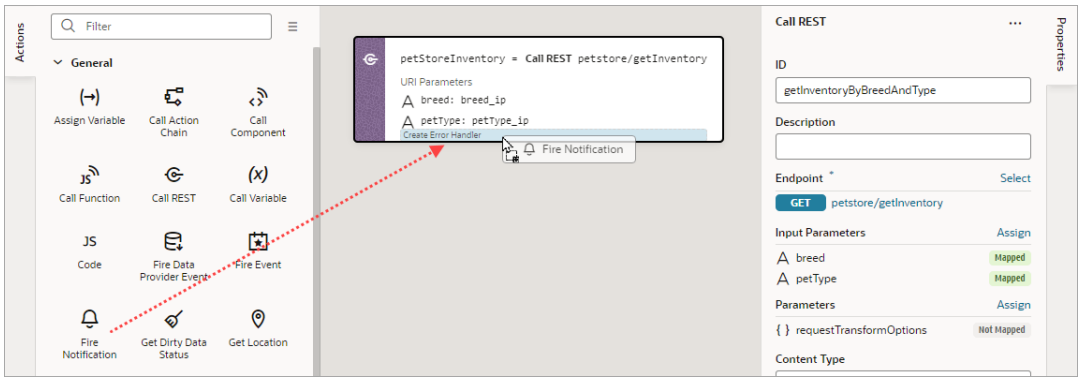
   You map variables to parameters in the assignment window by dragging the variable in the **Sources** pane onto the parameter in the **Target** pane. In some cases, you might need to

make multiple mappings. To delete a line mapping a variable to a parameter, right-click the line and select **Delete**. You can also select a parameter in the Target pane to view and edit the expression for its assignment in the lower pane.

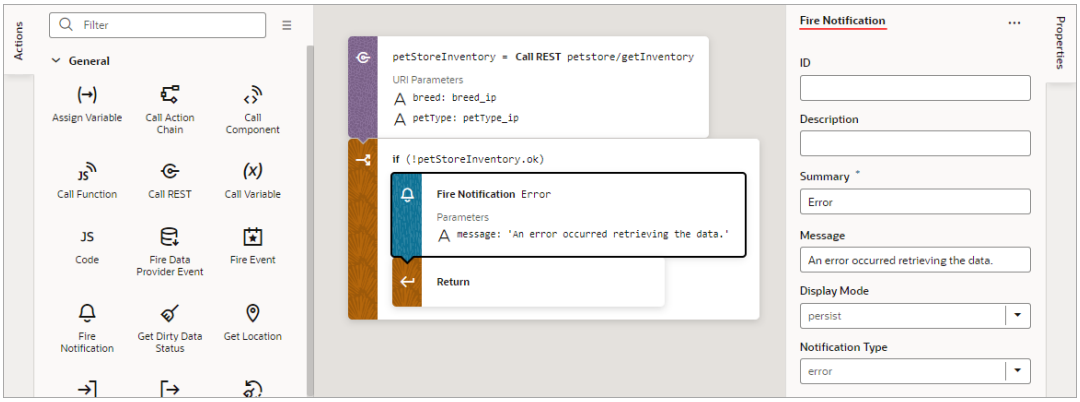


If a suitable variable does not exist, use the **+** icon beside the relevant node (Action Chain, Page, and so on) to create a new variable.

5. Specify any other properties that may be required for the REST call.
6. To handle a REST call error, drag the first action to take if an error occurred onto the **Create Error Handle** area that appears at the bottom of the Call REST card on the canvas:



The dropped action is added in an `If` condition that checks if an error occurred during the REST call. Configure the action in the Properties pane and add any other required actions to the condition.



The object returned by the Call REST action is automatically named and shown by the Store Result In property.

If the underlying REST API request returns a status code, the error object is returned for you to handle the error yourself, otherwise an auto-generated error notification is shown.

Service Definitions

To change the details of a service connection endpoint, such as its path, method (example: GET, PUT, HEAD) or the schema of the request or response, see Edit a Service Connection.

Transform Functions

For details about creating JavaScript functions that modify the format of data and parameters for REST requests and modify the format of data from Rest responses, see Add Transforms.

Add a Call Variable Action

You add a Call Variable action to an action chain to call a method on an InstanceFactory variable defined for the current scope (flow, page, or application). Using this action with any other type results in an error.

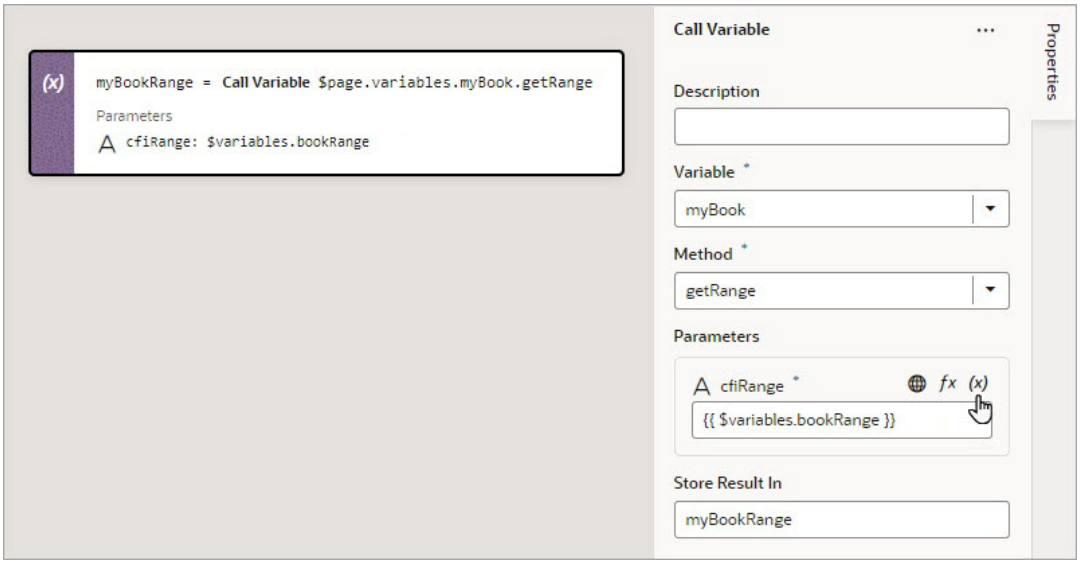
You can call any method on the current instance associated with the InstanceFactory variable, including asynchronous ones. However, since actions are by design synchronous, this action will wait for the asynchronous call to resolve before proceeding to the next action in the chain.

Before you use a Call Variable action, make sure an InstanceFactory type variable is already defined for the application. See Create a Type From Code.

For API information about this action, see Call Variable in the *Oracle Visual Builder Page Model Reference*.

To use a Call Variable action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. From the **Variable** list, select an InstanceFactory type variable defined for the application.
3. For the **Method** list, select the method you want to call. The available methods are based on the definition file imported for the type.



4. If the method requires input parameters, hover over the far-right side of a parameter and click to choose the value, or click ***fx*** to create an expression for the value.

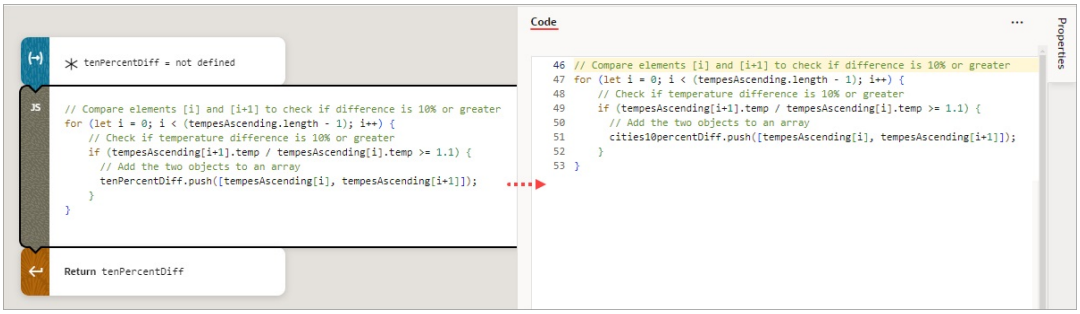
   If the method returns a value, it is assigned to the auto-generated variable shown by the Store Result In property in the Properties pane.

Add a Code Action

You use a Code action to add JavaScript code to an action chain.

To use a Code action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. Using the Properties pane, write or edit your custom code:



If needed, you can drop an action between single line and block statements in custom code, but not within a block statement (for example, not within a while or for block statement). A drop indicator line appears where you can drop an action within custom code.

Add a Fire Data Provider Event Action

You add a Fire Data Provider Event action to dispatch an event on a data provider in order to reflect changes to your data. For example, a component using a particular Service Data

Provider (SDP) may need to display new data because new data has been added to the endpoint used by the SDP.

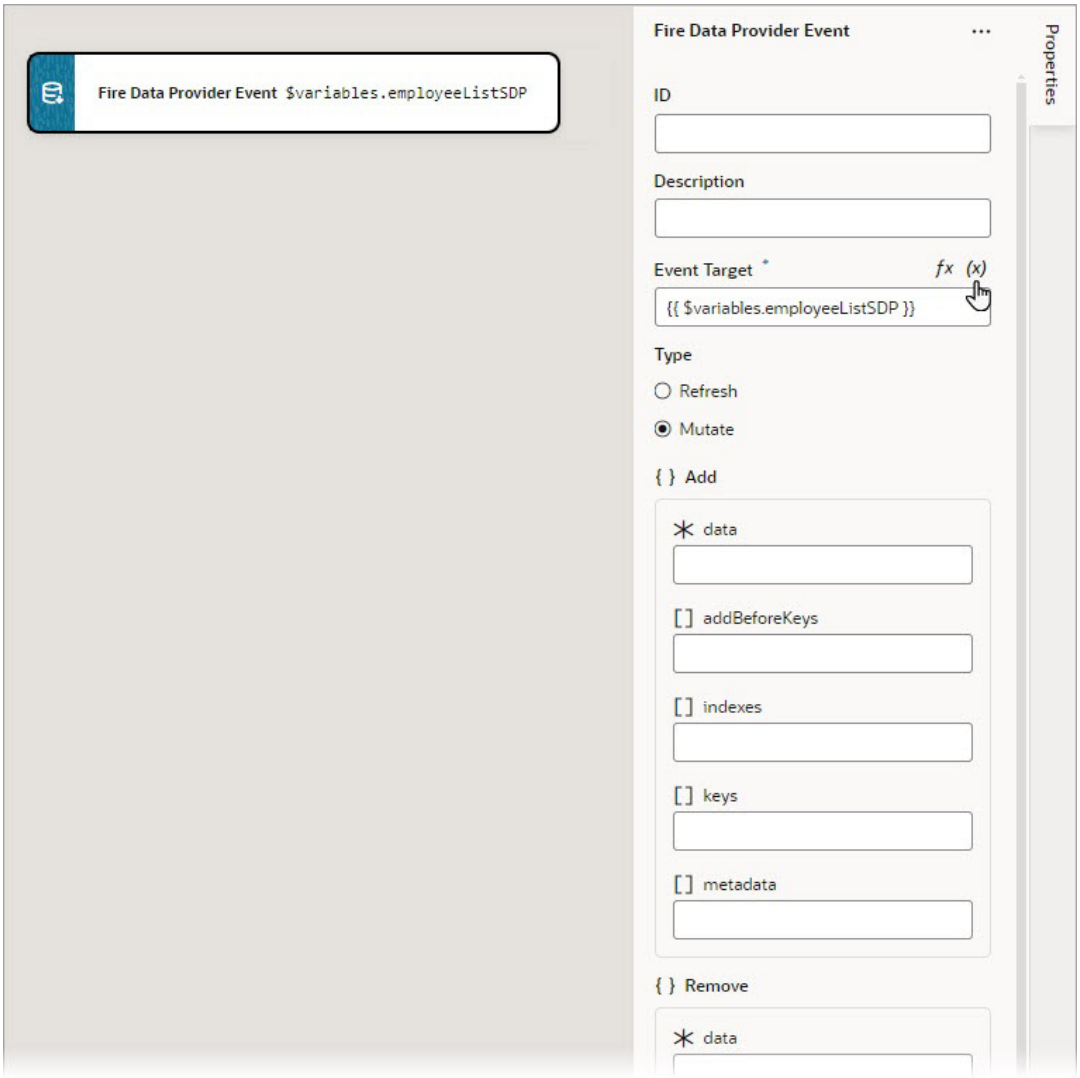
The action can be called either with a mutation or a refresh event. The refresh event re-fetches and re-renders all data, and the mutation event is used to specify which changes to show.

> **Note:**
>
> This action is not necessary for a VB Array Data Provider (ADP) variable, since the data array of an ADP variable, exposed via the `data` property, can be updated directly using the Assign Variable action. Assigning the data array is automatically detected by Visual Builder, and all listeners are notified of this change. Users will be warned of this when the fireDataProviderEvent is used with an ADP, prior to mutating the `data` property directly.

For API information about this action, including further details about its properties, see Fire Data Provider Event Action in the *Oracle Visual Builder Page Model Reference*.

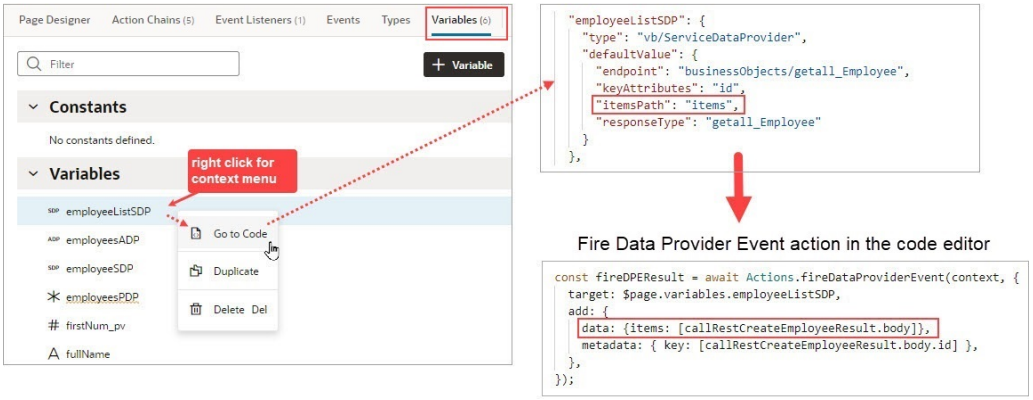
To use a Fire Data Provider Event action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. Update the **ID** field in the Properties pane to make the action more identifiable.
3. For **Event Target**, set the target of the event by hovering over the far-right side of the property and clicking to choose the relevant `ServiceDataProvider` or `ArrayDataProvider`.
4. Select the type of event you want to dispatch:
   - **Refresh**: Used to show all changes.
   - **Mutate**: Used to specify which changes to show. For example, after a record is deleted, you can use the Mutate option to indicate the deleted record, prompting the UI component bound to the SDP to remove it from the display.
     A mutation event can include multiple mutation operations (add, update, remove) 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.
5. If you chose the Mutate event, ensure that the `keyAttributes` property is set for the SDP variable. The parameters for showing the added, removed and updated records are under the **add**, **remove** and **update** sections. Here's what's required for each section, for the mutate event to perform optimally:

- **add** section: Use the **data** parameter to pass the added record or records from the add operation's returned result. If you are using an SDP variable, the structure of the data must match the structure specified by the `itemsPath` parameter of the SDP variable's definition:



  Use the **keys** parameter to pass the key values of the added records with the `Set<*>` format. Lastly, use the **metadata** parameter to pass the key values of the added records with the format `Array.<ItemMetadata.<KeyValue>>`.

  Example:

```
data: {items: [callRestCreateEmployeeResult.body],},
keys: [callRestCreateEmployeeResult.body.id],
metadata: [{key: callRestCreateEmployeeResult.body.id,}],
```

- **remove** section: Use the **keys** parameter to pass the key values of the deleted records with the `Set<*>` format
- **update** section: Same as the **add** section.

Add a Fire Event Action

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

A custom event, created using the Events tab, is defined for an application, flow, page, or fragment. It can be used to perform some action, such as navigating to a page, and it can carry a payload that you define when you create the event.

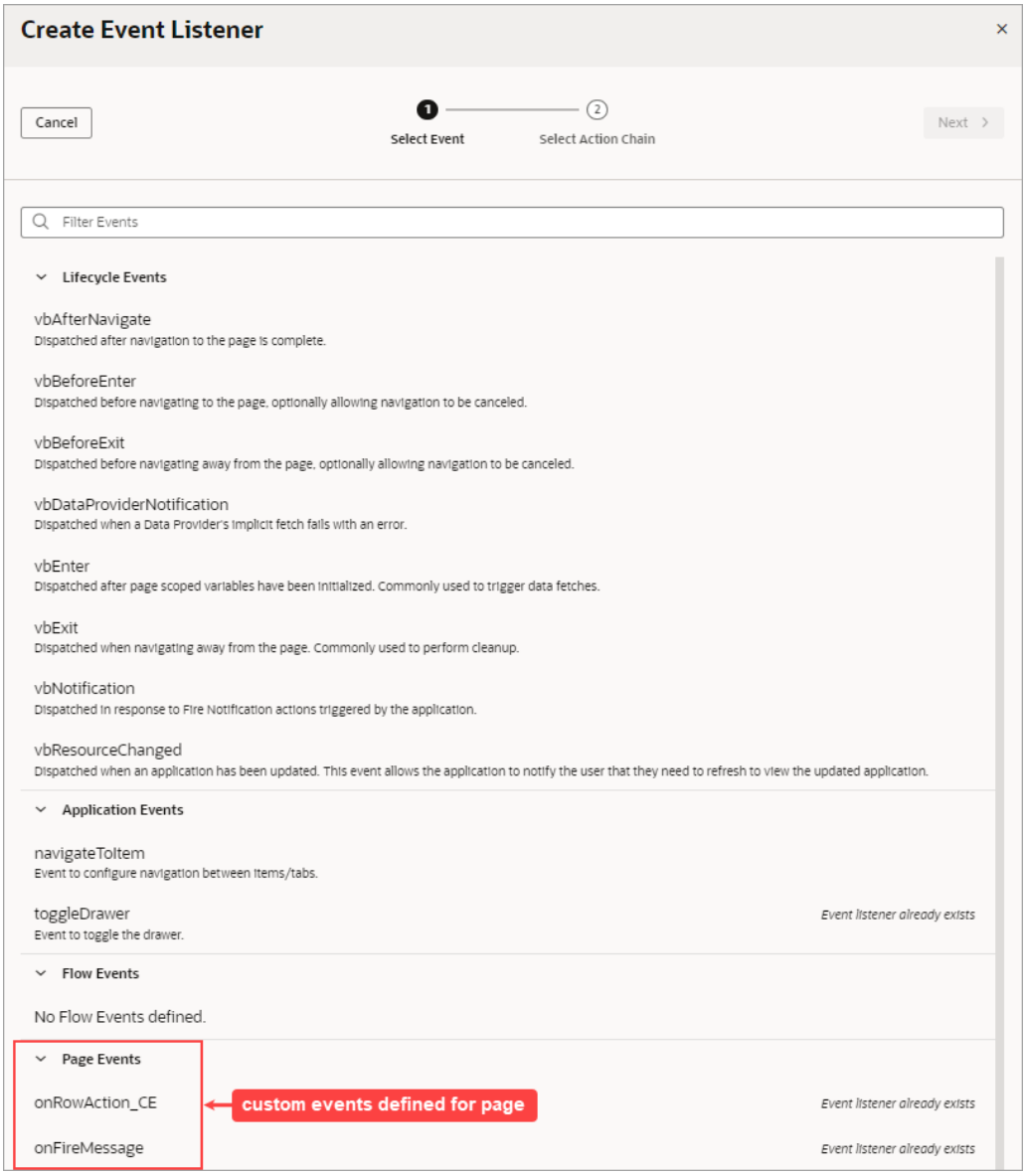
You can trigger a custom event by using a Fire Event action in an action chain, which can be started in several ways (see Start an Action Chain). You could also trigger a custom event by using an event helper's `fireCustomEvent()` method (see Module Function Event Helper) in a module function (JavaScript function). For more about triggering a custom event this way, see Start an Action Chain By Firing a Custom Event.

For API information about this action, see Fire Event in the *Oracle Visual Builder Page Model Reference*.

Here's an overview of how to use the Fire Event action in an action chain to trigger a custom event that starts a different action chain to handle the event:

1. Create a custom event, defining parameters if required.
2. Create an event listener, which can start more than one action chain:

a. In the Create Event Listener wizard, assign the event listener the custom event:



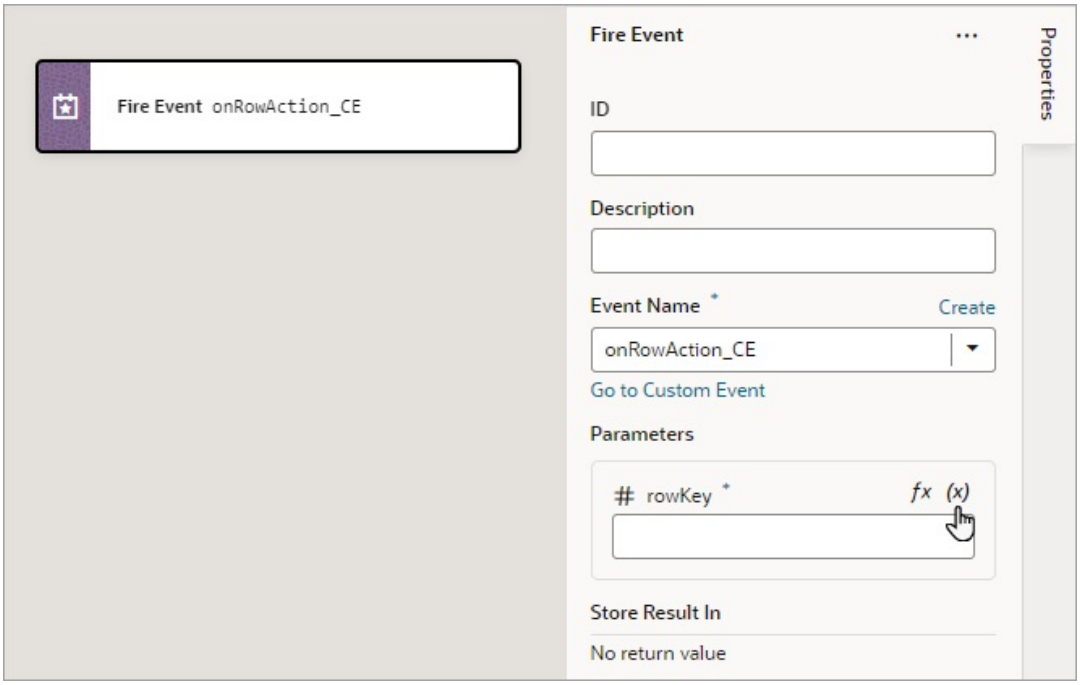
b. Create an action chain for the custom event, which will be started when the event is triggered by a Fire Event action. Create the action chain through the Event Listener tab, because if the listener's custom event has input parameters, the action chain is created with and passed an `event` object that contains the input parameters (example: `event.param1`, `event.param2`...).

3. To trigger the custom event and start its action chain, create an action chain and add a Fire Event action to trigger the custom event, providing any parameters defined for the event.

To use a Fire Event Action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. In the Properties pane, select an existing event from the **Event Name** list of available events, or click **Create** to create a new custom event at the appropriate scope level (page, flow, or application). The list contains events that are available within the current scope.

3. If the event has input parameters, the **Parameters** property is shown. To pass in a parameter, hover over the far-right side of the parameter and click to choose the variable. If you need to create a variable, use a **Create** link in the Variables dialogue to create it at the appropriate scope level.

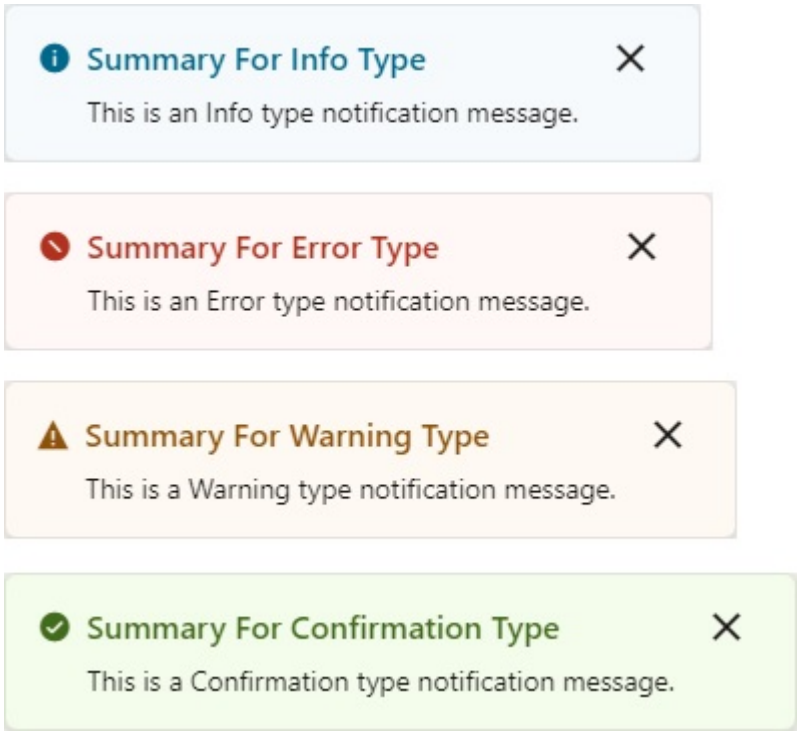


4. If the event returns a value, it is assigned to the auto-generated variable shown by the Store Result In property.

Add a Fire Notification Action

You add a Fire Notification action to display a notification to the user in the web browser.

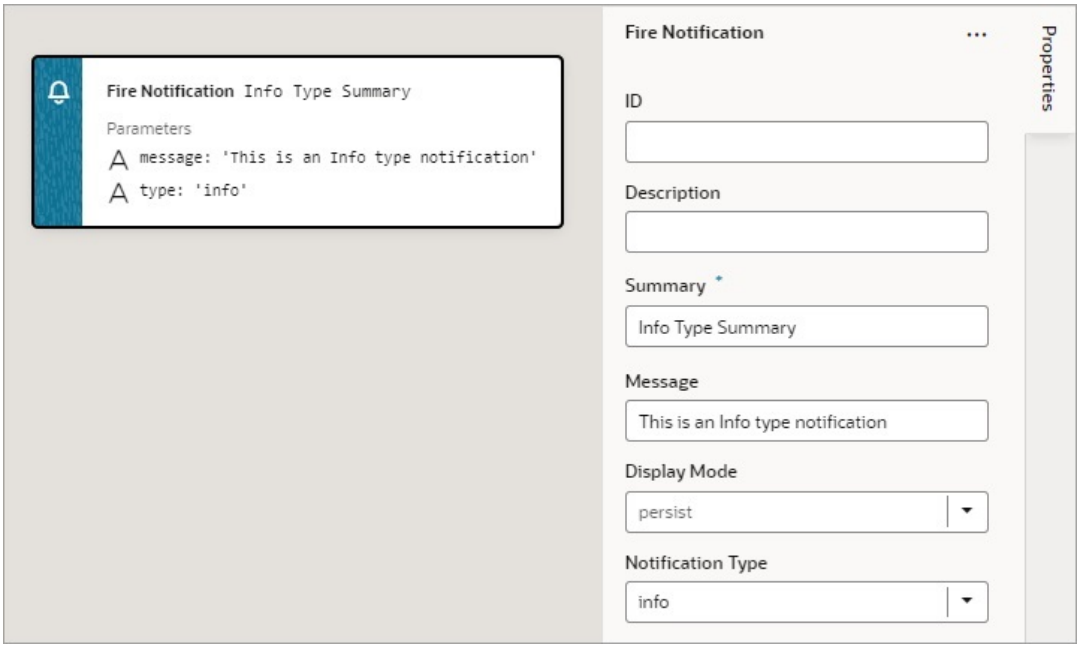
There are four types of notifications: Info, Error, Warning, and Confirmation. They display a summary and a message underneath:



For API information about this action, see Fire Notification Event in the *Oracle Visual Builder Page Model Reference*.

To use a Fire Notification action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. Update the **ID** field in the Properties pane to make the action more identifiable.
3. Enter a summary of the notification in the **Summary** field.

4. Enter the message you want to display in the **Message** field.

   The message can be a static string (`The name was updated.`) or it can contain variables (`{{ 'Could not create new Contacts: status ' + $chain.results.createContacts.payload.status }}`).

5. For **Display Mode**, choose how the notification is dismissed: **Transient** — goes away after a few seconds, or **Persist** — stays until the user closes it.

6. Select a **Notification Type** to specify the look of the notification window.

Add a For Each Action

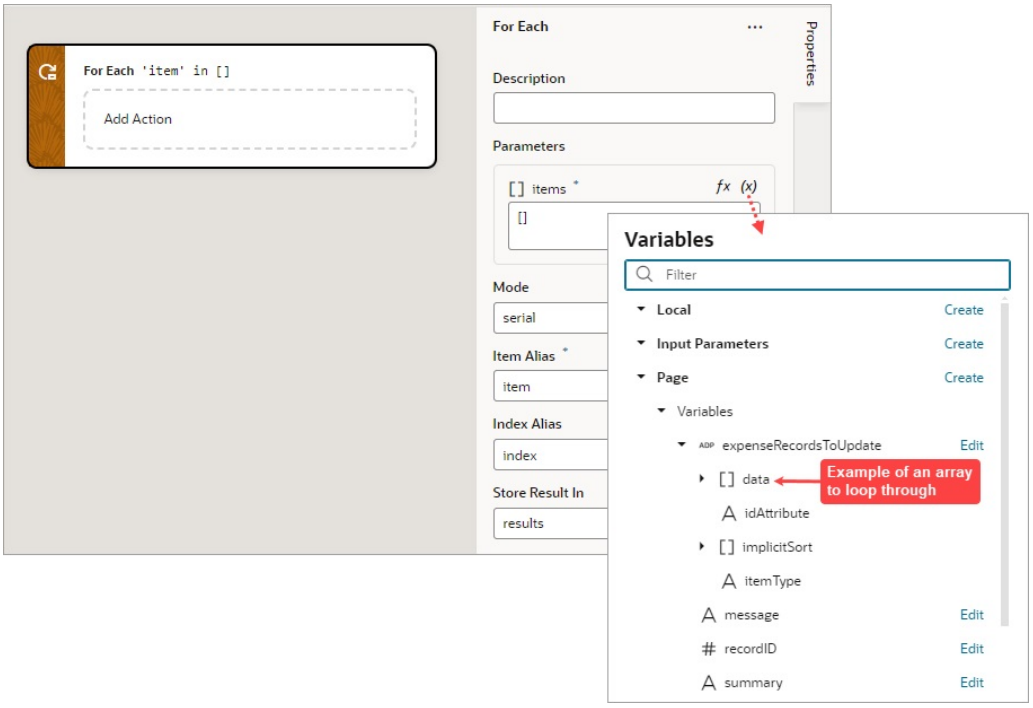
You add a For Each action to execute actions for each item in an array.

For API information about this action, see For Each in the *Oracle Visual Builder Page Model Reference*.

To use a For Each action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. Configure the action's properties in the Properties pane:

   a. For **Parameters**, hover over the property's far-right side and click to choose the array to loop through, such as this array, `$page.variables.expenseRecordsToUpdate.data`:

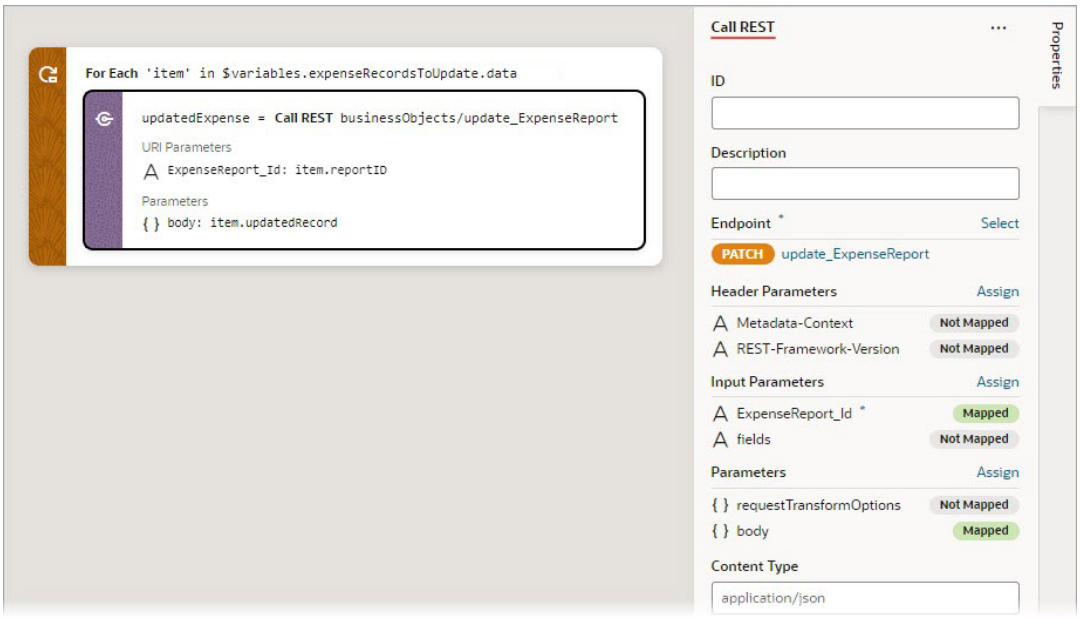


   b. For **Mode**, select whether your called actions run serially (default) or in parallel. Regardless of the mode, the For Each action will not complete until the actions for each item in the array are complete.

   c. For **Item Alias**, optionally, enter an alias for the current item in the array; the default is `item`.

d. For **Index Alias**, optionally, enter an alias for the loop index, which starts at 0 and increases by 1 for each iteration. The default alias is `index`.

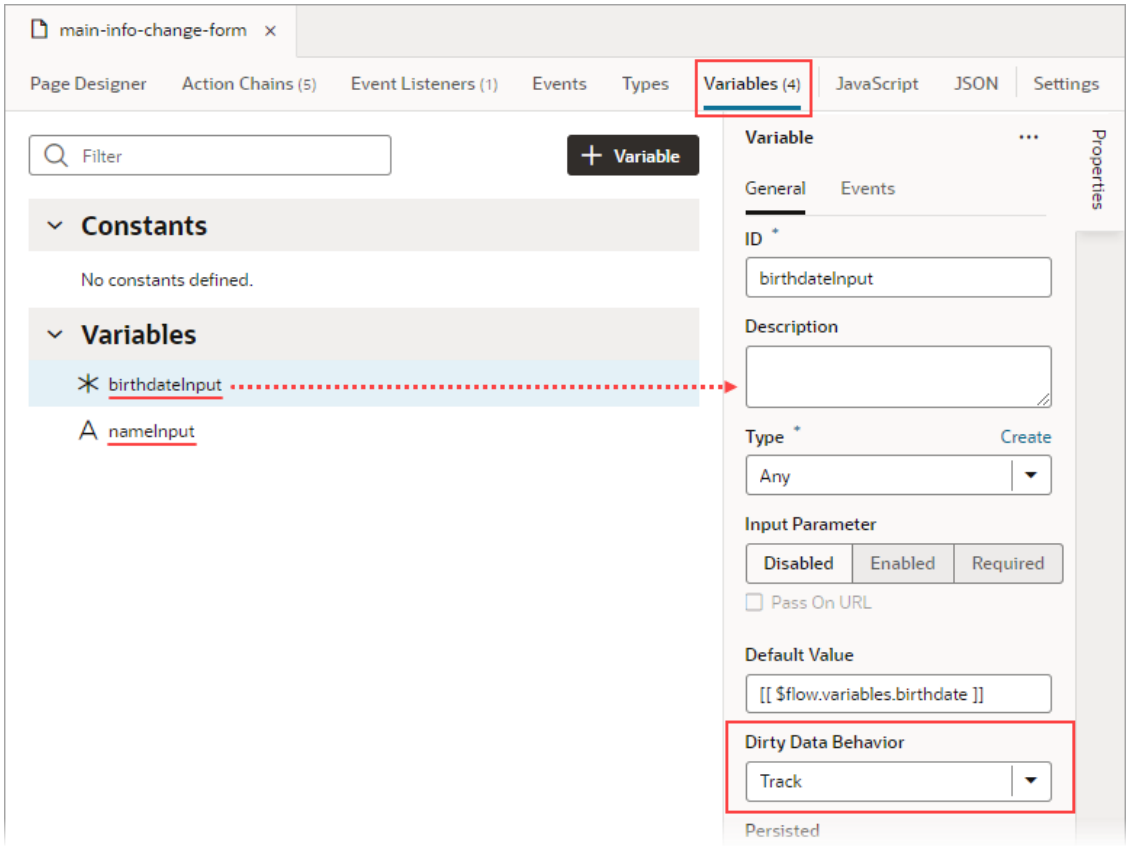
3. Add the actions you want to take for each item of the array to the **Add Action** area of the For Each action. Here's an example that loops to call a REST endpoint (`PATCH /ExpenseReport/{ExpenseReport_Id}`) that updates the expense record at the current iteration:



An array is returned with each element containing the return value from the last action in the loop, from each iteration. For instance, if the loop contains two actions that return results, actionA → actionB, and the loop iterates 5 times, the returned array will have 5 elements, each corresponding to an iteration and containing actionB's result from that iteration.

Add a Get Dirty Data Status Action

Use a Get Dirty Data Status action to check if any of the values have changed for the tracked variables within a particular scope (application, flow, page, fragment, layout), including any contained flows, pages, fragments, layouts, and their extensions. Tracked variables outside the scope being checked aren't considered. For example, in this image, both variables defined at the page level have their Dirty Data Behavior property set to Track.:



Whenever the value for any of these tracked variables changes, the dirty data status for their scope (referred to as `context` in code) is automatically changed from '`notDirty`' to '`dirty`'. If any tracked variables are defined for a layout, fragment, or extension of this page, and one of those variables changes, the page's dirty data status is also automatically set to '`dirty`'. To reset the scope's dirty data status back to '`notDirty`', use the Reset Dirty Status action.

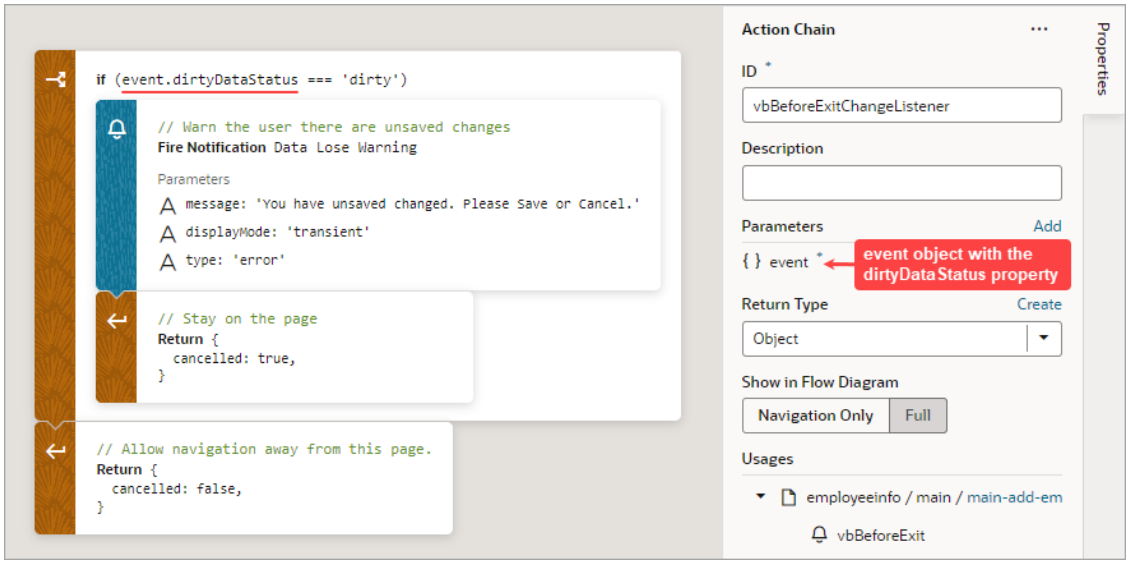
When checking the dirty data status of a particular scope and its subscopes, it's the scope from which the action chain is called that matters, not the scope in which the action chain is defined. For instance, if a page event initiates a flow's action chain with a Get Dirty Data Status action, the Get Dirty Data Status action returns that page's dirty data status, not the flow's, because the action chain is called from the page.

This functionality works with all data types except Service Data Providers (SDPs). You'll have to handle tracking value changes for SDPs manually, if needed.

For information about this action in the *Oracle Visual Builder Page Model Reference*, see Get Dirty Data Status.

Get Dirty Data Status Action vs Dirty Data Status Property

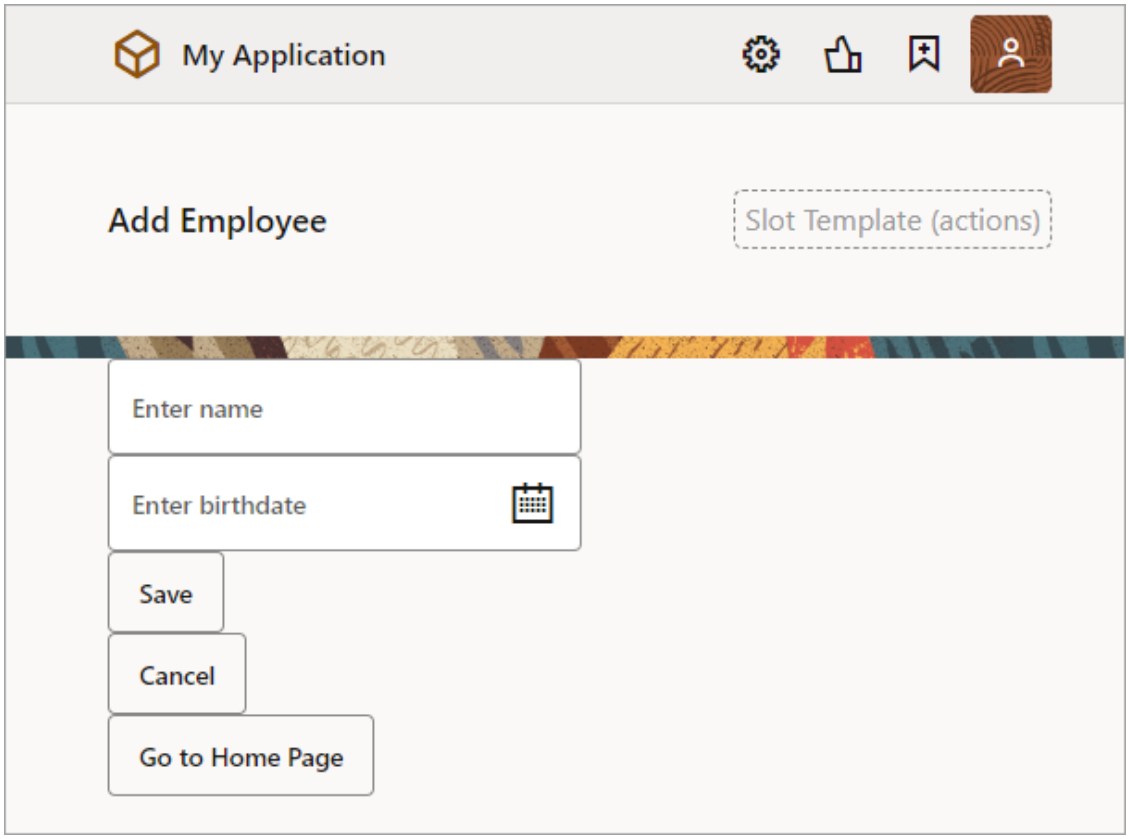
The Get Dirty Data Status action is used to check the dirty data status of a scope's tracked variables, so you can do things like implement a Save button that checks if a page actually has dirty data before posting its data to a database. The action doesn't account for the consequences of navigating away from the page. If you want to check if navigating away from a page will result in the tracked variables losing their data, use the page's `vbBeforeExit` event listener. This event listener is triggered when navigating away from the page and starts an associated action chain that receives an `event` parameter with a `dirtyDataStatus` property. Here's an example of an action chain started by a page's `vbBeforeExit` event listener, which prevents navigating away from the page if the tracked variables will lose their data:



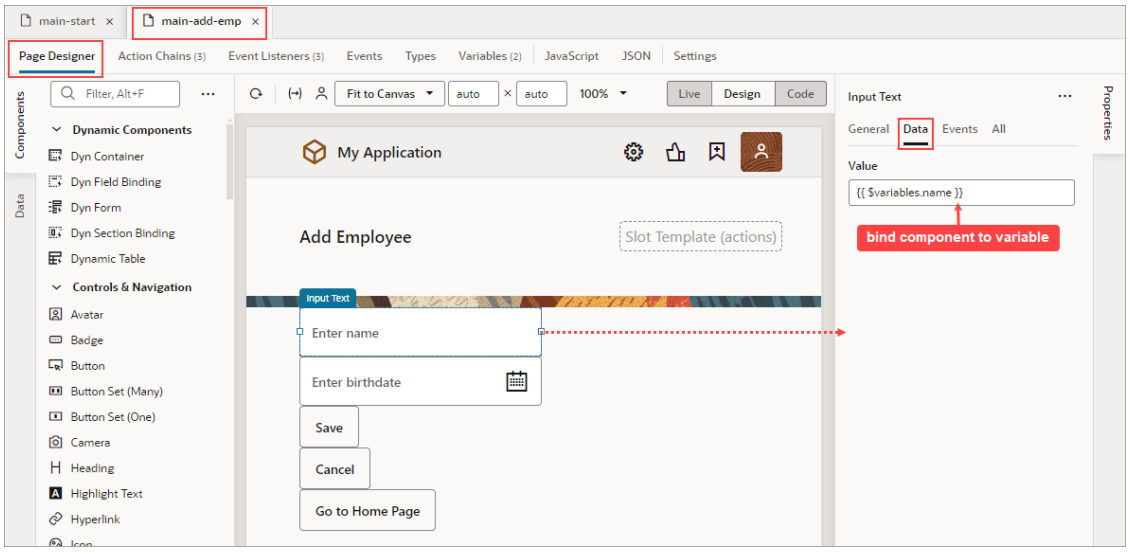
The `event.dirtyDataStatus` property, unlike the Get Dirty Data Status action, considers the effect that navigating away from the page would have on the tracked variables. The property is set to `'dirty'` if navigating away from the page will cause the tracked variables to lose their data; otherwise, it's set to `'notDirty'`. Variables within a scope retain their values as long as the destination of the user's navigation is within the scope or its subscopes. For instance, variables defined for a flow retain their values when a user navigates to a different page within the same flow, but not when navigating to a page in a different flow. Variables defined for a page do not retain their values when a user navigates to a different page in the same or a different flow.

Example:

This example shows an Add Employee page for adding a new employee's information. The page has two fields, one for a name and one for a birthdate, and their dirty data status needs to be tracked. The page also has a Save, Cancel, and Go to Home Page button. When the Save button is clicked, the employee's information is posted to storage.



The name and birthdate components are bound to page variables to hold their values:



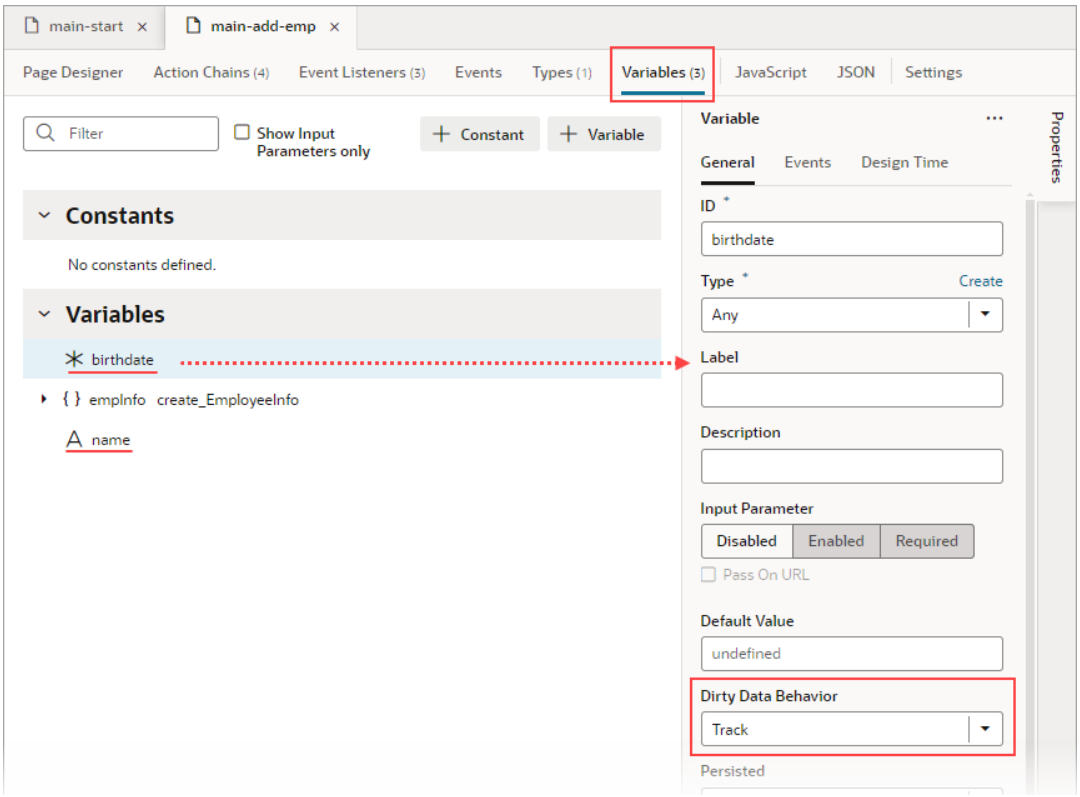
To add dirty data functionality to this example:

1. Set the page variables' Dirty Data Behavior property to "Track".
2. For the Save button's action chain, add a Get Dirty Data Status action to check if the page actually has dirty data (unsaved changes) before posting the new employee's information to storage.

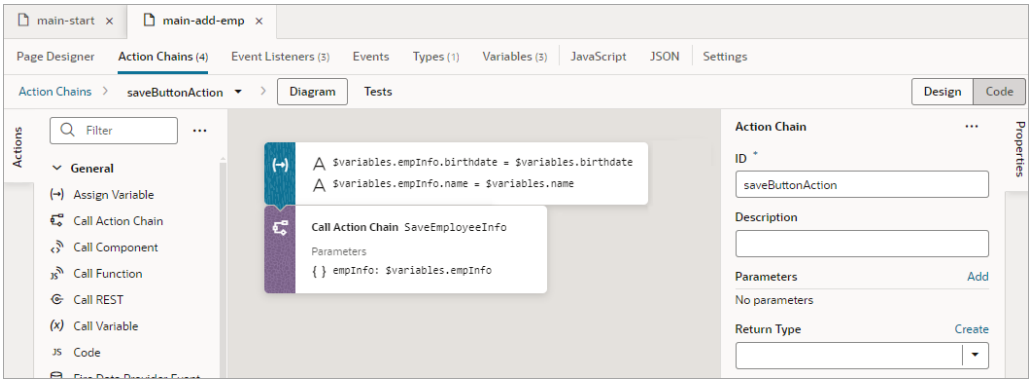
3. To warn users of unsaved changes due to navigating away from the page, add a `vbBeforeExit` event listener and use the `event` parameter's `dirtyDataStatus` property to check if navigation away from the page will result in the tracked page variables losing their data.
4. For the Cancel button's action chain, which navigates to the home page, add a Reset Dirty Data Status action to reset the page's dirty data status. This is needed for the `vbBeforeExit` event listener's action chain to allow the navigation to the home page when there is dirty data.

To begin:

1. Go to the page's **Variables** tab and set the page variables' **Dirty Data Behavior** property to **Track**:



2. To add the dirty data functionality to the Save button's action chain, to check if there's actually dirty data before posting:
   a. Go the Save button's action chain. Here's an example, which passes a new employee's information to an action chain that posts the data to storage:

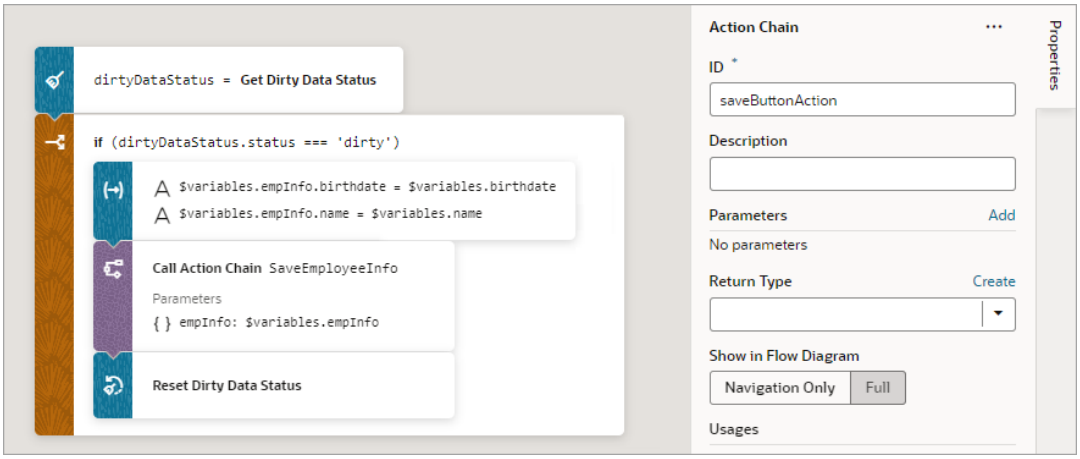


b. Add a Get Dirty Data Status action to the top of the action chain.

c. Wrap the code for posting the new employee's data in an If action to check if there's dirty data to post. If there's no dirty data, do nothing.

d. At the end of the If action's code, add a Reset Dirty Data Status action to reset the dirty data status back to '`notDirty`'.

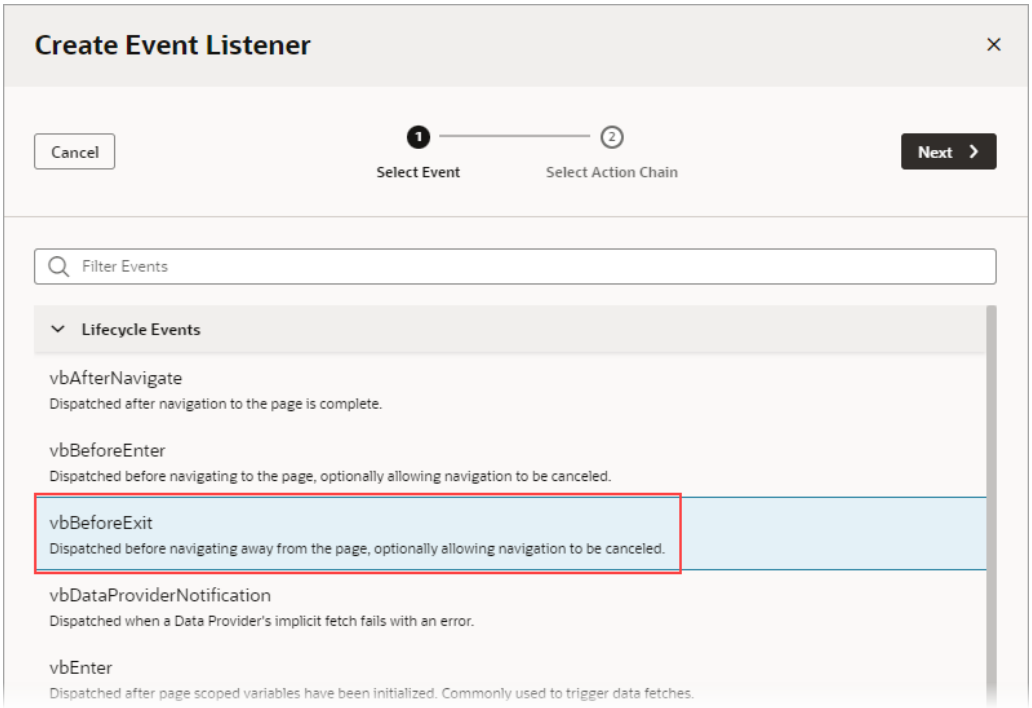
Here's the action chain with the added dirty data functionality:



3. Next, we need to create a `vbBeforeExit` event listener to listen for when a user tries to navigate away from the page, which includes using the browser's Back and Forward buttons. If there are unsaved changes, a notification will warn the user of the unsaved changes and prevent the navigation. The event listener's action chain will be automatically passed an `event` object with a `dirtyDataStatus` property to check if the navigation away from the page will result in the tracked page variables losing their data.
To begin:

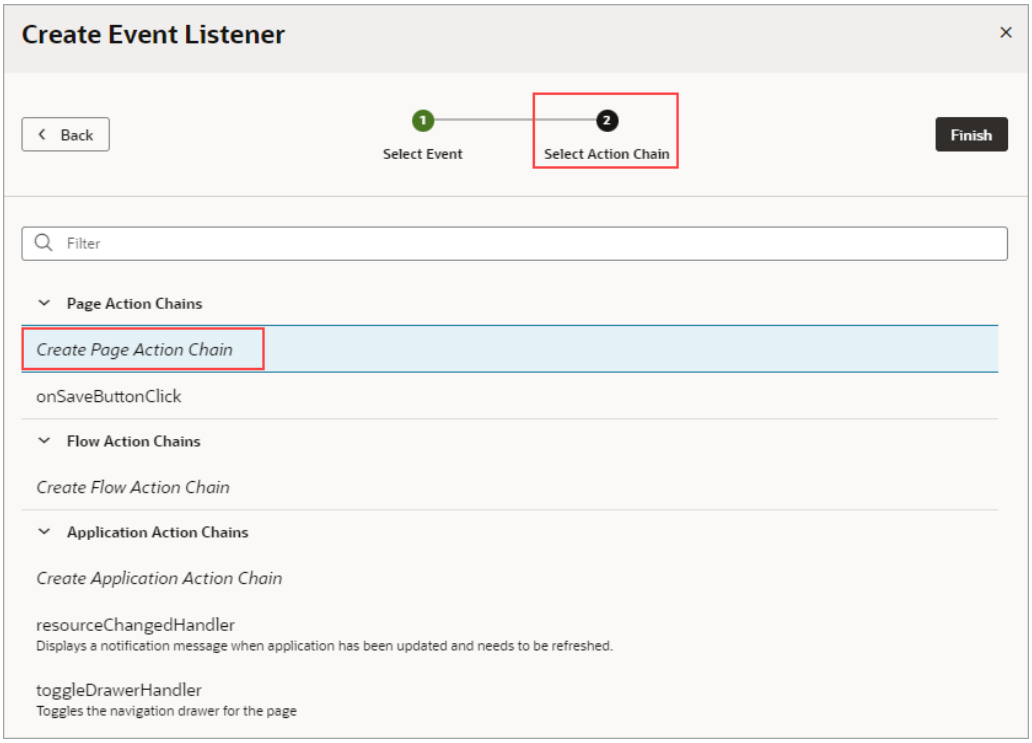
a. Open the **Event Listeners** tab and click the **+ Event Listener** button to create a new event listener.

b. Select **vbBeforeExit**, which starts its associated action chain whenever a user tries to navigate away from the page. Click **Next**:

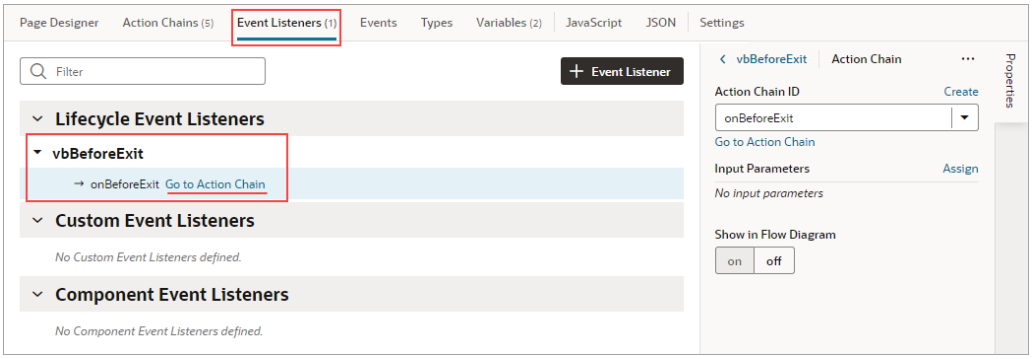


We now need to create the action chain that's started by this event listener.

c. On the Select Action Chain step of the Create Event Listener wizard, select the **Create Page Action Chain** option, under Page Action Chains. Click **Finish**.



d. Back on the Event Listeners tab, hover over the new event listener that you just created and click the **Go to Action Chain** link that appears:

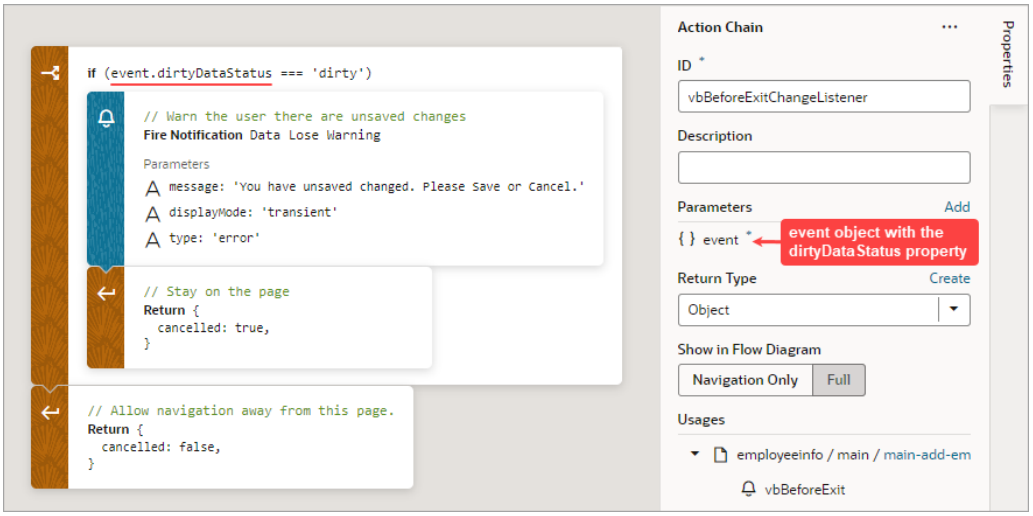


You're taken to the Action Chain editor to create the action chain that warns the user of unsaved changes.

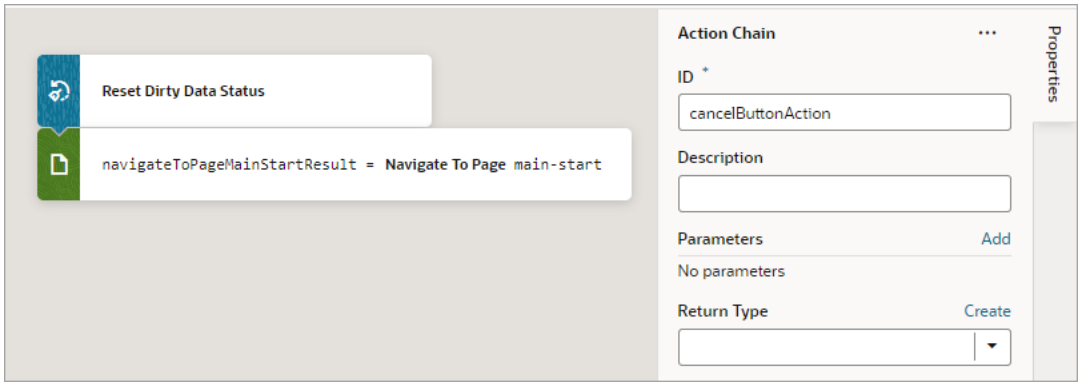
e. Add an If action to check if the navigation away from the page results in the tracked variables losing their data.
Use the `event` parameter that was passed to the action chain, which has a `dirtyDataStatus` property. The property is set to `'dirty'` if there will be lost tracked data, otherwise it's set to `'notDirty'`.

f. To handle the case in which the navigation results in a lose of tracked data, within the If action, add a Fire Notification action to notify the user of unsaved changes. To prevent the navigation away from the page, add a Return action to return the return object with its `cancelled` property set to `true`.

g. To handle the case in which there is no dirty data, return the return object with its `cancelled` property set to `false`.
Here's the completed action chain:



4. Finally, go to the Cancel button's action chain, which navigates to the home page. Add a Reset Dirty Data Status action to reset the page's dirty data status back to 'notDirty'. This is needed in case there's dirty data, which would prevent the `vbBeforeExit` event listener's action chain from allowing the navigation to the home page.



Add a Get Location Action

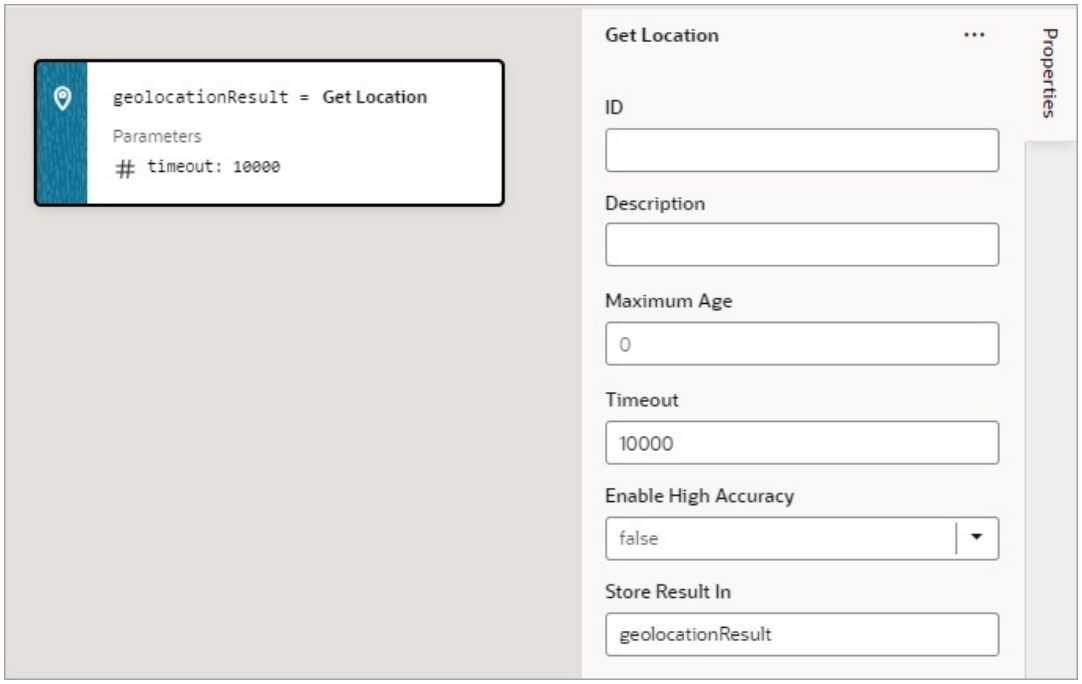
You add a Get Location action to get a user's live location.

This action requires the user's consent. As a best practice, it should only be fired on a user gesture, so the users can associate the system permission prompt for access with the action they just initiated.

For API information about this action, see Get Location in the *Oracle Visual Builder Page Model Reference*.

To use a Get Location action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. Update the **ID** property in the Properties pane to make the action more identifiable.
3. Set the **Maximum Age** (in milliseconds) of a possible cached position that is acceptable to return. If set to `0` (default), 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.

4. Set the **Timeout** value, representing the maximum length of time (in milliseconds) that the device is allowed to take in order to return a position.
5. Set the **Enable High Accuracy** value that indicates whether 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` (default), the device can save resources by responding more quickly or using less power. For mobile devices, you should set this to true in order to use GPS sensors.

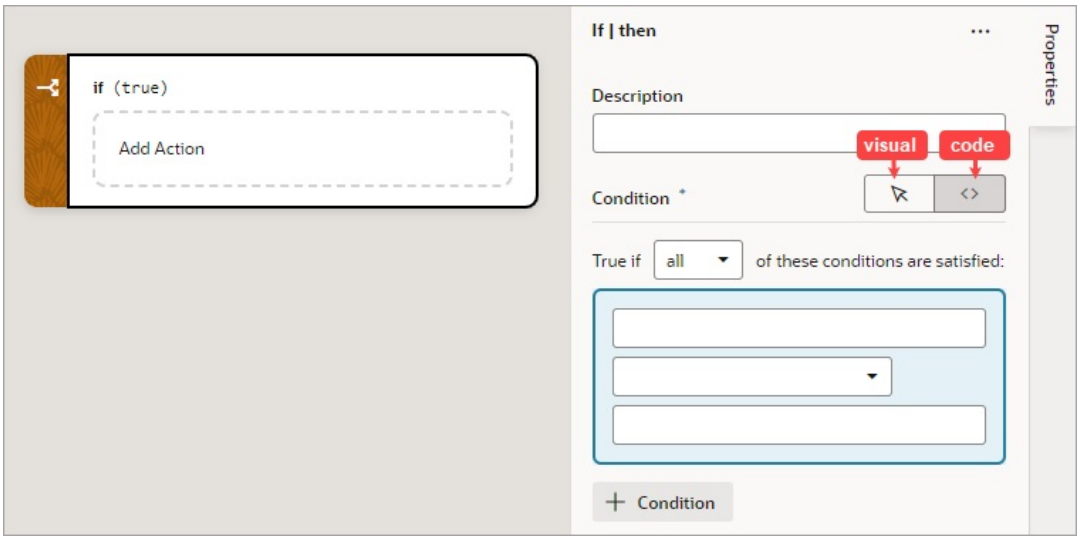
If a value is returned by the action, it is assigned to the auto-generated variable shown by the Store Result In property.

Add an If Action

You use this action to add If, Else and Else If conditions.

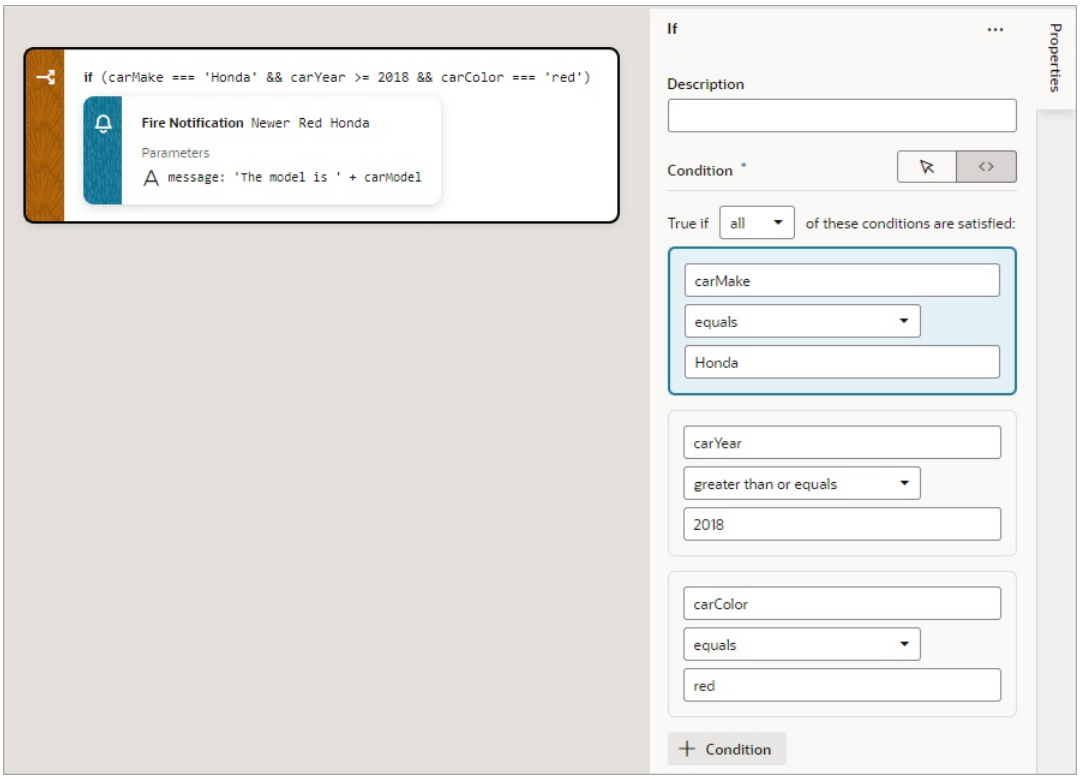
To use an If action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. With the If block selected on the canvas, use either the visual condition builder or the code condition builder to create the condition. To visually create the condition, click the first field in the blue box and select the variable to compare against a value or another variable. Select the comparison operator using the second filed, and enter the value or variable to compare against in the third field.



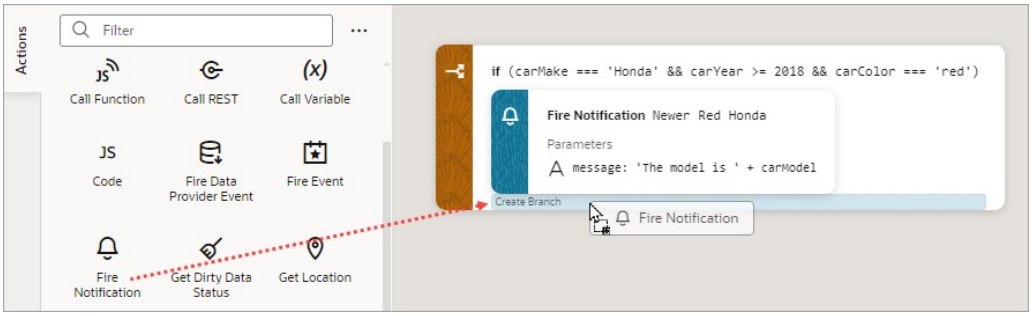
3. To add another condition, click the **+ Condition** button and create the condition as explained in the previous step.
4. To reorder, delete, or add a condition before or after an existing condition, right-click the condition to bring up its context menu.
5. To combine the conditions with an AND operator, select `all` from the **True if** list. To combine the conditions with an OR operator, select `any` from the list.
6. Add the action to take for the If block when the conditions are met. You can either select the block and double-click the action, or drag and drop the action onto the **Add Action**

area. If another action is to follow, double-click the next action or drag and drop it onto the bottom edge of the preceding action.



7. To add an Else or Else If condition, you have two options:

   - Drop the action to take for the new condition onto the **Create Branch** area at the bottom of the If block, which appears when you hover over it with an action.



   - Right-click an If or Else If block and select **Duplicate**.

   By default, an Else condition is created. To turn it into an Else If, enter a condition for it in the Properties pane.

Add a Login Action

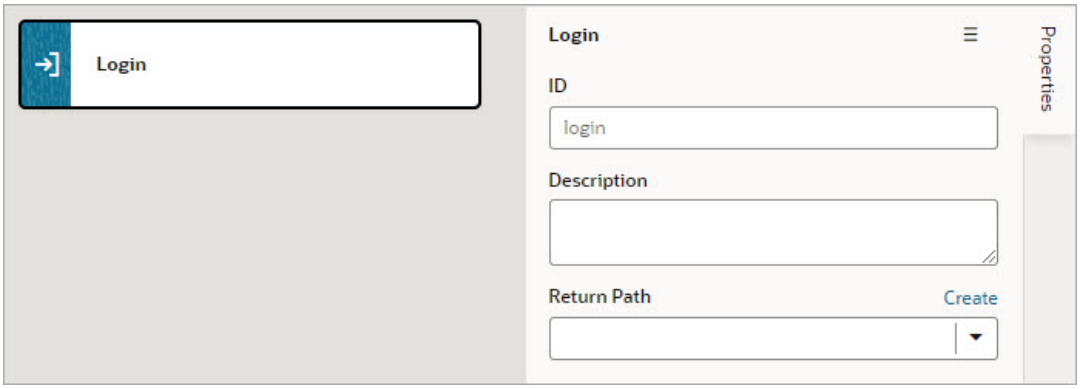
The Login action launches the URL specified by the Security Provider configuration. It invokes the `handleLogin` function on the Security Provider with this action's **Return Path** property (path of the root page or flow to go to after a successful login).

If the login was successful and the `Return Path` property was specified, the page specified by `Return Path` is launched. If the `Return Path` property wasn't specified, the application's default page is launched.

For API information about this action, see Login in the *Oracle Visual Builder Page Model Reference*.

To use a Login action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. Update the **ID** property in the Properties pane to make the action more identifiable, and optionally, enter a description for the action.
3. For **Return Path**, select the root page to go to when the login is successful, or click the **Create** link to create a new root page through the Create Root Page wizard.

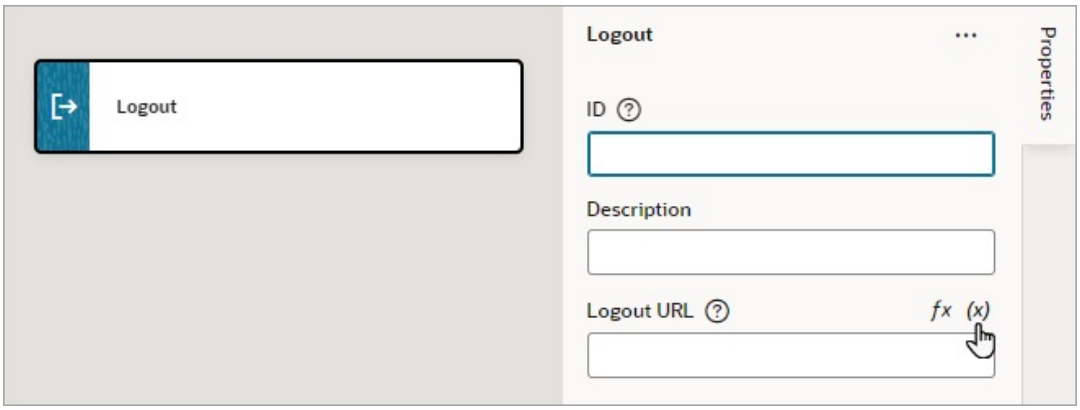
Add a Logout Action

The Logout action is used to launch a specified logout page, and if not specified, it automatically launches the logout URL specified for the Security Provider configuration.

For API information about this action, see Logout in the *Oracle Visual Builder Page Model Reference*.

To use a Logout action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. Update the **ID** property in the Properties pane to make the action more identifiable, and optionally, enter a description for the action.
3. For **Logout URL**, enter the URL to navigate to for logging out, or hover over the far-right side of the property and click to select the variable that holds the logout URL. If not defined, the logout URL from the Security Provider configuration is used.

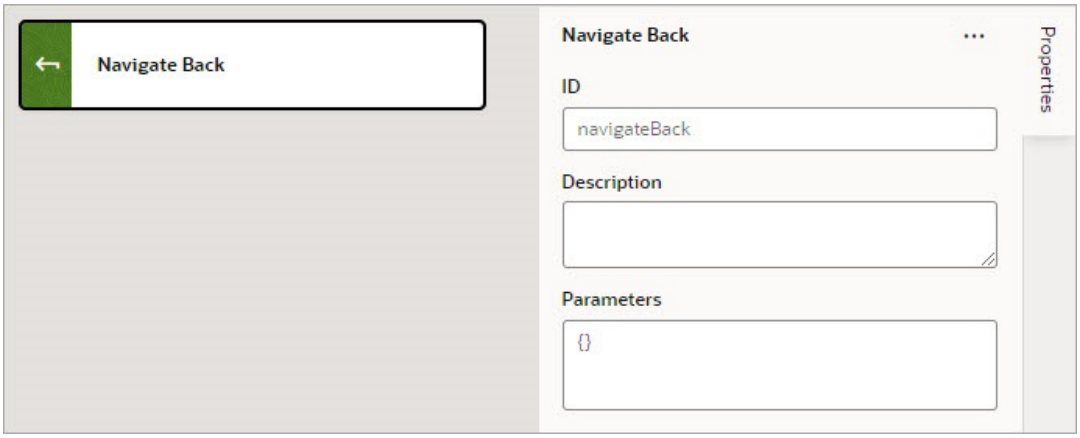
Add a Navigate Back Action

Add a Navigate Back action to return to the previous page in a browser's history.

For API information about this action, see Navigate Back in the *Oracle Visual Builder Page Model Reference*.

To use a Navigate Back action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. Optional: For **Parameters**, specify a key/value pair map of parameters to pass to the previous page. If a parameter is not specified, the original value of the input parameter on the destination page is used. If a parameter is specified, it has precedence over `fromUrl` parameters.

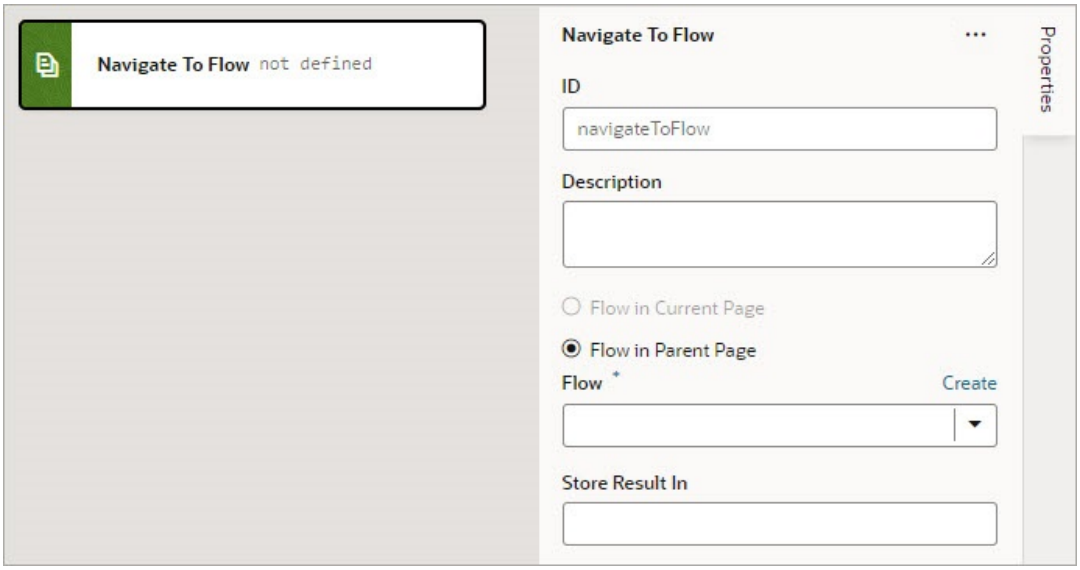
Add a Navigate To Flow Action

You use this action to navigate to a flow in the current application, and if necessary, to pass parameters to the flow.

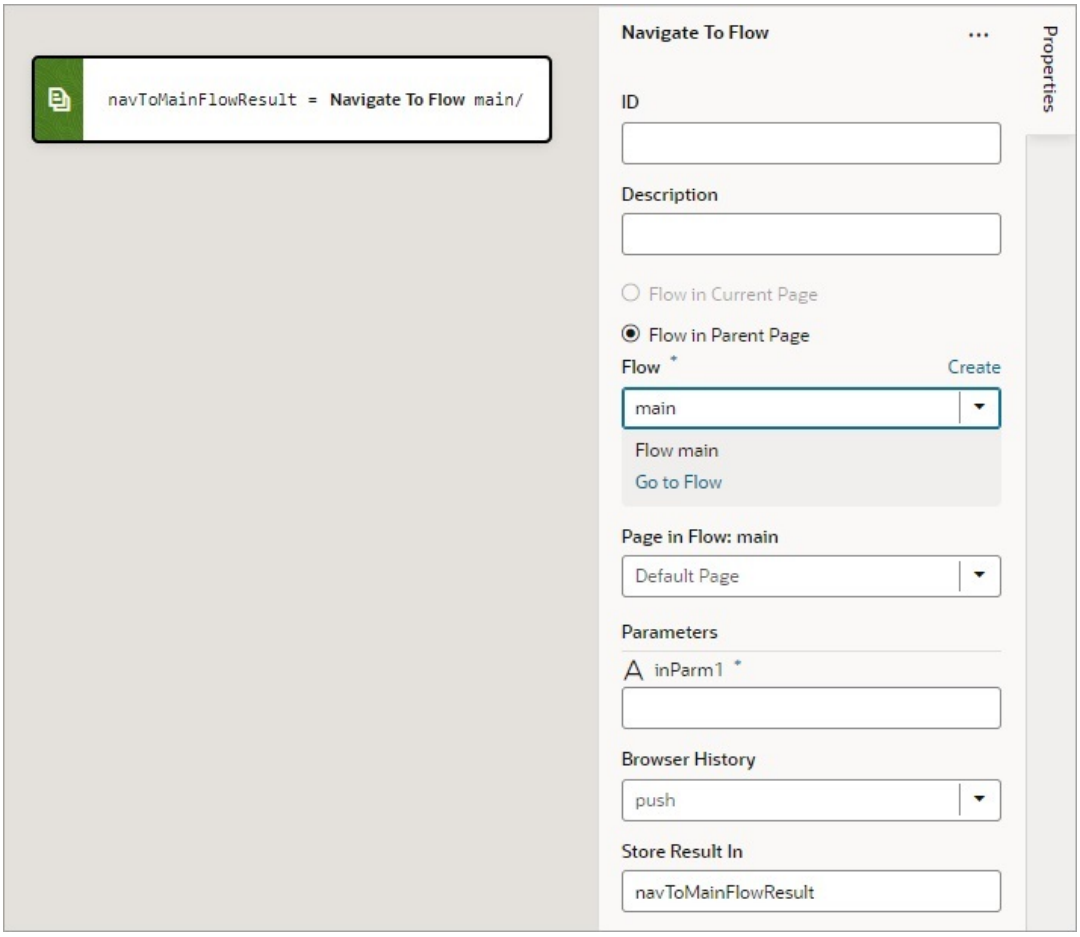
For API information about this action, see Navigate To Flow in the *Oracle Visual Builder Page Model Reference*.

To use a Navigate To Flow action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. For the flow options, the **Flow in Current Page** option is only available if the page includes a Flow Container component. Selecting it provides you with the options for navigating to a flow or page within the current page. Selecting **Flow in Parent Page** provides you with the options for navigating to a flow of the parent page.
3. For the **Flow** property, select a flow or click the **Create** link to create a new flow to navigate to.
4. If the selected flow has input parameters, enter them for the **Input Parameters** property that appears after selecting the flow.



5. For **Browser History**, select either `push` (default), `skip`, or `replace` to define the effect on browser history. This value is used only if the resource is used in the same window. If you choose `skip`, the URL is not modified. If you choose `replace`, the current browser history entry is replaced instead of pushed, meaning that the back button will not go back to that page.

If a value is returned by the flow, it is assigned to the auto-generated variable shown by the Store Result In property.

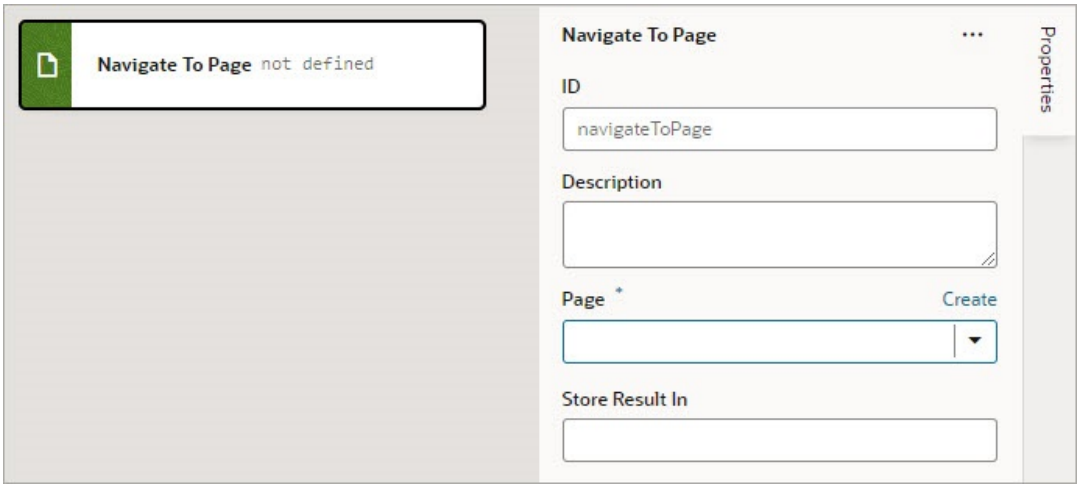
Add a Navigate To Page Action

You use this action to navigate to a page in the current application, and if necessary, to pass parameters to the page. To navigate away from the current application, use the Open URL action.

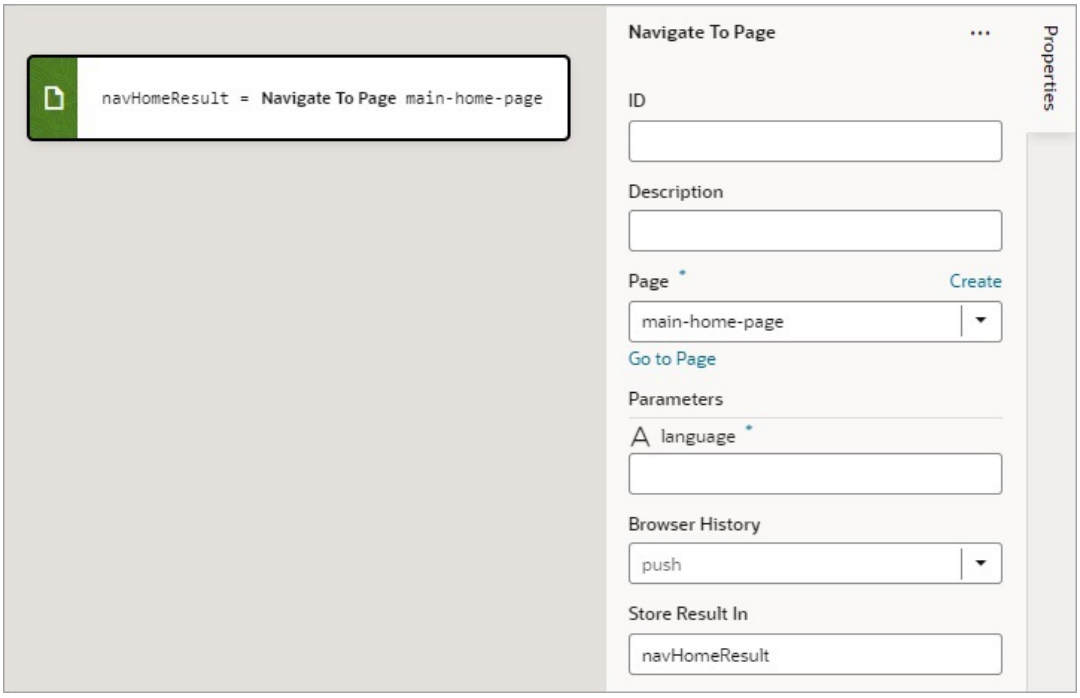
For API information about this action, see Navigate To Page in the *Oracle Visual Builder Page Model Reference*.

To use a Navigate To Page action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. For the **Page** property, select a page or click the **Create** link to create a new page to navigate to.
3. If the selected page has input parameters, enter them for the **Input Parameters** property that appears after selecting the page.



4. For **Browser History**, select either push (default), skip, or replace to define the effect on browser history. This value is used only if the resource is used in the same window. If you choose skip, the URL is not modified. If you choose replace, the current browser history entry is replaced instead of pushed, meaning that the back button will not go back to that page.

If a value is returned by the page, it is assigned to the auto-generated variable shown by the Store Result In property.

Add an Open URL Action

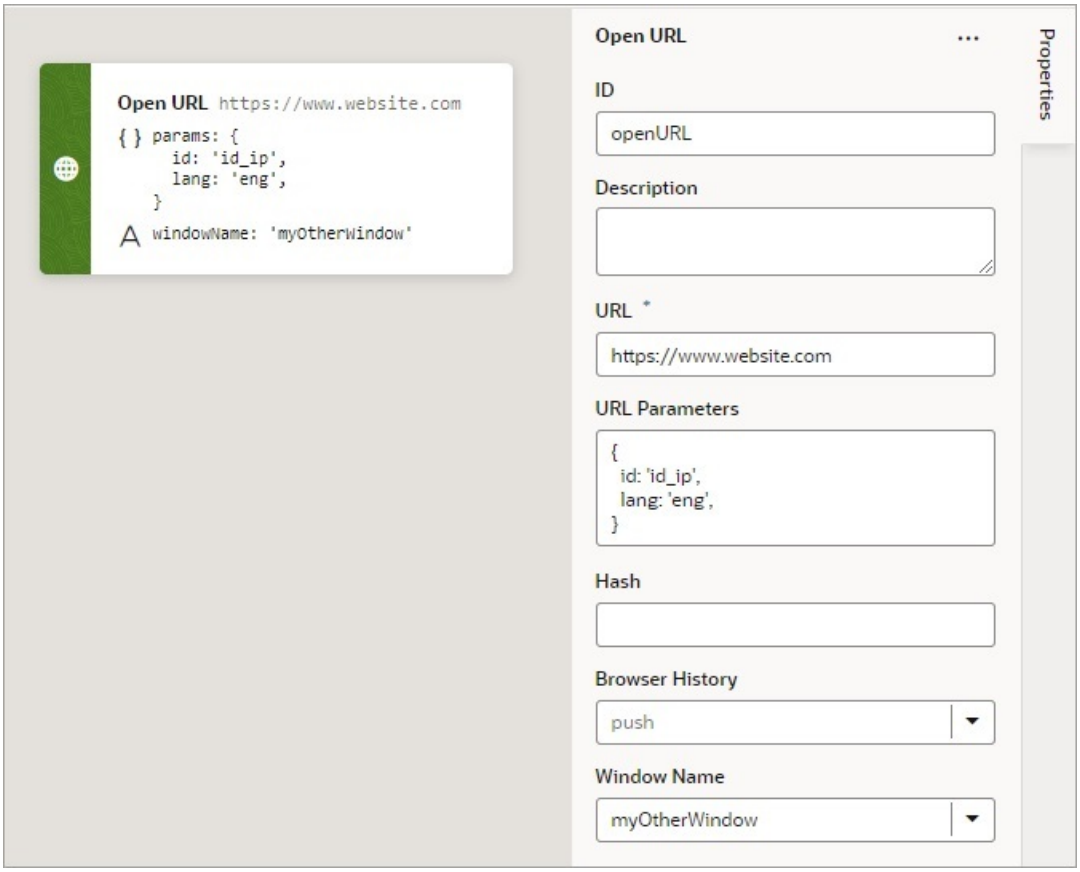
You add an Open URL action to navigate to an external URL. In a web app, this action opens the specified URL in the current window or in a new window.

For API information about this action, see Open URL in the *Oracle Visual Builder Page Model Reference*.

To use an Open URL action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. In the Properties pane, enter the **URL** to navigate to.
3. Optional: For **URL Parameters**, if required, provide a key/value pair map of query parameters to pass to the specified URL.
4. Optional: For **Hash**, specify the hash entry to append to the URL.
5. For **Browser History**, select either `replace` or `push` (default) to define the effect on browser history. This value is used only if the resource is used in the same window. If you chose `replace`, the current browser history entry is replaced instead of pushed, meaning that the back button will not go back to that page.
6. For **Window Name**, specify a name identifying the window as defined in the `window.open()` API. If not defined, the URL opens in the current window. For apps on mobile devices, you have three possible values: `_self` (default), `_blank`, or `_system`. For local file types, this property is ignored.

   Here's an example to open a new browser window with the specified URL. If you specify a value for the **Window Name** property (as shown here), once on the URL, the browser back button will re-enter the last page and the page input parameters will be remembered.



Add a Process Action

You use Process actions to run a business process, which automates a task that needs to be performed by application users, such as: reviewing data, approving or rejecting requests, and submitting files. For details about adding a business process action, see Add a Process to an Action Chain.

> **Note:**
>
> Oracle Process Cloud Service, which is included in the Enterprise edition of Oracle Integration Gen 2, is deprecated. See Process Features for details.

Add a Reset Dirty Data Status Action

You use a Reset Dirty Data Status action to reset the Dirty Data status of the scope (application, page, fragment, layout, flow) that the action is used in to '`notDirty`'. The Dirty Data status also gets reset for any tracked variables within any contained pages, fragments, layouts, and flows, and within any extensions of them. The Dirty Data status of a scope (referred to as context in code) changes from '`notDirty`' to '`dirty`' when one of its tracked variables has its value changed.

This action takes no parameters and is used with the Get Dirty Data Status action.

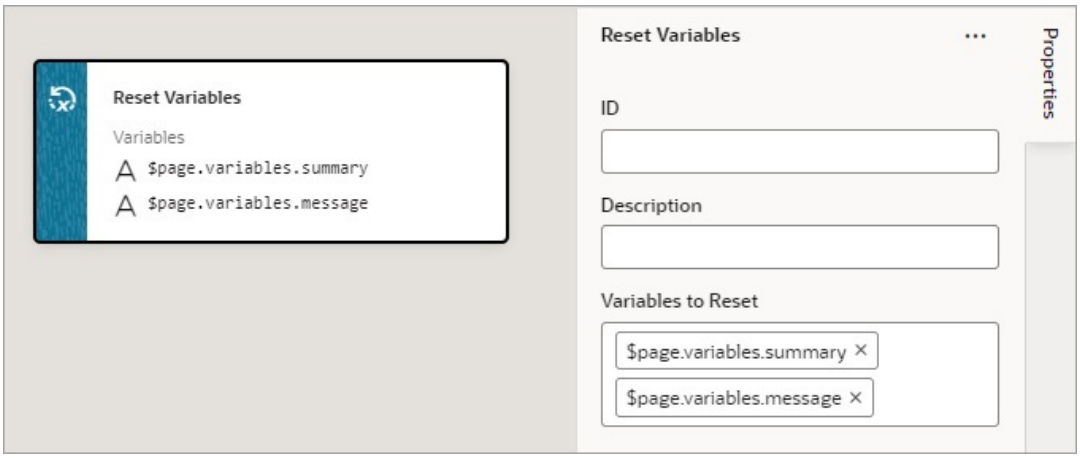
Add a Reset Variables Action

You add a Reset Variables action to reset variables to their default values, as specified in the variable definitions.

For API information about this action, see Reset Variables in the *Oracle Visual Builder Page Model Reference*.

To use a Reset Variables action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. Update the **ID** field in the Properties pane to make the action more identifiable.
3. Click the **Variables to Reset** box to select the variables that you want to reset. You can also start to type the variable's name in the box and select it when it appears.



Add a Return Action

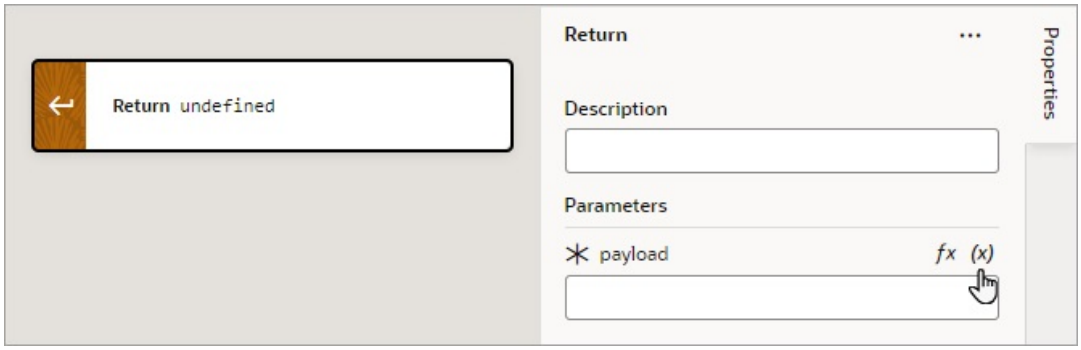
The Return action is used to return a payload for an action chain and to return control back to where the action chain was called. For instance, action chain A can call action chain B, which returns a value, then action chain A can use that returned value for further processing.

The Return action can also be used to exit an action chain early due to an exception, such as an invalid value, or some other condition. If no value is returned by the Return action, the value of undefined is returned by default.

For the Run In Parallel action, which uses `aysc()` functions to run blocks of code in parallel, the Return action can be used to return a value for a block of code. For further details, see the *Use 2: Run Multiple Action Chains in Parallel to Produce a Combined Result* section here.

To use a Return action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. For the **Payload** property, hover over the far-right side of the property and click to choose the variable to return.



Add a Run In Parallel Action

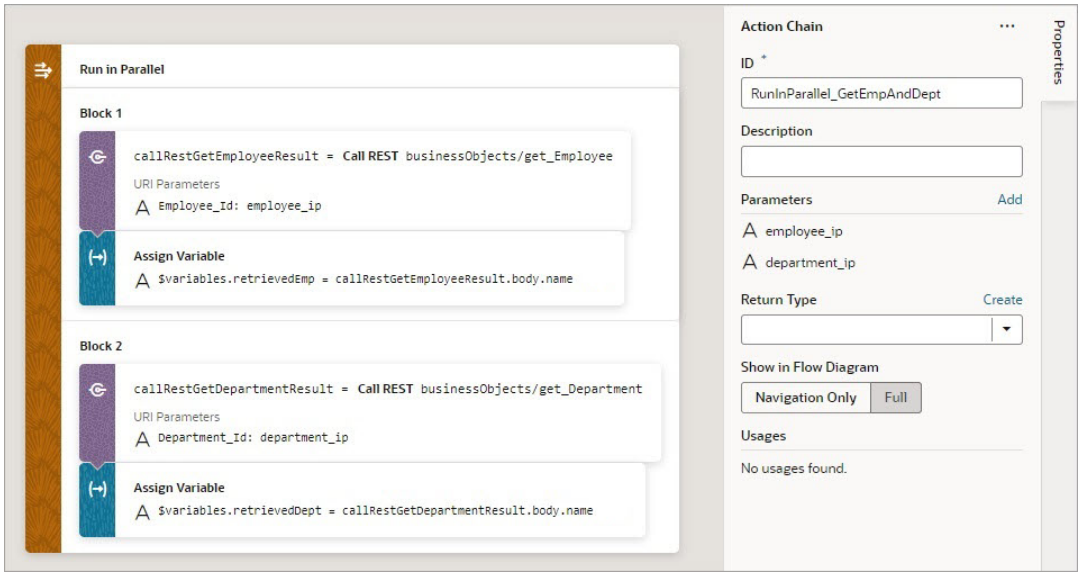
The Run In Parallel action is used to run multiple code blocks in parallel, and you can also use this action to wait for their results in order to produce a combined result.

For API information about this action, see Run in Parallel in the *Oracle Visual Builder Page Model Reference*.

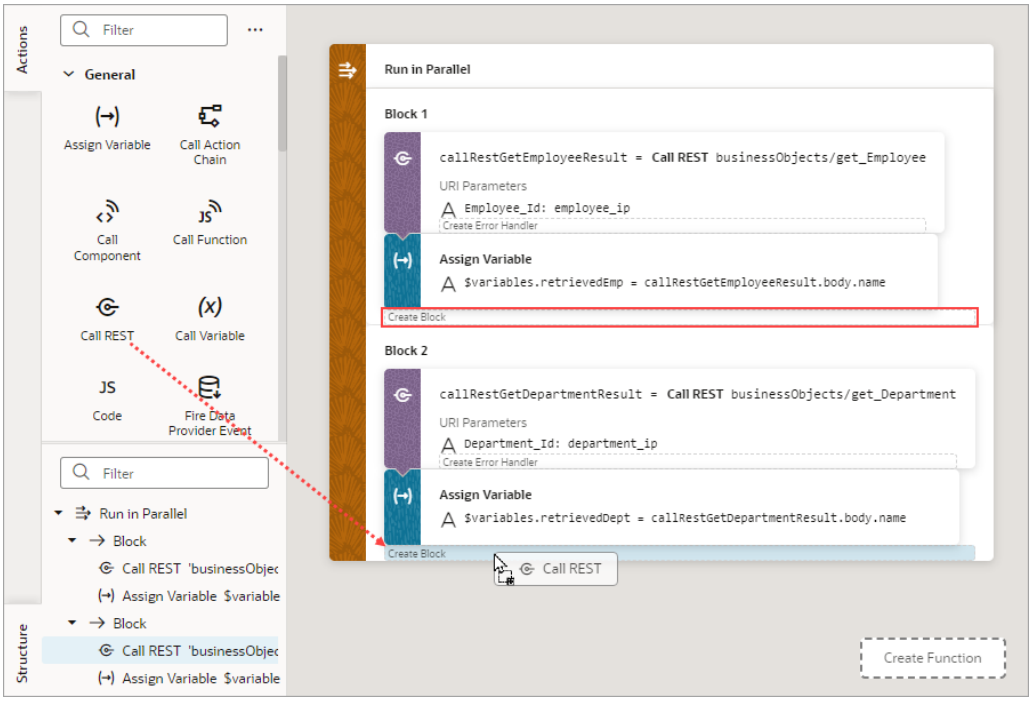
Use 1: Run Multiple Action Chains in Parallel

To use a Run In Parallel action to just run multiple action chains in parallel:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. Drop the actions to run for each block in the **Add Actions** area of the Run in Parallel action. For example, you could make two REST calls and assignments in parallel:



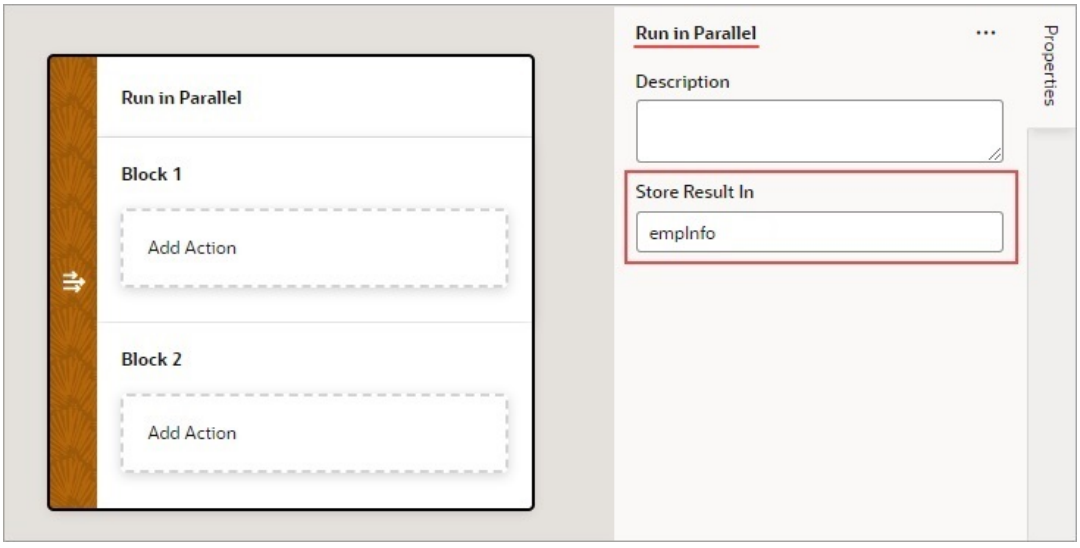
3. To add another block of code to run in parallel, you have two options:
   - Right-click a block and select **Duplicate** from the context menu.
   - Drag the first action for the block from the Action's palette onto a **Create Block** area that appears at the bottom of the action's blocks:



Use 2: Run Multiple Action Chains in Parallel to Produce a Combined Result

To use a Run In Parallel action to produce a combined outcome from the results of multiple action chains:

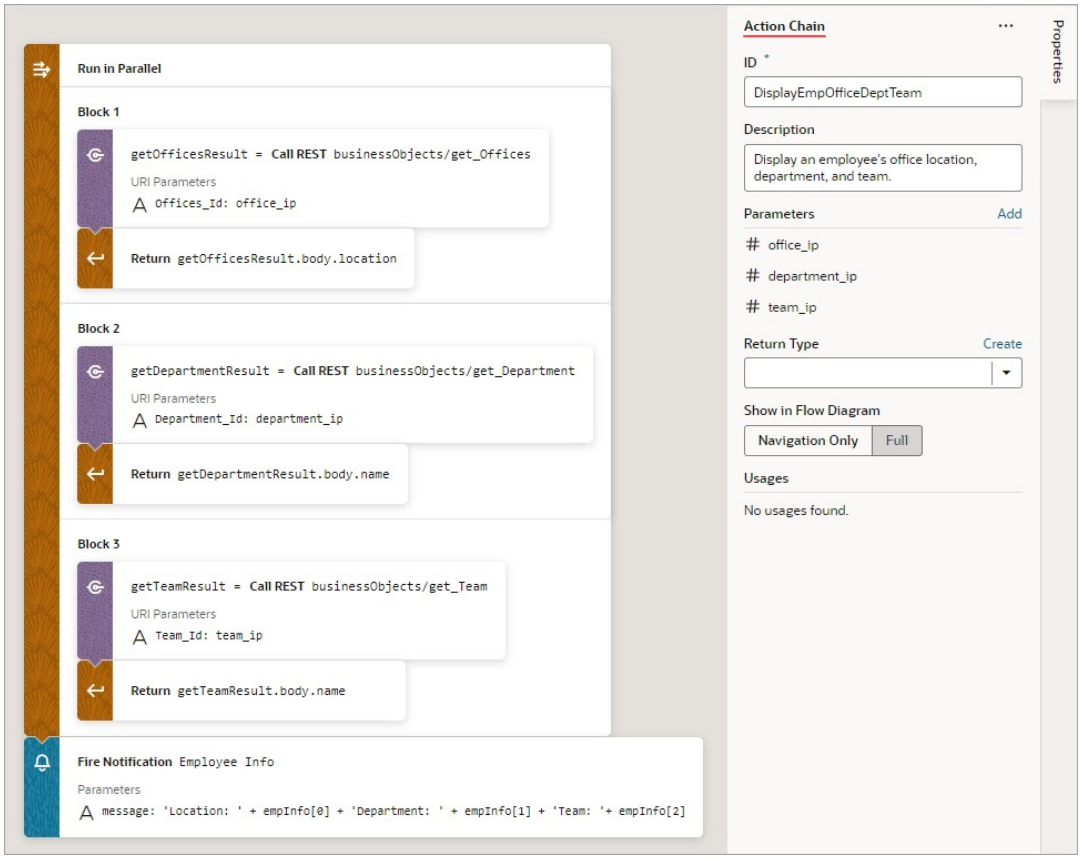
1. Add the **Run In Parallel** action in one of three ways, as explained at the end of Built-In Actions.
2. For the **Store Result In** property, provide a name for the array that will hold the result from each block. The first block's result is stored at index 0, the second block's result is stored at index 1, and so on.



3. Drop the actions to run for each block in the **Add Action** area of the Run in Parallel action.
4. To add another block of code to run in parallel, you have two options:

- Right-click a block and select **Duplicate** from the context menu.
- Drag the first action for the block from the Action's palette onto a **Create Block** area that appears at the bottom of a block when you drag an action over it.

5. Drop a Return action at the end of each block to return its result in the array that was named using the action's Store Result In property:



An array is returned by the Run in Parallel action (`empInfo` for this example, set in step 2): the first element contains the first block's result, the second contains the second block's result, and the third contains the third block's result.

Add a Scan Barcode Action

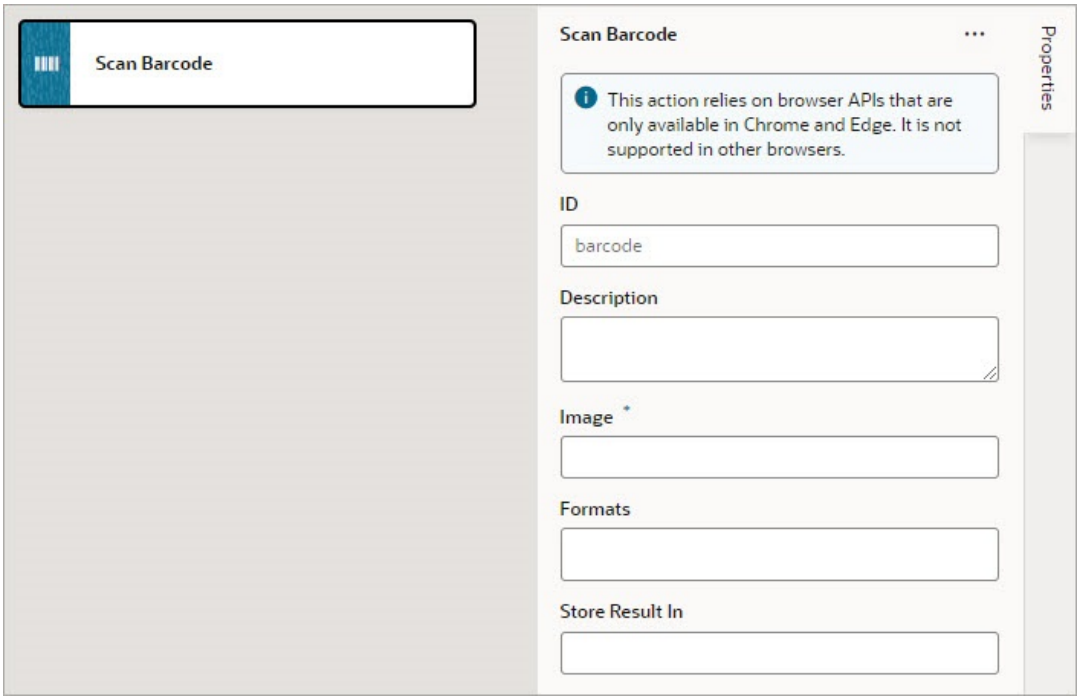
You can add the Scan Barcode action when you want your application to decode information such as URLs, Wi-Fi connections, and contact details from QR codes and barcodes.

For API information about this action, see Scan Barcode in the *Oracle Visual Builder Page Model Reference*.

> **Note:**
>
> The Scan Barcode action relies on the Shape Detection API for browsers and is only supported by these operating systems: Operating system support.

To use a scan barcode action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.



2. Specify the action's properties in the Properties pane:
   a. Update the **ID** field to make the action more identifiable.
   b. In the **Image** field, enter an image object (either a CanvasImageSource, Blob, ImageData, or an `<img>` element) to decode.

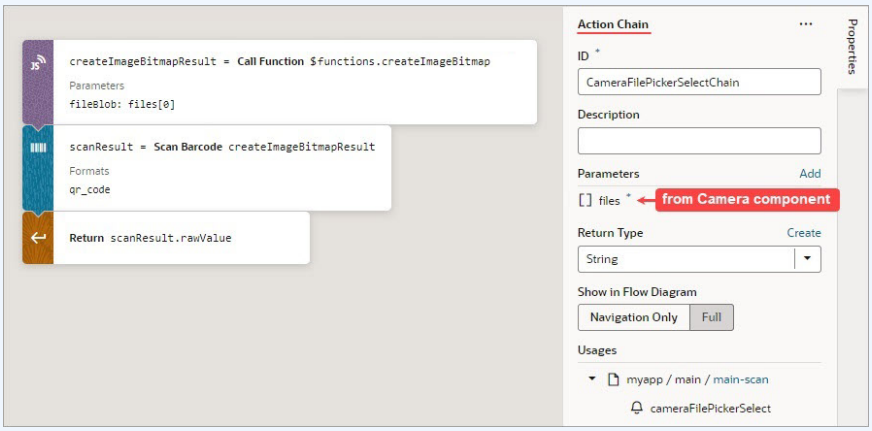
Note:

If you're using the camera component to pass a Blob to the Scan Barcode action, you might run into the `Failed to execute 'detect' on 'BarcodeDetector` error. To get around this error, convert the Blob to an ImageBitmap before passing it to the Scan Barcode action. For example:

i. Add a module function to do the image conversion, something like:

```
  // Convert Blob to ImageBitmap
  //
 PageModule.prototype.createImageBitmap =
function(fileBlob) {
    return window.createImageBitmap(fileBlob);
  };
```

ii. Add a Call Function action to the action chain, similar to:



iii. Pass the converted ImageBitmap as the **Image** property for the Scan Barcode action.

c. Optional: For the **Formats** property, select the barcode formats you want the browser to search for.

Barcode formats unlock a variety of use cases. QR codes can be used for online payments, web navigation, or social media connections, aztec codes can be used to scan boarding passes, and shopping apps can use EAN or UPC barcodes to compare prices of physical items.

If `Formats` is not specified, the browser will search all supported formats, so limiting the search to a particular subset of supported formats may provide better performance.

On success, a DetectedBarcode object is returned using the auto-generated variable shown by the Store Result In property. If the browser does not support the Shape Detection API or if a specified format is not supported, an exception is thrown.

Add a Share Action

You add a Share action to share content with other applications, such as Facebook, Twitter, Slack, and SMS, by invoking the native sharing capabilities of the host platform. This action

requires the user's consent, and as a best practice, consent should only be sought on a relevant user action.

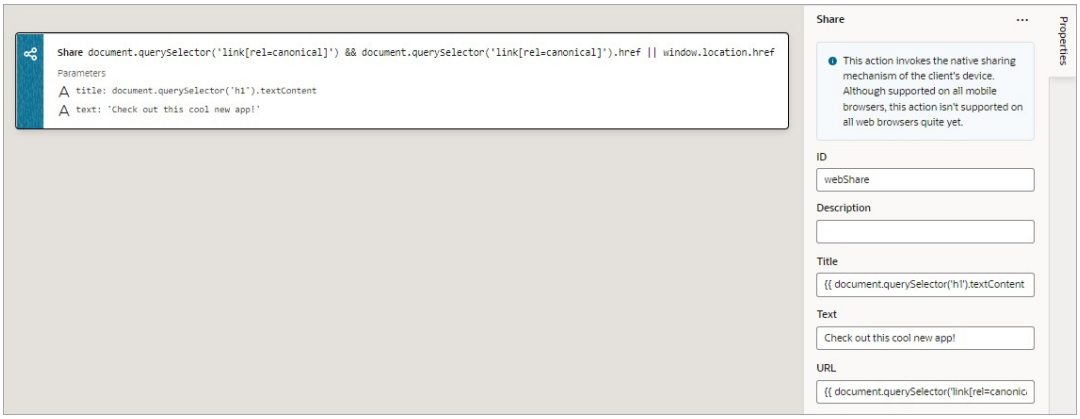
For API information about this action, see Share in the *Oracle Visual Builder Page Model Reference*.

> **Note:**
>
> Web apps require the web browser running the app to support the Share action. Currently, not all browsers support this native feature.

To use a Share action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. Update the **ID** field in the Properties pane to make the action more identifiable.
3. Configure the **Title**, **Text**, and **URL**. All properties are individually optional, but at least one property must be specified. Any URL can be shared, not just those under the website's current scope. Text can be shared with or without a URL.
   a. In the **Title** field, enter the title of the document to be shared.
   b. In the **Text** field, enter the text that will form the body of the message being shared.
   c. In the **URL** field, enter the URL that refers to the resource being shared.

   Here's an example that shares the current page's title and URL:



Add a String Operation Action

Index Of

You use an Index Of action to search a string for a substring. If the substring is found, the index where the substring starts is returned; otherwise, -1 is returned.

After you add the action, use the Variable property to select the string. For the Search String parameter that appears, enter the substring to search for. Optionally, specify the starting index for the search in the Position parameter. The copy of the array is returned in the variable shown by the Store Result In property.

Split

You use a Split action to divide a string into substrings by specifying separator character(s). The substrings are returned in an array without the separator character(s). For example, if you provide a string with several words, and provide the space character as the separator, an array is returned containing the words between the spaces.

After you add the action, select the string using the Variable property. For the Separator parameter that appears, enter the character(s) to use as the separator. Optionally, enter the maximum number of returned substrings using the Limit parameter.

Substring

You use the Substring action to return a substring by specifying its start and end indexes.

After you add the action, select the string using the Variable property. Enter the start and end indexes in the Start Index and End Index parameters. The resulting substring is stored in the variable shown by the Store Result In property.

Trim

You use a Trim action to return a copy of a string with blank spaces removed from both ends.

After you add the action, select the string using the Variable property. The trimmed string is stored in the variable shown by the Store Result In property.

Add a Switch Action

You add a Switch action when you want to match a value against a set of values, in order to execute appropriate actions for that case.

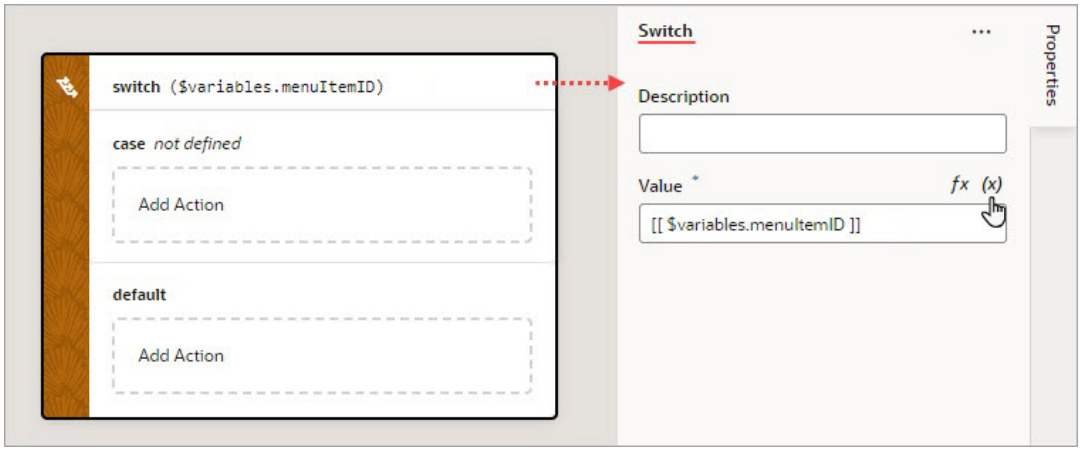
For API information about this action, see Switch in the *Oracle Visual Builder Page Model Reference*.

To use a Switch action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.

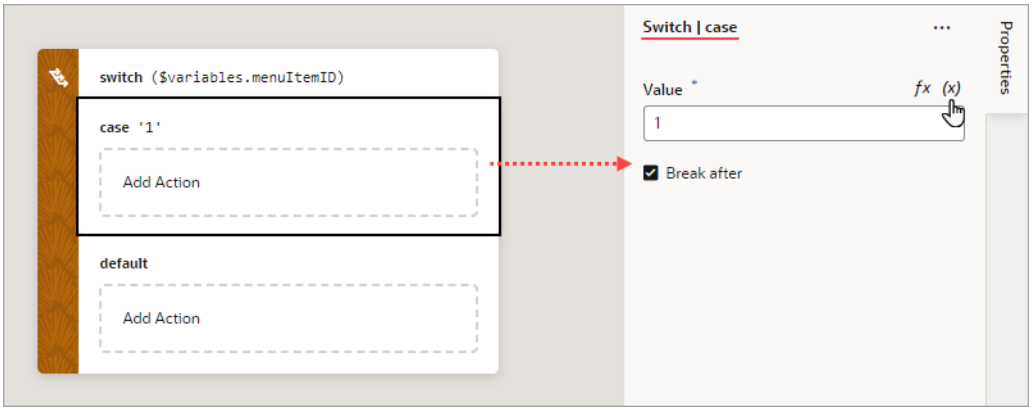
2. For **Value**, hover over the far-right side of the property and click to select the variable that is to be compared against each case value. If the value doesn't match any case value, or if it's null or undefined, the default case is executed.

   For example, when a menu defines a set of options, you can use the Switch action to determine which option was selected and to perform actions for that option. In this case, the page variable `menuItemId` holds the selected menu option, and it's selected using the Switch action's **Value** property:

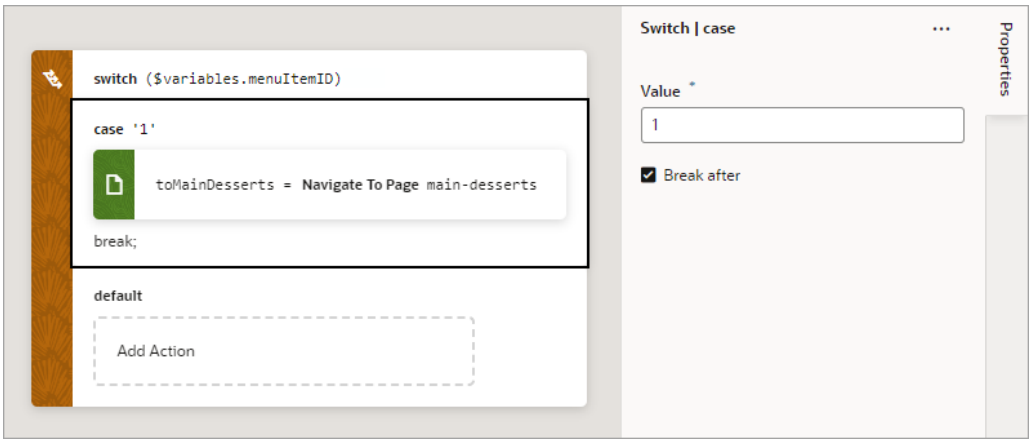


3. To define a case:

- Set a value for the case by selecting it within the Switch action, then using the **Value** property to enter its value or to select the variable that holds it. To select a variable, hover over the properties far-right side and click .



- Drop the actions to execute for the case in the **Add Action** area of the case block and configure their properties in the Properties pane:



- If the rest of the cases are to be ignored, select **Break after**. If not selected, the rest of the cases get executed, regardless of their values, until running into a break statement (if at all).

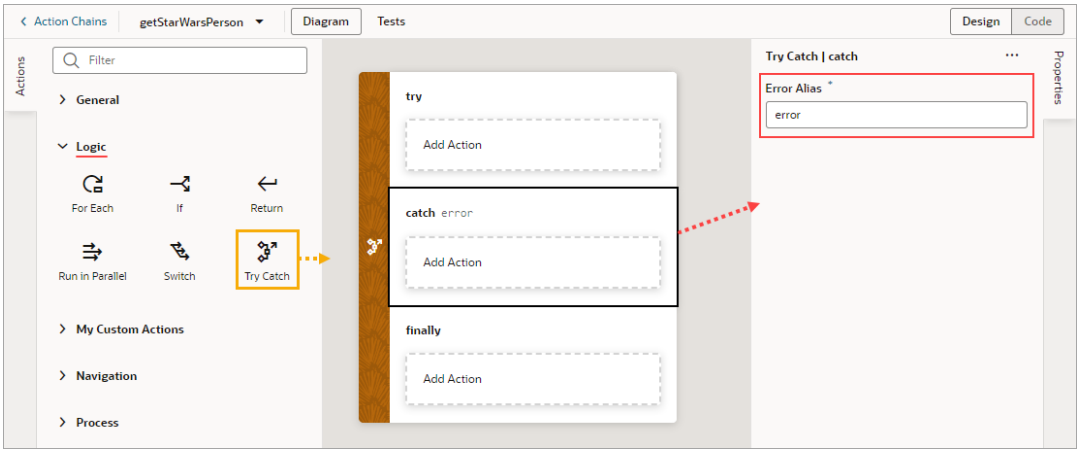
4. To add a new case block, right-click a case and select **Duplicate**.

Add a Try-Catch Action

You add a Try-Catch action to gracefully handle errors and avoid program crashes.

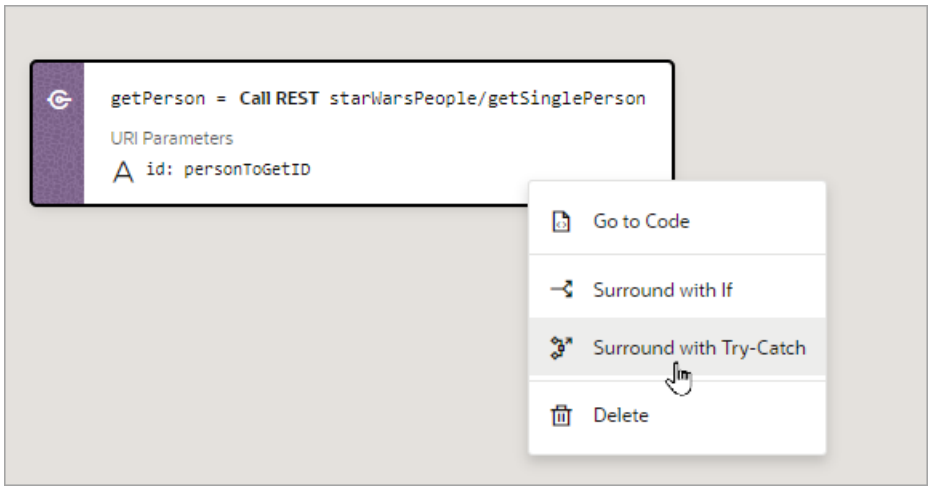
To use a Try-Catch action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.
2. To change the alias for the error object, which has the `name` and `message` properties, select the Catch block on the canvas and change the alias in the Properties pane:

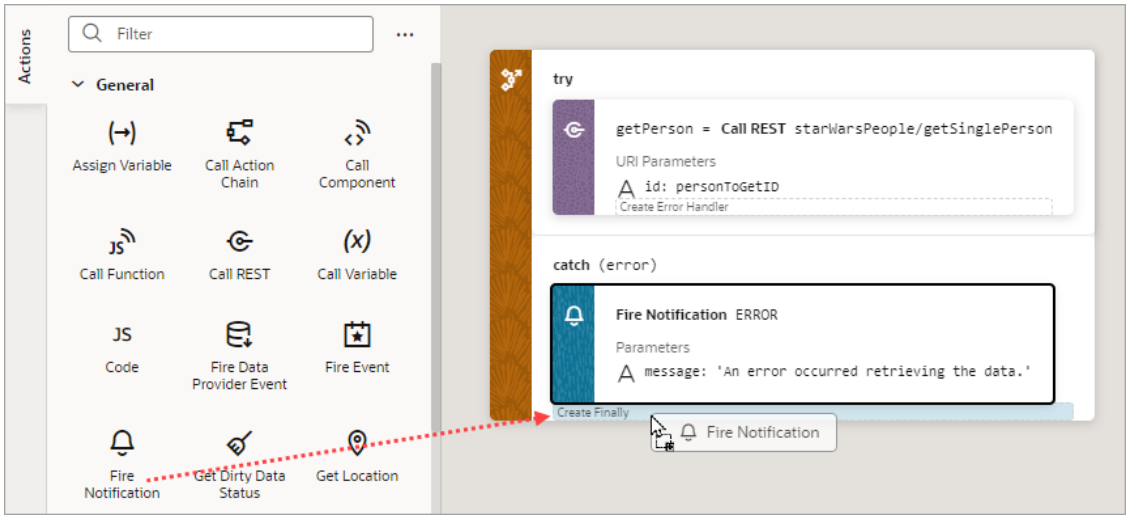


3. To add an action to a Try-Catch block, select the block and double-click the action in the Actions palette, or drag an action from the Action's palette onto the desired block:

You can also surround an action with a Try-Catch action by right-clicking it and selecting **Surround with Try-Catch**:



If you need a Finally block, drag the first action that you want to add to the block over the Try-Catch action and drop it into the **Create Finally** area that appears:



Custom Actions

In addition to the built-in actions you see in the Actions palette, you can create your own actions, using JavaScript, and use them in action chains just the way you'd use built-in actions.

Custom actions are created within the context of a specific application, and can't be shared across apps.

Create a Custom Action

To create a custom action, you provide its metadata in a JSON file and its code in a JavaScript file. The metadata contains basic details about the action, any input parameters needed by its implementation method, and optionally, an object for returning values.

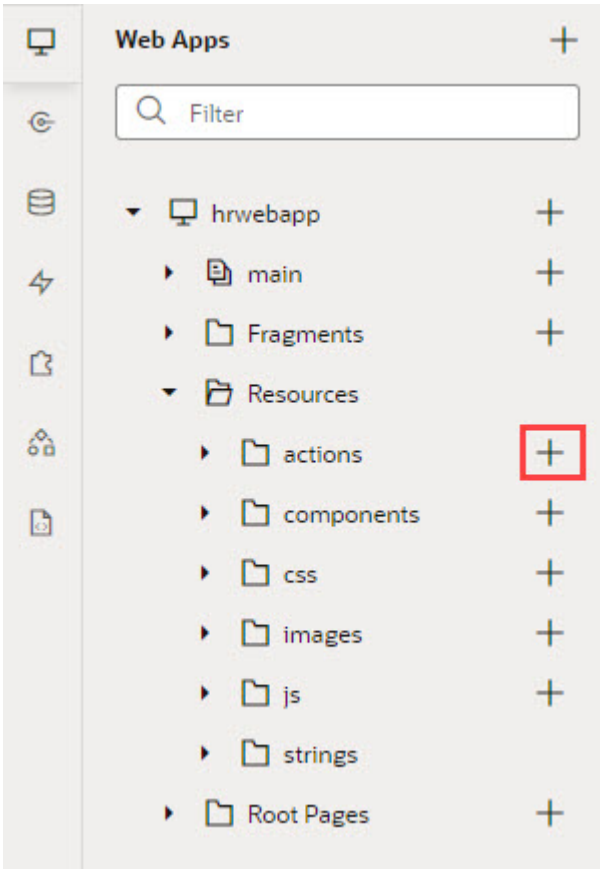
Here's an overview of what's required to create a custom action:

1. Create the Action Files (`action.json` and `action.js`):
   - **action.json**: Contains the metadata for the custom action. Used to define input parameters and to define an object for returning values. This file is also used by the Designer to add the action to the Actions palette, and to display the action's properties in the Properties pane.
   - **action.js**: Contains the code used to implement the custom action.
2. Add the Metadata to `action.json`.
3. Add the Code for the custom action to `action.js`.

Create the Action Files

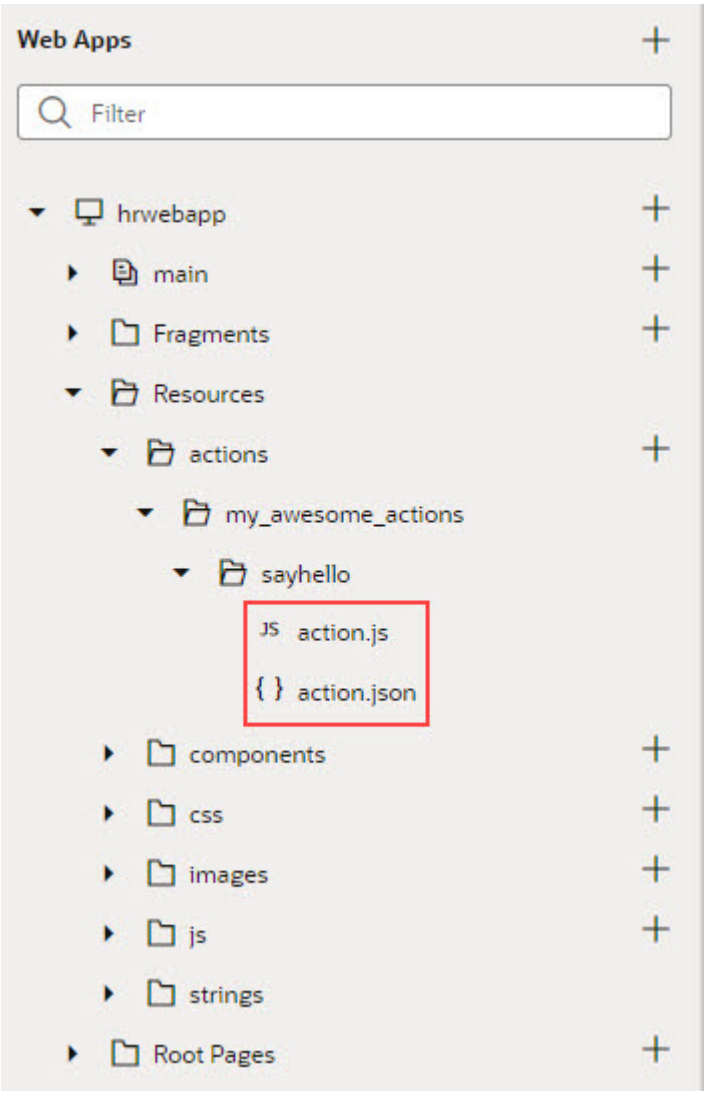
To create the template `action.js` and `action.json` files, for you to start with:

1. Select the **Web Apps** tab, expand the Application node (first node) and the **Resources** node under it, then click the Create Custom Action icon (+) next to the **actions** node:



2. In the **ID** field, enter the name of the action group folder for your new custom action, followed by a forward slash and the name of your new action, as in: `<action-group>/<action-name>`
   For example, you might enter `my_awesome_actions/sayhello`, where `my_awesome_actions` is the name of your group folder, and `sayhello` is the name of your action.

   The two newly created `action.js` and `action.json` templates are stored under the path `resources/actions/<action-group>/<action-name>/`, as shown here:

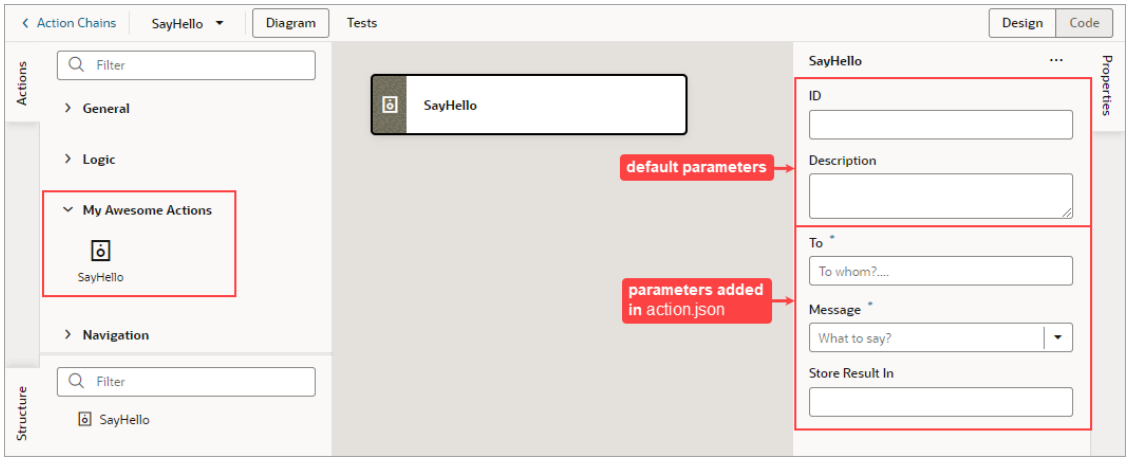


Add the Metadata

You provide the custom action's metadata in the `action.json` file, which includes:

- The action's basic details (ID, display name, icon…)
- An object used to return values from the action's implementation method *(optional)*
- Input parameters needed by the action's implementation method *(optional)*

The Action Chain editor uses the metadata to add the custom action to the Actions palette and to display its parameters in the Properties pane:



When you created the `action.json` file, the default parameters, ID and Description were automatically created for you, but their metadata isn't added to `action.json`. You only add the metadata for the input parameters and the return object that you want to add to the custom action.

Here's an example of the `action.json` file for the sample `sayHello` custom action, with a breakdown of its parts:

- The first set of properties provide the basic details about the custom action.

- The `resultShape` property provides the definition for the object returned by the action's implementation method, which can be used as input for another action when creating an action chain. For instance, the string returned by this action can be an input for an action that writes the string to a log file.
- The `propertyInspector` property defines the action's input parameters.

Define the Custom Action's Properties

Use this table to help you define the properties for your custom action in the `action.json` file.

Property	Required	Description
"id": "",	Yes	Unique ID for custom action.
"category": "My Category",	No	Category to contain custom action in Actions palette. If not specified, action is placed under the default category for custom actions, *Custom*.
"defaultParameters": { },	No	If input parameters are defined and they need default values, use this property to specify default values for them by specifying the input parameter names and their values (name-value pairs). Defaults are assigned when action is first added to an action chain.
"description": "",	No	Brief description of custom action.
"displayName": "",	Yes	Name to display in Actions palette.
"helpDescription": "",	No	Help text to appear for action when user hovers over action's title in Properties pane and clicks the question mark icon.
"iconClass": "",	Yes	Icon to display for action in Actions palette.
"referenceable": "self | extension"	No	Indicates if action is available in extensions; default is `self`, indicating it isn't available in extensions.
"idPrefix": "",	No	Used to auto-generate action IDs for actions when they are added to action chains. When action is added to an action chain, action's ID field in the Properties pane is auto-populated. If specified, ID field is populated using this property's value, otherwise, the action's name is used.
"resultShape": { "someName": "string" },	No	If one or more values are to be returned by the implementation method, use this property to define the object to return them.
"showInDiagram": "on" | "off"	No	If set to `on`, action is available on Actions palette of flow diagram.

Property	Required	Description
"tests": { "requiresMock": "on" | "off" }	No	Setting for action's mock requirements for action chain tests: **on:** Indicates action needs to be mocked. **off:** Indicates Visual Builder provides suggestions for expected action results if `resultShape` parameter is specified in `action.json`. If action has no `resultShape`, suggestions are enabled for the action's input parameters specified in propertyInspector section. Default is `off`.
"propertyInspector": [{}]	No	Metadata for the input parameters needed by implementation method. Input parameters are displayed in Properties pane of Actions editor.

Define Input Parameters for a Custom Action

Use this table to help you define input parameters for your custom action. In the `action.json` file, use the `propertyInspector` property to define the parameters you need:.

Property	Required	Description
"name": "",	Yes	Name of input parameter.
"help": "",	No	Help text to appear for input parameter when user hovers over parameter's title in Properties pane and clicks the question mark icon.
"label": "",	Yes	Label to display for input parameter in Properties pane.
"placeholder": "",	No	Hint text to display for input parameter in Properties pane.
"required": true | false,	No	Indicates if a value is required for input parameter.
"options": [{}],	No	If `component` property for input parameter is set to `comboboxOne` or `comboboxMany`, use this property to specify all of the values to be available for the combobox.
"type": "",	Yes	Parameter's data type, which can be number, string, boolean or object; if the type is not specified, values are stored as strings.
"component: "inputText | textArea | comboboxOne | comboboxMany"	Yes	Indicates if parameter is a text field, text box or a combobox with single or multiple selections.

Add the Code

You provide the code for your custom action using the `action.js` template file that was created for you when you first created the new action.

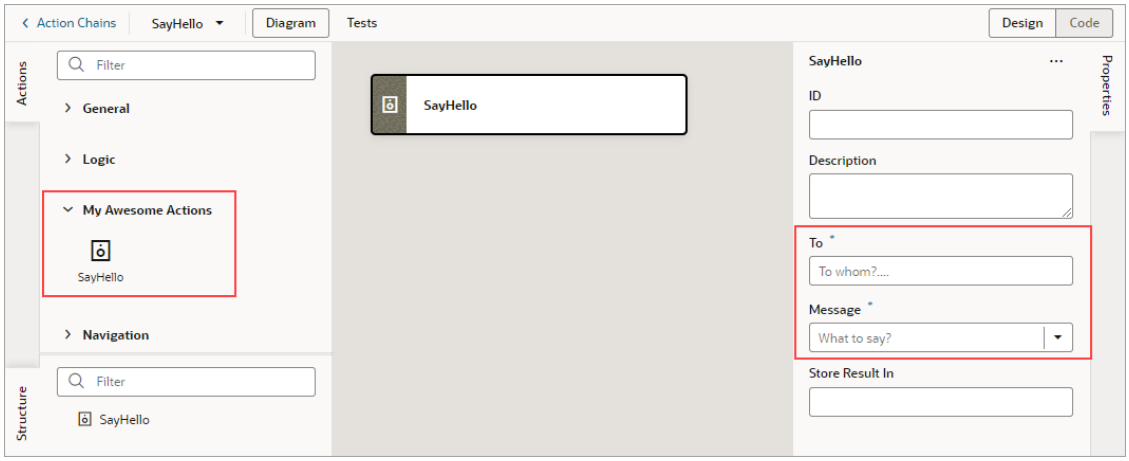
To provide the code for your action, use the `perform()` method, which receives the input parameter, `parameters`. The input parameter contains the values for the action's input parameters, as entered in the Properties pane.

How Are Input Parameters Passed?

Here's an example of the `perform()` method, in `action.js`, that implements the sample `sayHello` custom action by using the `alert()` method:

```
define(['vb/action/action'], (Action) => {
            'use strict';
            class CustomAction extends Action {
               perform(parameters) {
                  const to = parameters.to;
                  const message = parameters.message;

                  alert(`${message}, ${to}`);

                  if (to && message ) {
                     return Action.createSuccessOutcome({result: to + ", " +
message});
                  }
                  return Action.createFailureOutcome({result: "Something
wrong, unable to SayHello"});
               }
            }
            return CustomAction;
         });
```

The values you enter for the action's input parameters, To and Message, in the Properties pane, are passed to the `perform()` method using the input parameter, `parameters` (shown in the code above).



How are Values Returned?

If you need one or more values returned by the method that implements the custom action, you need to define the object that returns them in `action.json`. You do so by using the `resultShape` property.

To return values from the implementation method, you use the `createSuccessOutcome()` method of the `Action` class.

As an example, here's the declaration of the return object for the sample `sayHello` custom action:

```
"resultShape": { "result": "string" },
```

Specify Path to Code

Lastly, you need to tell Visual Builder where the custom action's code file is stored, by adding a `requirejs` property to the `app-flow.json` file. Here are the steps:

1. In the Navigator, on the **Web Apps** tab, select the web app, then select the **JSON** tab to open the `app-flow.json` file.
2. Specify the path to the custom action's code file by adding a `requirejs` property in `app-flow.json` and naming the path, using this format:

```
"<custom-action-ID>": "resources/actions/<action_group>/<action_name>"
```

In this example, `<custom-action-ID>` is the ID for the custom action (case sensitive), as specified in `action.json`.

Here's an example of specifying the path:

```
"requirejs": {
   "paths": {
      "demo/SayHello": "resources/actions/my_awesome_actions/sayhello/action"
    }
},
```

Start an Action Chain

You set up an action chain to be triggered when an event occurs in an artifact. The type of event available depends on the artifact. For example, you can trigger an action chain to start when a lifecycle event such as `vbEnter` is fired to load a page. Or, use the `onValueChanged` variable event when a variable's value changes. You can also use custom events to start an action chain from another action chain.

Start an Action Chain From a Component

When you add a component to a page or layout, you'll need to create a component event and component event listener if you want it to trigger some behavior (for example, to open a URL). The suggested option in the component's Properties pane creates these for you.

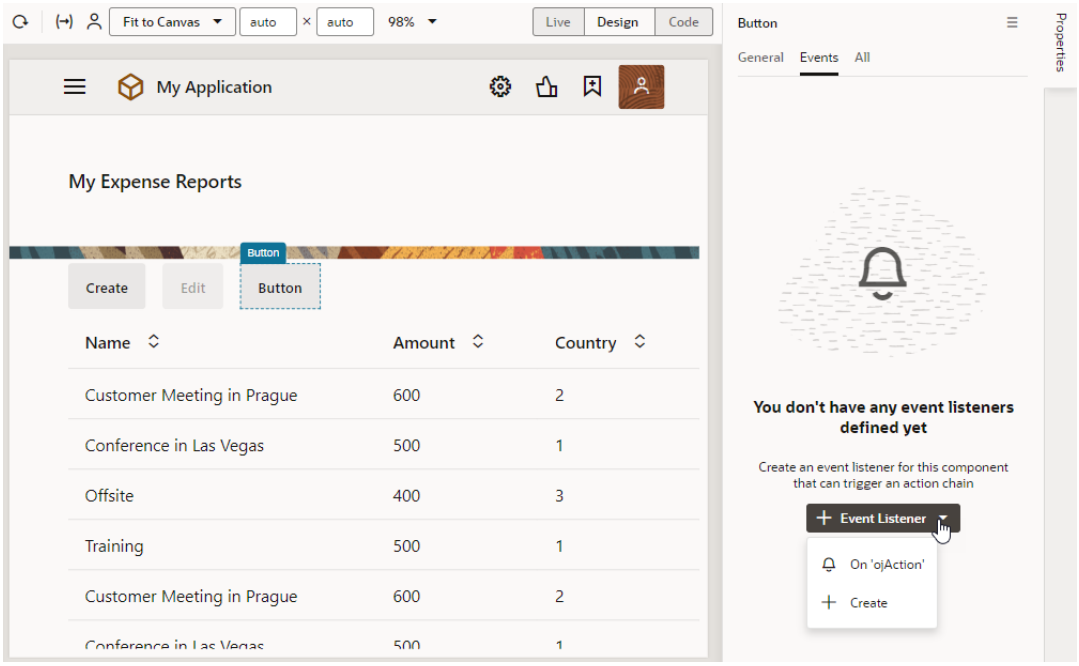
There are various predefined events that you can apply to a component, and the events available are usually determined by the component. For example, the `ojAction` event is triggered when a button is clicked, so you would typically apply it to a button component (you couldn't apply it to a text field component). Each button will have a unique event and an event listener listening for the button's `ojAction` event, and the listener would start an action chain (or multiple action chains) when the event occurs. Each component event will usually have a corresponding component event listener.

> **Note:**
>
> You can add an event to a component only from the component's Properties pane. You can't create one in the Events tab of pages.

To start an action with a component:

1. Select the component in a page or layout.

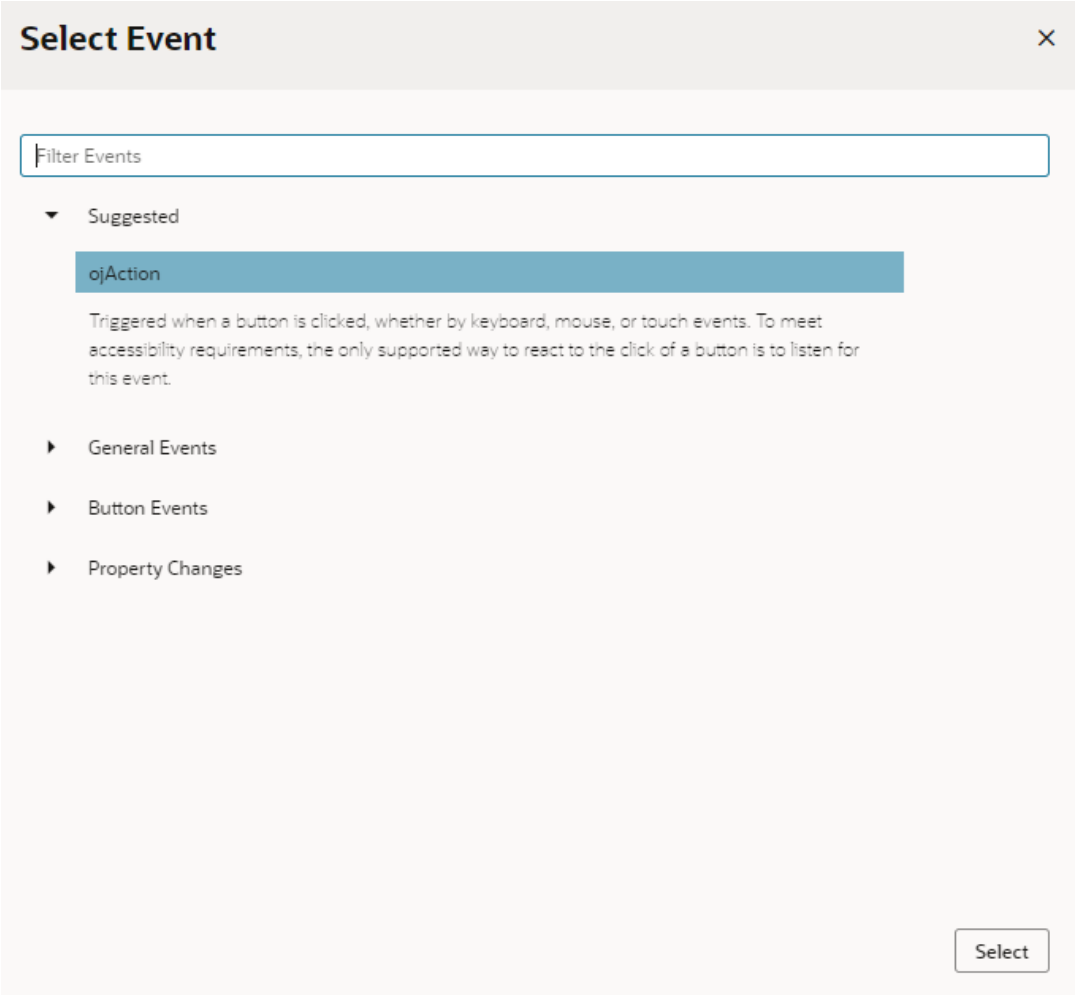
   Typically, you assign events to elements such as buttons, menus, and fields in form components. You can select the component on the canvas, in the Structure view, or in Code view.



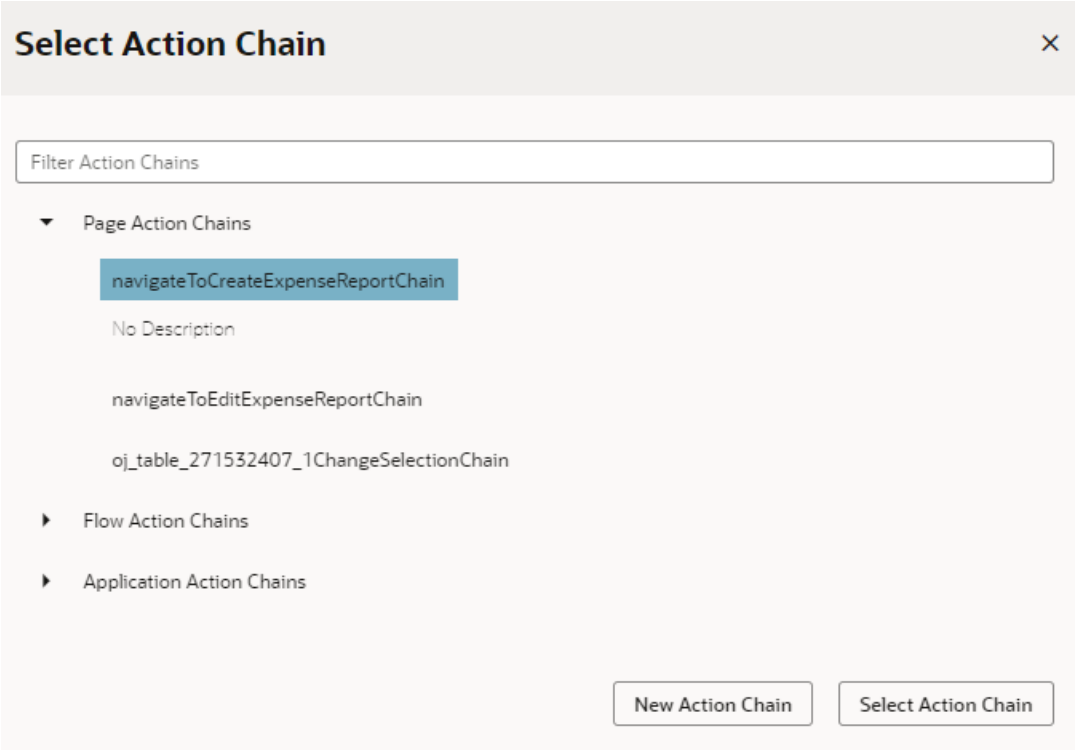
2. In the component's Events tab in the Properties pane, click **+ Event Listener**. You can choose the suggested event as a quick start or you can create a custom event to use a different event.

   When you add the new event using the quick start, an action chain is created for you and the Action Chain editor opens automatically. When you add the new event using the custom option, you'll need to select an event.

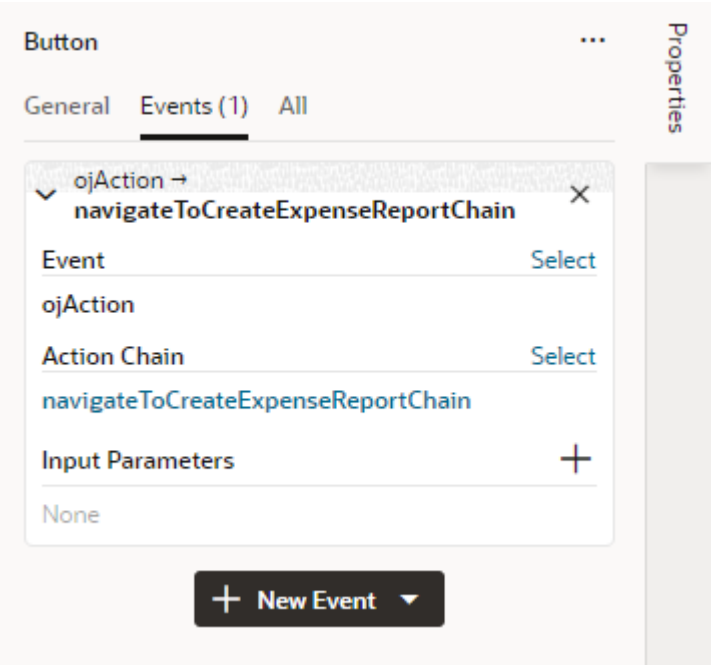
3. For a custom event, select the event you want to use to trigger an action chain. Click **Select**.



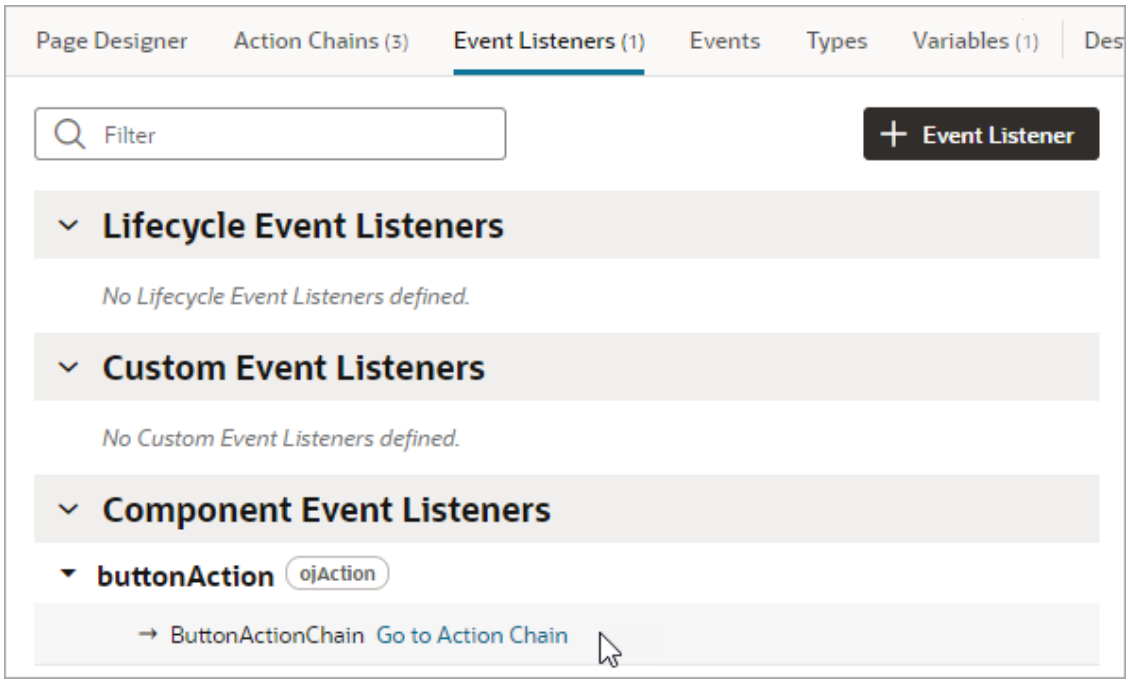
4. Select the action chain you want the event to trigger and click **Select Action Chain**. Alternatively, click **New Action Chain** to create a new action chain.



The Events tab in the Properties pane shows events on the component that Visual Builder responds to by triggering action chains. You can edit the properties, for example, to add input parameters that you want to use in the action chain. Input parameters can provide values from the component and its page to the action chain, which the action chain can then use to determine its behavior. For example, a table selection event could supply details of which row was selected to its action chain.



If you used the quick start option to add an event, a component event listener is created for the new event, and the listener is mapped to the action chain it created for you. If you open the Event Listeners tab, you'll see it listed under Component Event Listeners, along with the action chain that it will trigger.



Start an Action Chain When a Variable Changes

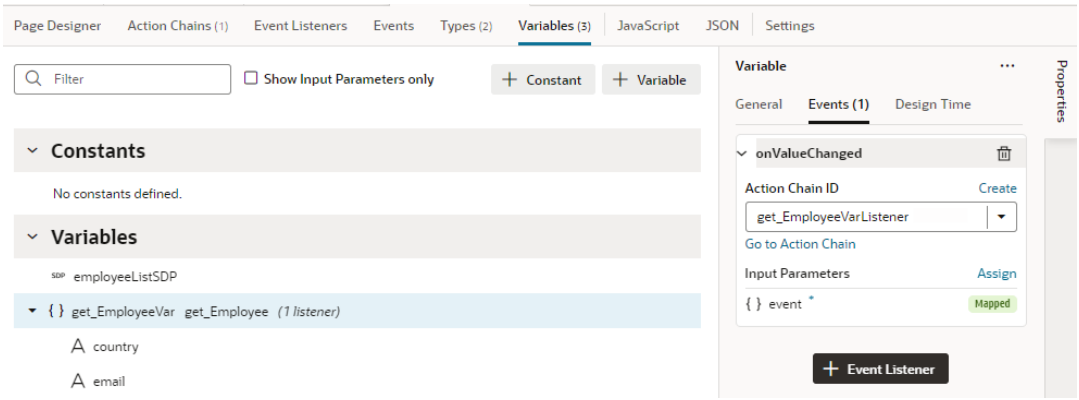
You can start an action chain when the value stored in a variable changes by adding an `onValueChanged` event to the variable.

When you use an `onValueChanged` event to trigger an action chain, the trigger has the payload of the variable's old and new values. For example, let's say you changed the name property of an Employee and reset the Employee; the framework sends an event that the Employee changed, and as part of the payload indicate that the name has changed.

To start an action chain when the value of a variable changes:

1. Open an artifact's **Variables** editor.
2. Select the variable in the list, then click **Events** in the Properties pane.
3. Click **+ Event Listener**.
4. Select an action chain from the list and click **Select**.

   When you add the event to the variable, a listener that listens for the `onValueChanged` event on the variable is automatically created. The variable's Events tab in the Properties pane displays the action chain the event listener starts; you can change or remove the action chain, assign input parameters, and add more action chains.



> **Note:**
>
> Variable events and event listeners are not listed in an artifact's Events or Event Listeners tabs.

Start an Action Chain From a Lifecycle Event

Lifecycle events are predefined events that occur during a page's lifecycle. You can start action chains when these events occur by creating event listeners for them. For example, if you want to initialize some component variables when the page opens, you can create an event listener in your artifact that listens for the `vbEnter` event. You could then set the event listener to trigger an action chain that assigns values to the component's variables.

Before you create an event listener to trigger an action chain, it's important to understand a page's lifecycle, so you know where to plug in custom code to augment the page's lifecycle. Each page in your application has a defined lifecycle, which is simply a series of processing steps. These might involve initializing the page, initializing variables and types, rendering components, and so on.

Each stage of the lifecycle has events associated with it. You can "listen" for these events and start action chains whenever they occur to perform something based on your requirements. For example, to load data before a page loads, you can use the `vbEnter` event and start an action chain that calls a GET REST endpoint.

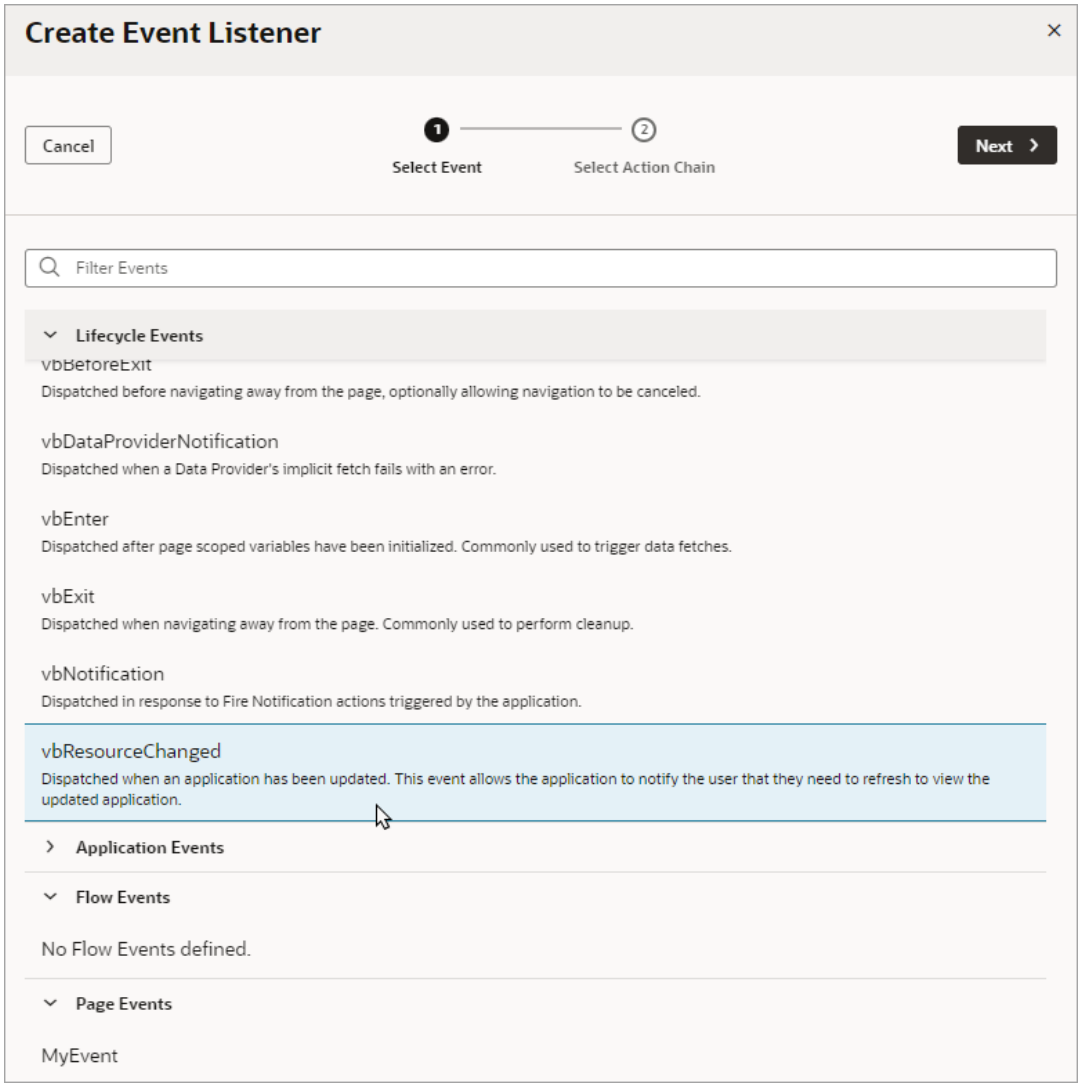
Keep in mind that one or more pages make a flow and each flow has its own lifecycle.

This table describes the lifecycle events you can use to start action chains:

Lifecycle Event	Description
`vbBeforeEnter`	Triggered before navigating to a page. Commonly used when a user does not have permission to access a page and to redirect the user to another page (for example, a login screen). Because this event is dispatched to a page before navigating to it, you can cancel navigation by returning an object with the property `cancelled` set to `true` (`{ cancelled: true }`). For this event, you can use these variable scopes to get data: • `$application`: All application variables can be used in the event's action chain • `$flow`: All parent flow variables can be used in the event's action chain • `$parameters`: All page input parameters from the URL can be used in the event's action chain
`vbEnter`	Triggered after container-scoped variables have been added and initialized with their default values, values from URL parameters, or persisted values, and is dispatched to all flows and pages in the current container hierarchy and the application. Commonly used to fetch data. For this event, you can use these variable scopes to get data: • `$application`: All application variables can be used in the event's action chain • `$flow`: All parent flow variables can be used in the event's action chain • `$page`: All page variables can be used in the event's action chain
`vbBeforeExit`	Triggered on all pages in the hierarchy before navigating away from a page. Commonly used to warn if a page has to be saved before the user leaves it, or to cancel navigation to a page (say, because a user doesn't have permissions to view that page) by returning an object with the property `{ cancelled: true }`.
`vbExit`	Triggered when navigating away from the page and is dispatched to all flows and pages in the current container hierarchy being exited from. Commonly used to perform cleanup before leaving a page, for example, to delete details of a user's session after logout.
`vbAfterNavigate`	Triggered after navigation to the page is complete and is dispatched to all pages and flows in the hierarchy and the application. The event's payload (`$event`) is an object with the following properties: • `currentPage <String>`: Path of the current page • `previousPage <String>`: Path of the previous page • `currentPageParams <Object>`: Current page parameters • `previousPageParams <Object>`: Previous page parameters
`vbNotification`	Triggered when a Fire Notification action is fired by the application.
`vbResourceChanged`	Triggered when an application has been updated. Commonly used to notify the user that they need to refresh to view the updated application.
`vbDataProviderNotification`	Triggered when a Data Provider's implicit fetch fails with an error.

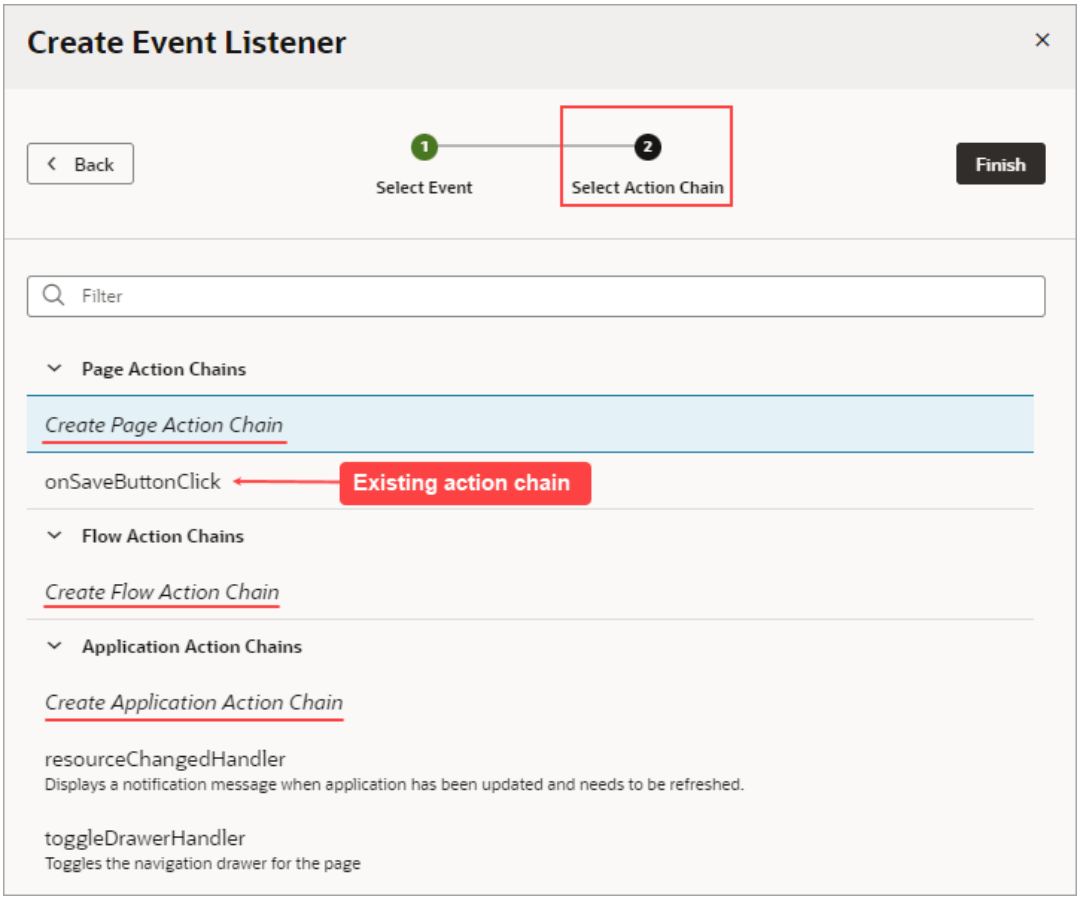
To start an action from a lifecycle event:

1. Open the Event Listeners tab for the page containing the event you want to trigger an action chain for.
2. Click **+ Event Listener**.
3. In the Create Event Listener wizard, expand the Lifecycle Events category and select the event you want to trigger an action chain for. Click **Next**.



4. Select the action chain you want to start. You can select any action chain that is scoped for the artifact. 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.

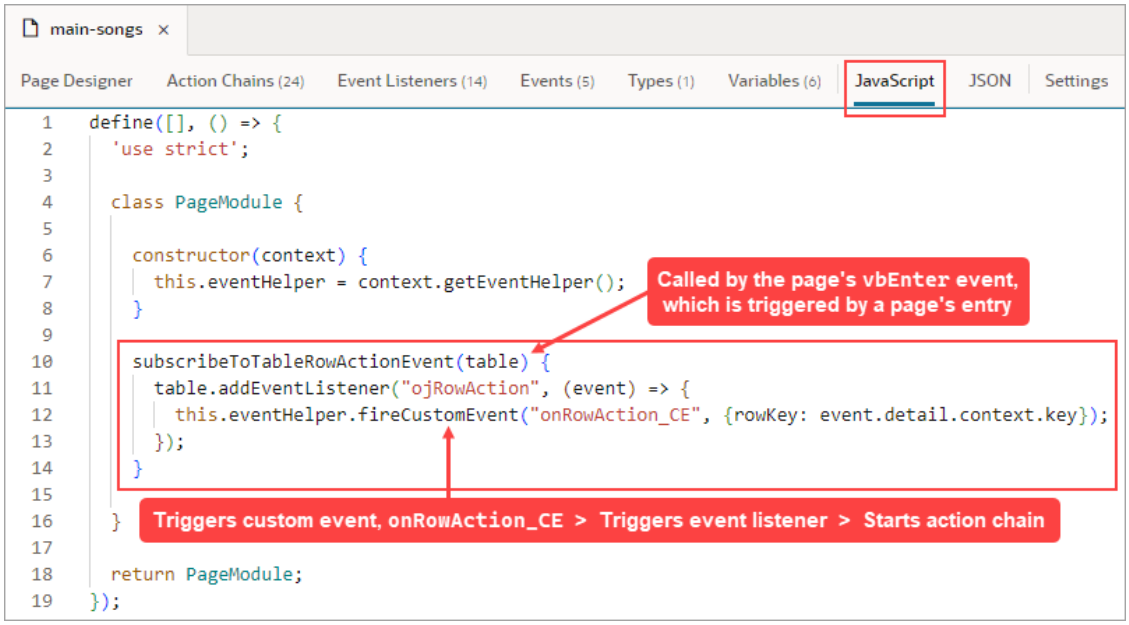
   If you want to create a new action chain, select the create action chain option at the appropriate level (page, flow, or application), then click **Finish**.



After you create an event listener, you can click **Add Action Chain** for the lifecycle event if you want it to start additional action chains.

Start an Action Chain By Firing a Custom Event

You can use the Fire Event action in an action chain to trigger a custom event, which in turn starts a different action chain to do things like display a notification, transform data, and so on. You could also trigger a custom event by using an event helper's `fireCustomEvent()` method (see Module Function Event Helper) in a module function (JavaScript function):

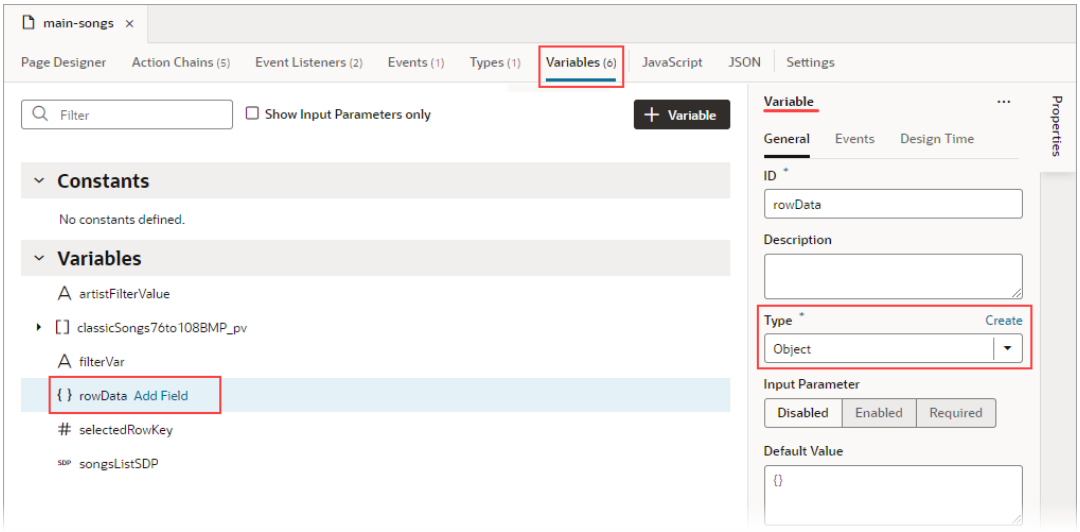


Each custom event has a Behavior property, which sets whether an action chain runs serially or in parallel. The default behavior is "Notify", which runs the action chain in parallel. For more about this property, see Choose How Custom Events Call Event Listeners.

After creating a custom event, you create an event listener for it to start one or more action chains.

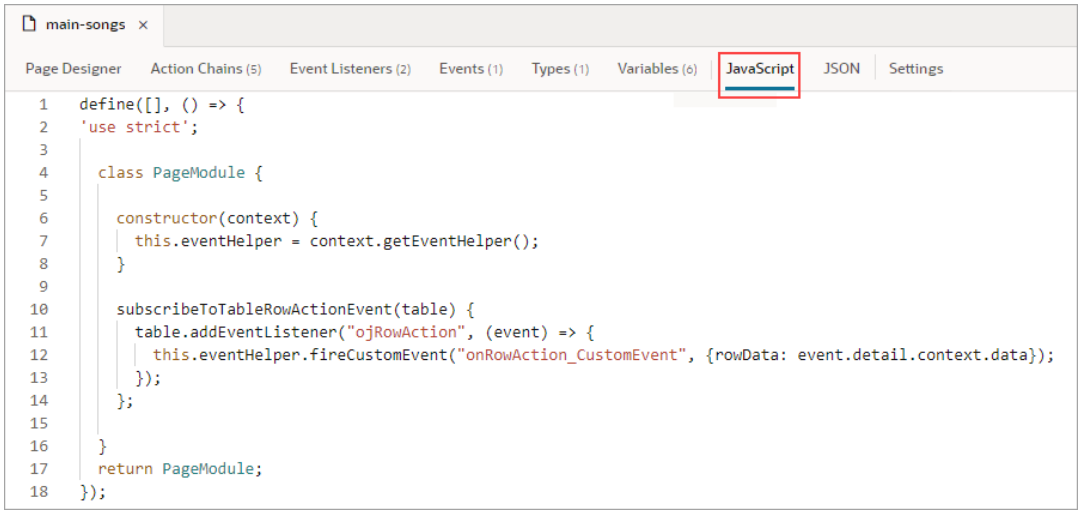
In this example, we'll use a module function to subscribe to a table's `ojRowAction` event to trigger a custom event that starts an action chain. The action chain then saves the selected row's data to a page variable. To begin:

1. Create a page variable of type Object to hold the selected row's data:



For this part, you create a JavaScript function (module function) to subscribe to the table's `ojRowAction` event, which is triggered when a user clicks a table's row. You'll use this event to trigger your custom event, which will start the action chain that saves the row's data to the `rowData` page variable.

2. To create the function, select the **JavaScript** tab. Use the context's `eventHelper` object's `fireCustomEvent()` method to trigger your custom event and to pass it the required payload. The `event` parameter contains the row's data (`event.detail.context.data`).

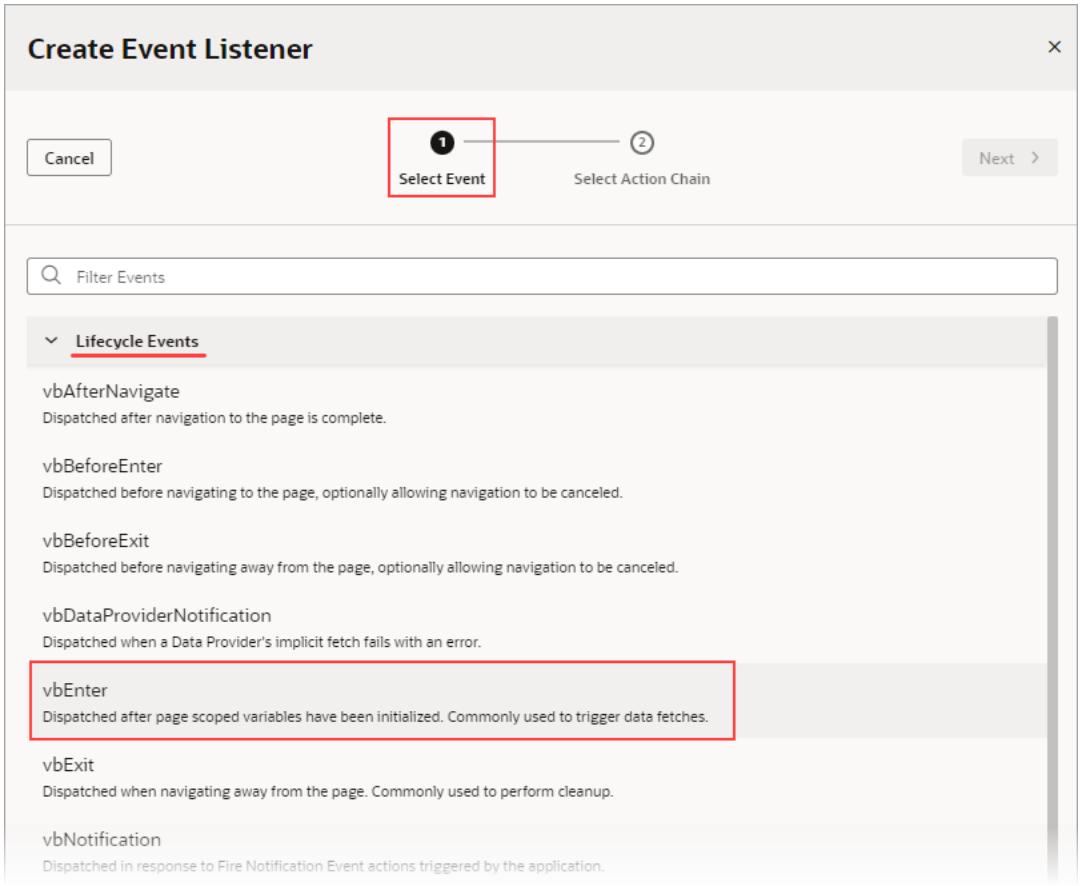


Here's the example code:

```
    constructor(context) {
      this.eventHelper = context.getEventHelper();
    }

    subscribeToTableRowActionEvent(table) {
      table.addEventListener("ojRowAction", (event) => {
        this.eventHelper.fireCustomEvent("onRowAction_CustomEvent",
{rowData: event.detail.context.data});
      });
    };
```

Next, you need to create the event listener for the page's `vbEnter` event, which is triggered when the page starts. You'll use this event listener to start an action chain that calls the function to subscribe to the table's `ojRowAction` event.

3. To create the event listener for the page's `vbEnter` event, select the **Event Listeners** tab, and click the **+ Event Listener** button to create it.
4. For the wizard's Select Event step, in the Lifecycle Events section, select `vbEnter` and click **Next**.

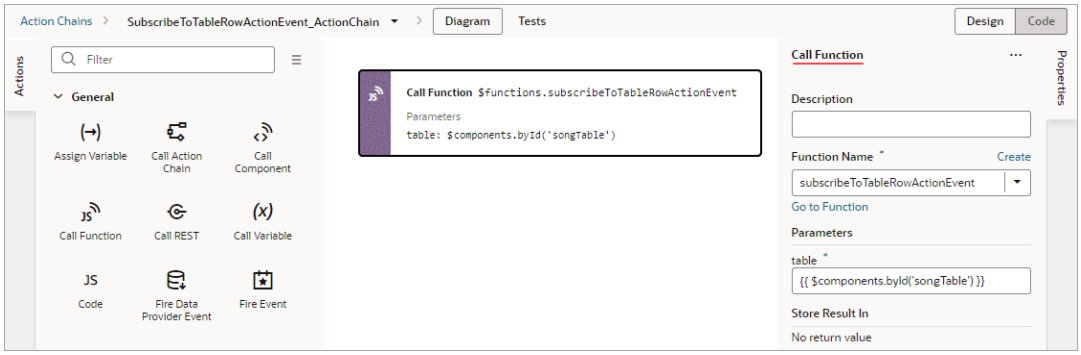


5. For the Select Action Chain step, click the Page Action Chains section's Add icon to create an action chain for the listener to initiate.

If the custom event for this listener has input parameters, which this one doesn't, the action chain would be created with an `event` input parameter that contains the custom event's input parameters.

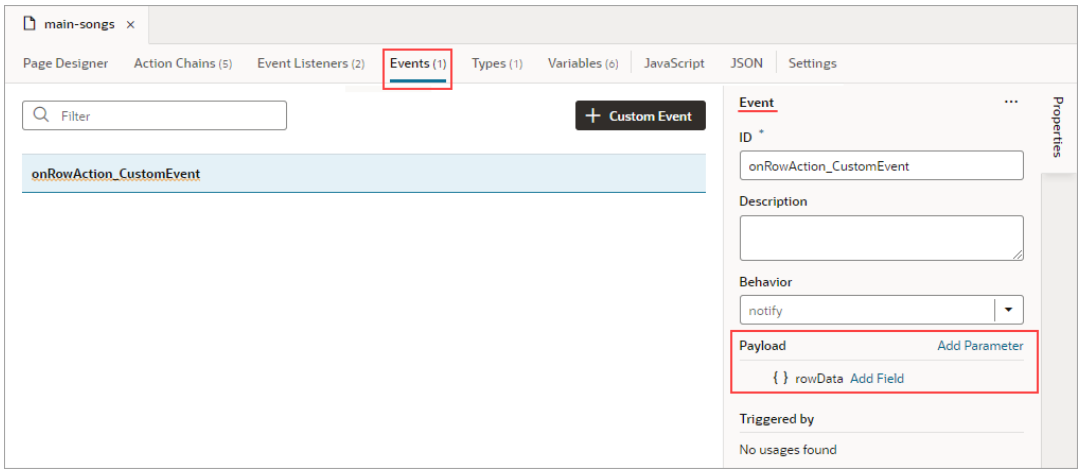
You are taken to the Action Chain editor, where you can create the action chain to call the function that subscribes to the table's `ojRowAction` event.

6. Add the Call Function action to the canvas. Set its `Function Name` property to the JavaScript function, and pass the table to the function using the `table` parameter:



For this next part, you'll create the custom event that will be triggered by the table's `ojRowAction` event. You'll also create the action chain that assigns the row's data to the `rowData` page variable.

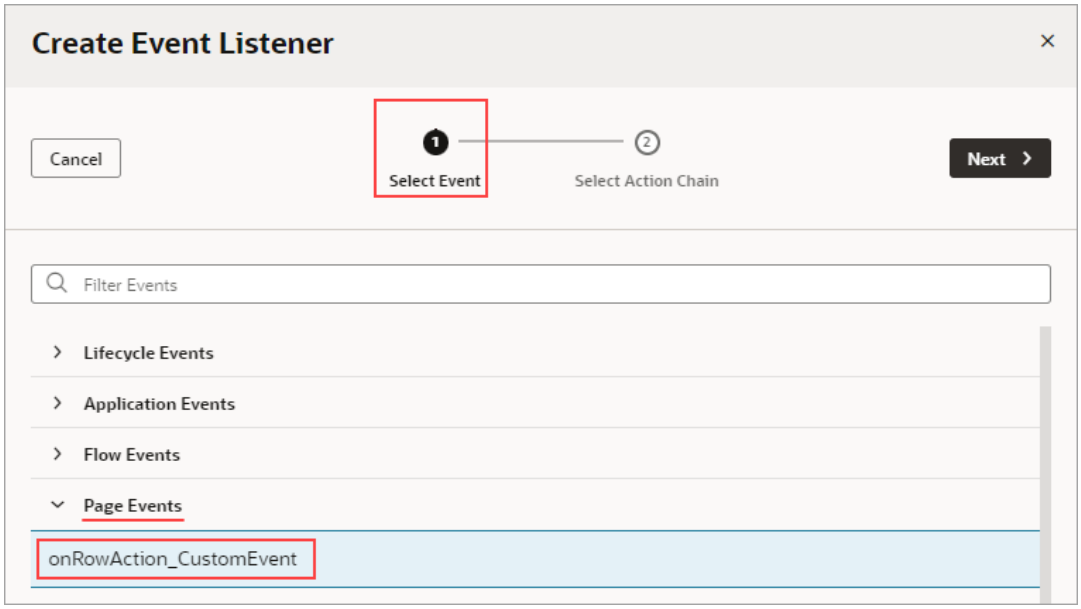
7. On the page's **Events** tab, click the **+ Custom Event** button to create a custom event. In the Properties pane, click the Payload property's **Add Parameter** link and define an input parameter of type Object. This input parameter will be used to pass the row's data to the action chain that assigns the data to the `rowData` page variable.



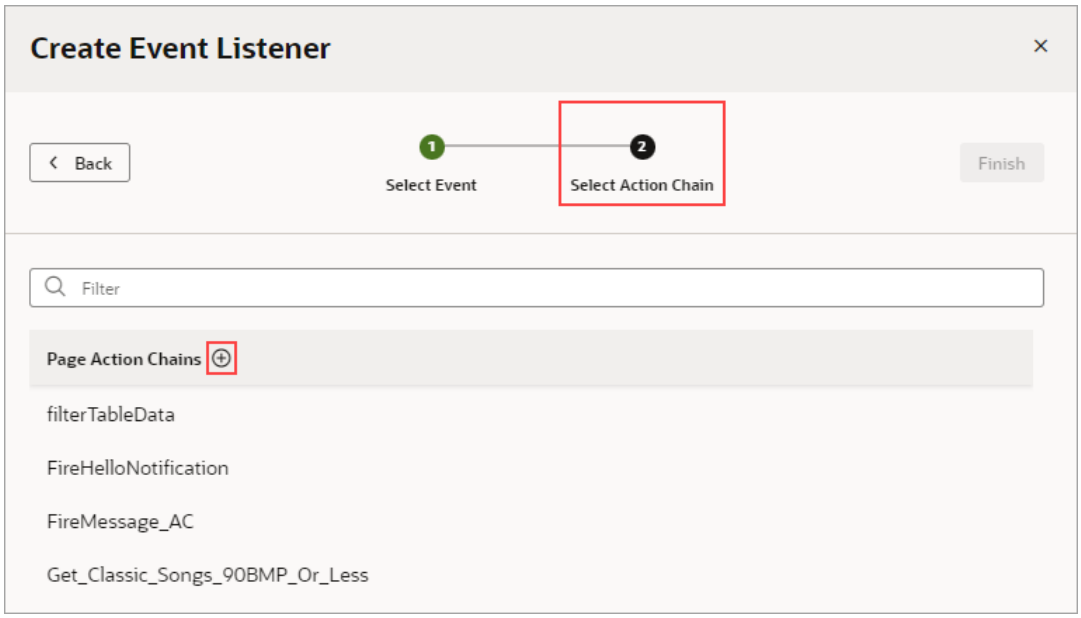
8. For the custom event's Behavior property, set whether the action chain runs serially or in parallel. The default, `notify`, is in parallel. For details about each option, see Choose How Custom Events Call Event Listeners.

We now need to create an event listener for the event to specify which action chains to start when the event occurs (more than one action chain can be started by an event listener).

9. On the page's **Event Listeners** tab, click the **+ Event Listener** button to create a listener.
10. For the wizard's Select Event step, scroll down to the Page Events section and select the custom event that you created. Click **Next**.

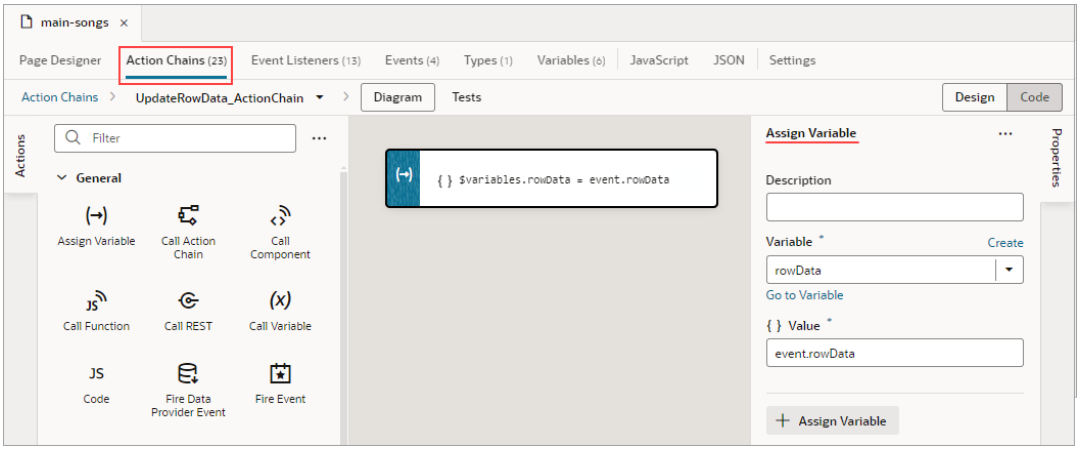


11. For the Select Action Chain step, click the Add icon for the Page Action Chains section to create an action chain for the listener to initiate.



When a listener's action chain is created here, if the listener's custom event has input parameters, the action chain is created with an `event` input parameter. This `event` object contains the custom event's input parameters (example: `event.param1`, `event.param2...`), and the `event` object is automatically passed to the new action chain.

12. In the Action Chain editor, note that the action chain has the `event` input parameter, which contains the custom event's input parameter. Add the Assign Variable action, and set its `Variable` property to the page variable that will contain the row's data. Lastly, set it's `Value` property to the relevant value in the `event` object that was passed to the action chain:



Test Action Chains

You can use the Tests editor–located on the Action Chain editor's **Tests** tab–to implement a test-driven development approach to designing, creating, and maintaining your action chains, or to implement your own methodology. Using the Tests editor, you can easily define test cases for an action chain and run them at any time, to ensure that code changes haven't broken any functionality.

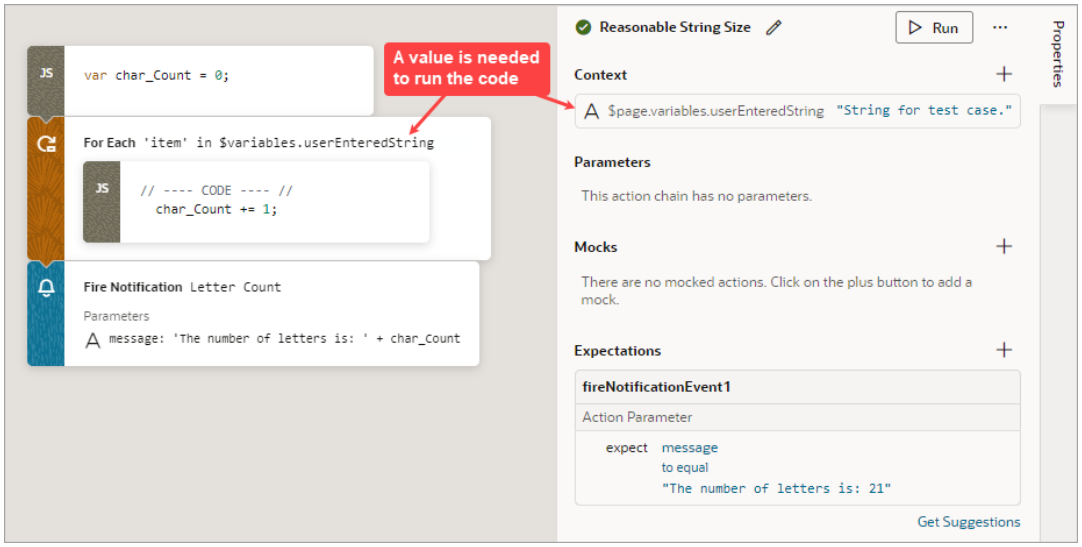
The Tests editor removes the need to manually code a test for each code path by:

- Displaying the action chain's input parameters, context variables and context constants, so that you just need to enter their values for the test.
- Displaying the actions that need their results provided for them, for you to provide the results for the code path being tested. For Call R actions, functionality is available for you to quickly get and copy their responses.
- Suggesting expectations for the test, based on the provided values.

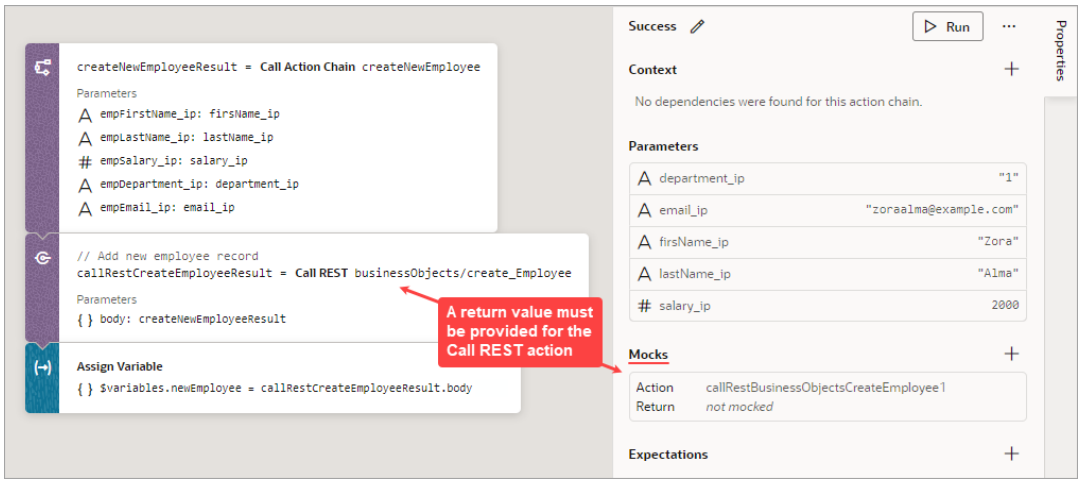
When testing action chains, the first thing you should do is figure out all of the possible code paths, since each one is a scenario that needs to be tested to achieve full test coverage. More complex code paths, however, might have more than one scenario that should be tested.

After identifying the code paths, you create at least one test for each, depending on how many scenarios there are for a path. For each test, you need to:

- Provide any initial values, such as initial values for variables and input parameters, that are needed to execute the code in the code path being tested. For instance, this code needs the value of the `$page.variables.userEnteredString` variable to execute. The variable is used to count the number of characters that a user entered into an Input Text component that's bound to the variable. Since the variable's value is needed to run the code, you need to provide a value for it that is appropriate for the test case being tested.



- For each action that can't automatically return a value during testing, due to limitations, you need to provide the action's return value for the code path being tested, as a mock. Actions that need their results provided for them are shown in the Mocks section.
  In this example, a new employee record is added using a Call REST action, which can't automatically return a value during testing. A return value must be provided for the Call REST action, which would be the action's result after adding the new record:

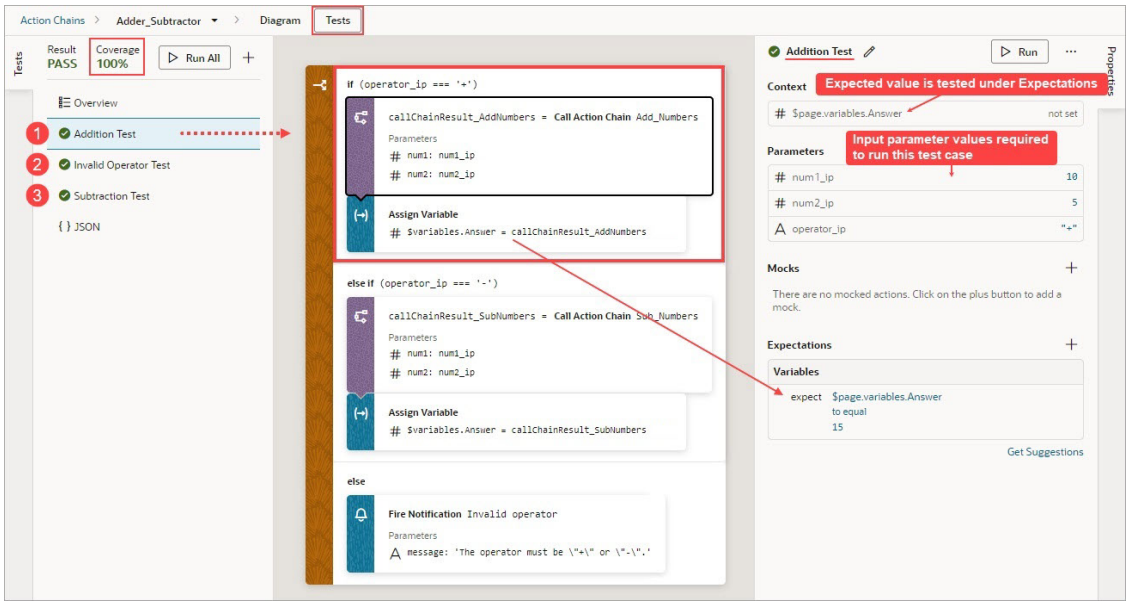


- Once you've provided the values for the test case, expectations are automatically generated for you, based on those values. Select the expected results, such as a variable's final value, to test against.
  For instance, after you provide the initial values and mocks for a test, if Visual Builder detects that a variable's final value will be 5, this expectation will be suggested to you. You can then add the expectation to the test, to test against. For the test to pass, all expectations have to be met.

The goal is to fully test your action chain by testing each of its code paths. If your tests cover the expected results for each code path, the value for Coverage will be 100%.

In the example below, three tests have been created for the three code paths that need to be covered to achieve full test coverage: Addition Test, Invalid Operator Test, and the Subtraction Test. Therefore, the value for Coverage is 100%, as you can see in the upper left corner of the editor.

The Addition Test, which tests the `if operator_ip === '+'` code block, is shown in the Properties pane. In the Parameters section, the three input parameter values that were entered to execute the addition block of code are shown, and the Expectations section shows the expected results:



The source code for all of your tests is stored in a separate JSON file, `actionchainname-tests.json`, for easier maintenance. To view this file's contents, click **JSON** in the left pane. You can also find this file under the artifact's chains folder in the Navigator's Source View tab.

Create a Test for a Test Case

The first time you access the Tests editor, click the **+ Test** button to create a test for a particular test case. The test name defaults to `Test 1`; enter a more descriptive name for the test case, if you want.

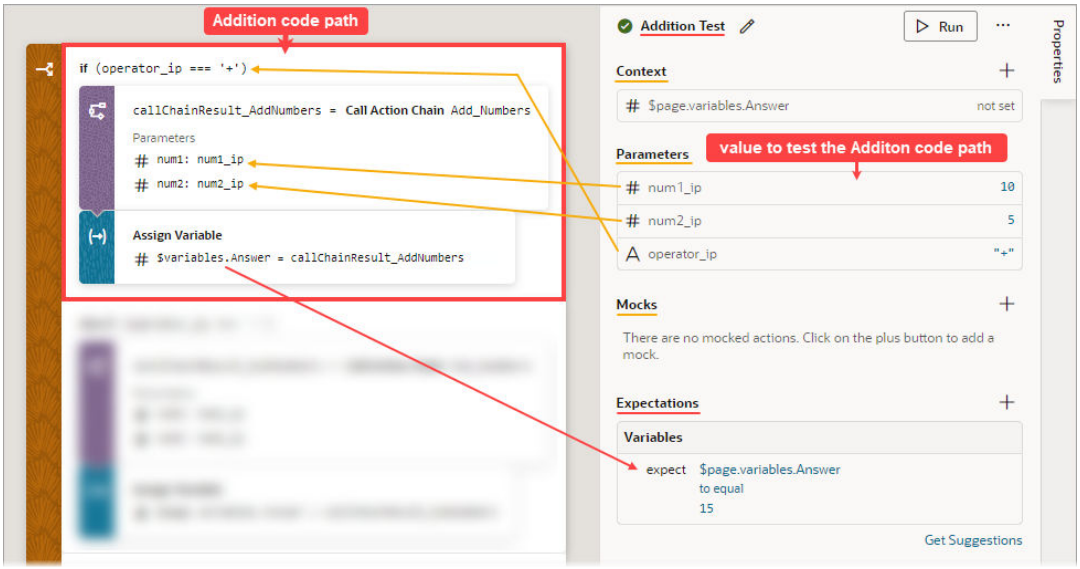
To create a test for a test case:

1. In the Context section, provide the initial values for any context variables and constants that are used in the code path that is being tested. For instance, if a variable is used in the

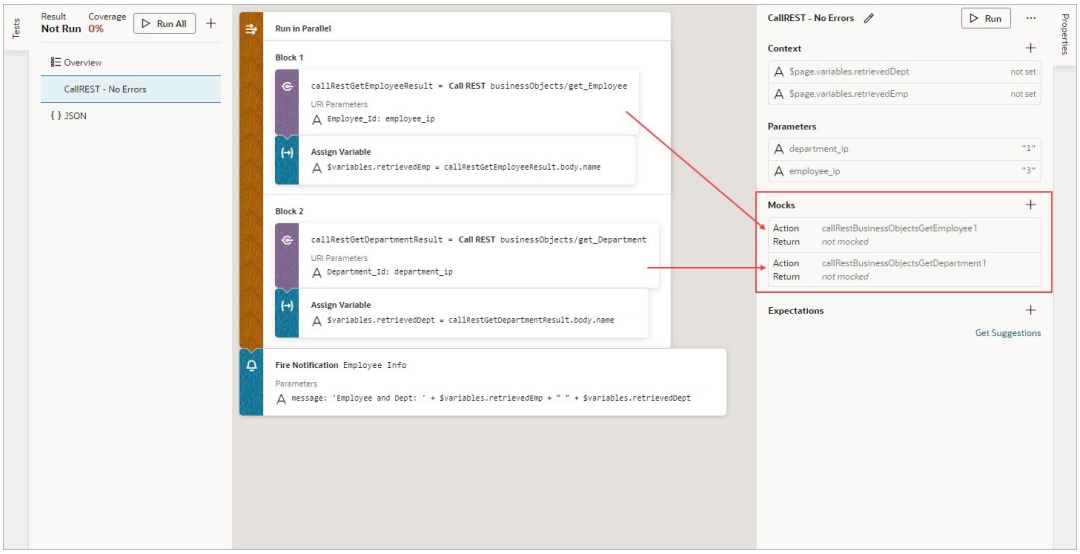
code path for a calculation, you'll need to provide a value for the variable that appropriately tests the code path.

> **Note:**
>
> If a variable or a constant's value is set by the code being tested and not required to execute the code, you don't provide an initial value for it. The expected value for the variable or constant will be suggested to you as an expectation. For instance, in the example that follows, the value for the variable `$page.variables.Answer` in set by the Assign Variable action. Since the value is set by code, the expected value for the variable is suggested to you in the Expectations section, as an expectation.

2. In the Parameters section, provide the values for any input parameters that are used in the code path. In this example, values have been entered for the three input parameters that are used in the addition code path:



3. In the Mocks section, which shows actions that need their results provided for them, provide the results for any listed actions, and ensure that the values are appropriate for the test case. For instance, if a Call REST action is used, you'll need to provide a response from the Call REST action that properly tests the code for the particular test case.

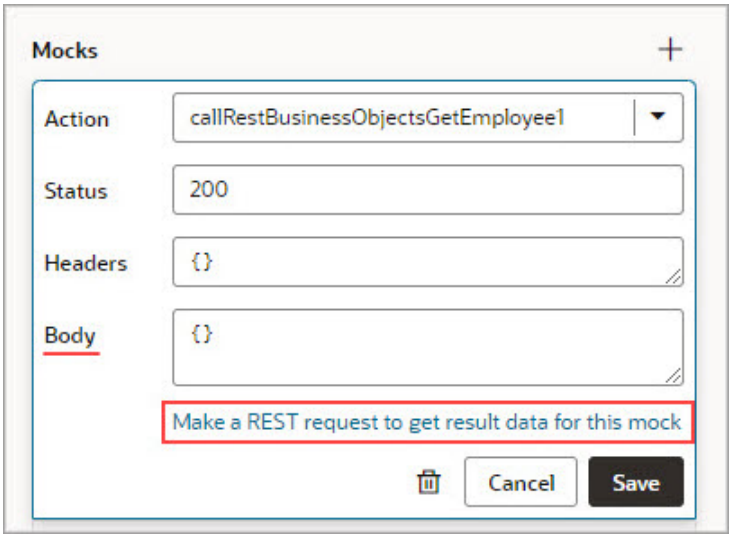


These actions always require mocked results:

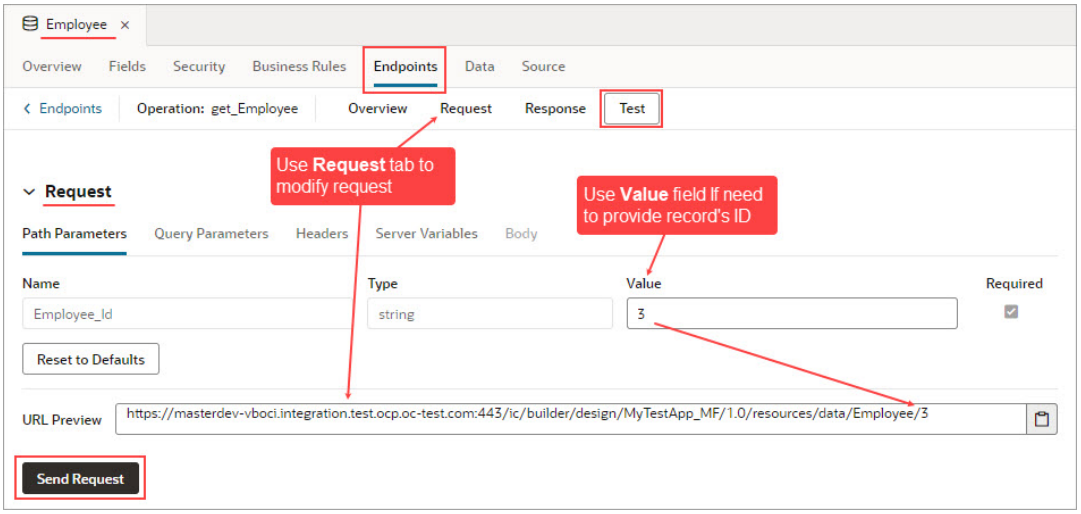
- Call Component
- Call REST
- Call Variable
- Get Location
- Scan Barcode

To provide a mock for an action, click the action in the Mocks section. In the resulting window, provide a value that is appropriate for the test case.

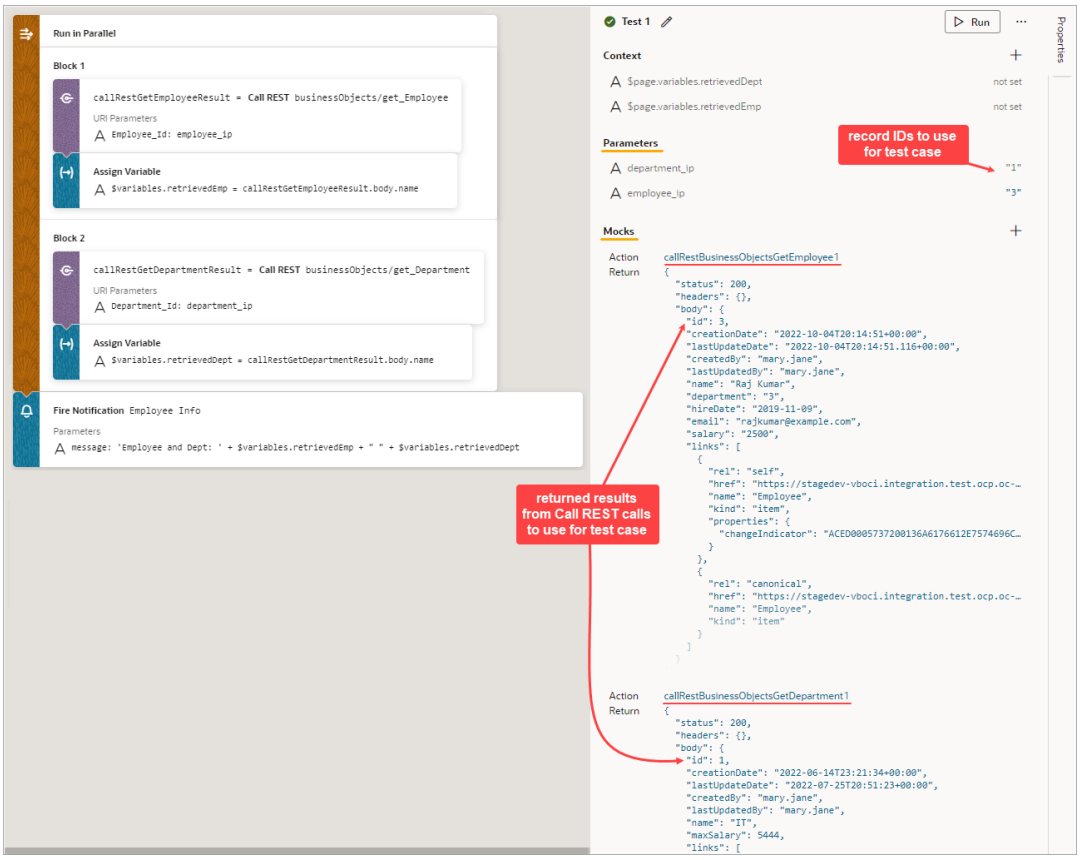
If you need to mock a Call REST action, click the **Make a REST request to get result data for this mock** link to copy a response from a REST request to paste into the Body field:



You will be taken to the **Endpoint** tab for the Call REST action. Here, you can modify the request to get and copy the response required to mock the action's result:



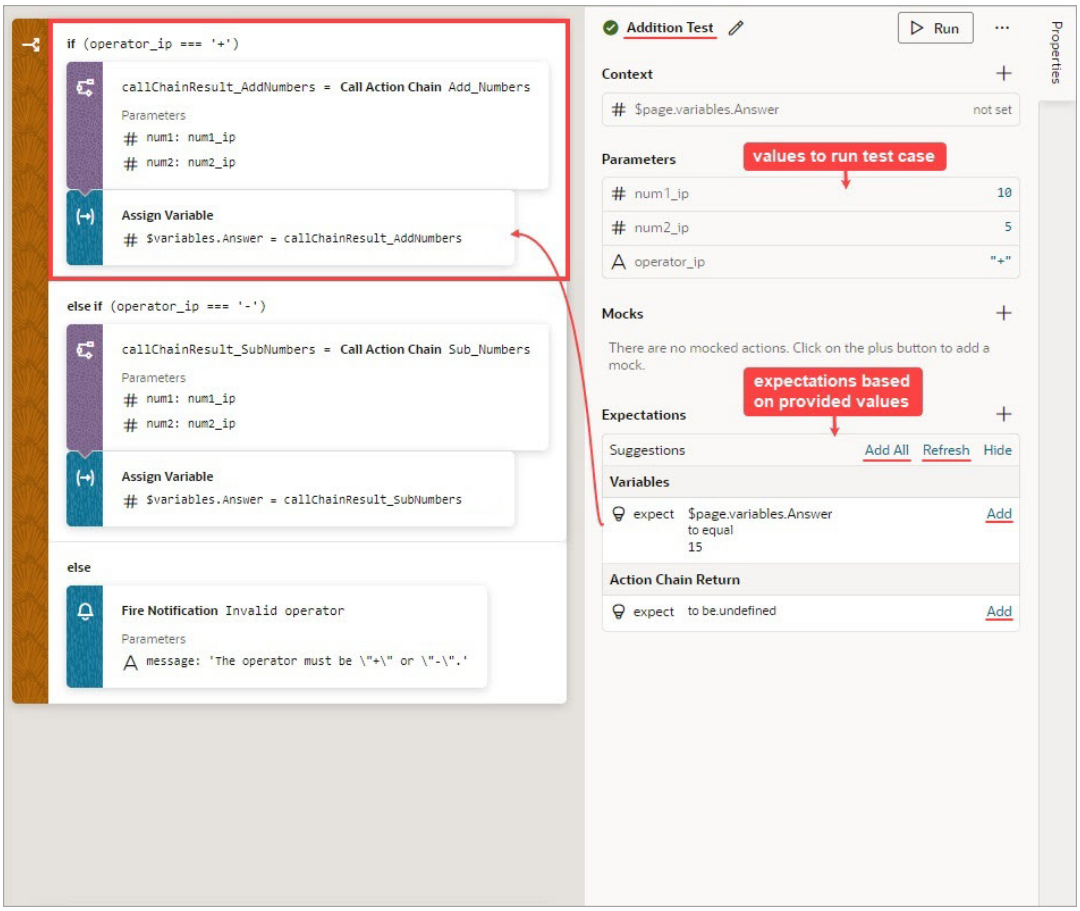
Shown here are the copied responses from the Call REST action requests, provided as mocks:



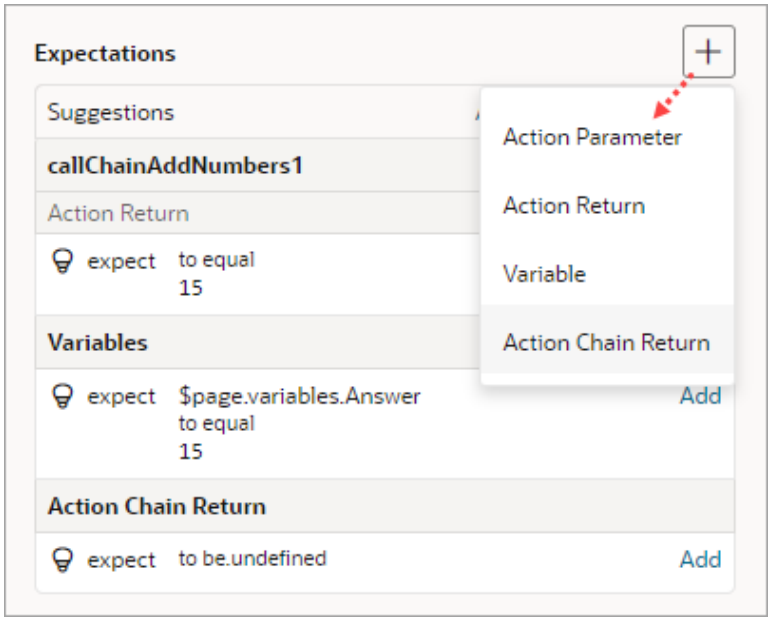
To add a mocked result for an action that isn't shown in the Mocks section, click the Add icon (**+**) for the Mocks section. Select the action from the **Action** drop-down list, then provide the value for the test case in the **Return** field.

4. Now that you've provided the values for the test case, you can get suggested expectations based on those values by clicking the **Get Suggestions** button in the Expectations section.

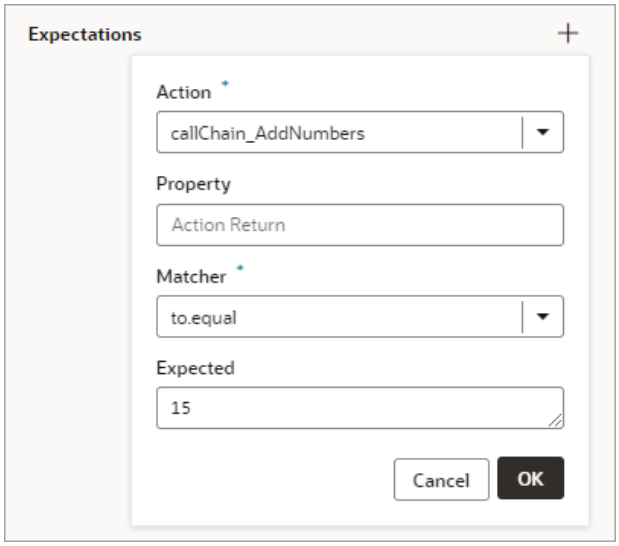
Shown here are the suggested expectations that are based on the values entered for the Addition Test. In the Expectations section, use the **Add All** or **Add** links to add the applicable expectations. To refresh the expectations list, click **Refresh**:



To add your own expectation, click the Add icon (+) for the Expectations section. Select whether you'd like to create an expectation for an action's parameter, an action's return value, a context variable, or the action chain's return value:



Make the appropriate selections for the expectation and provide the expected value, as shown in this example. Click **OK**:



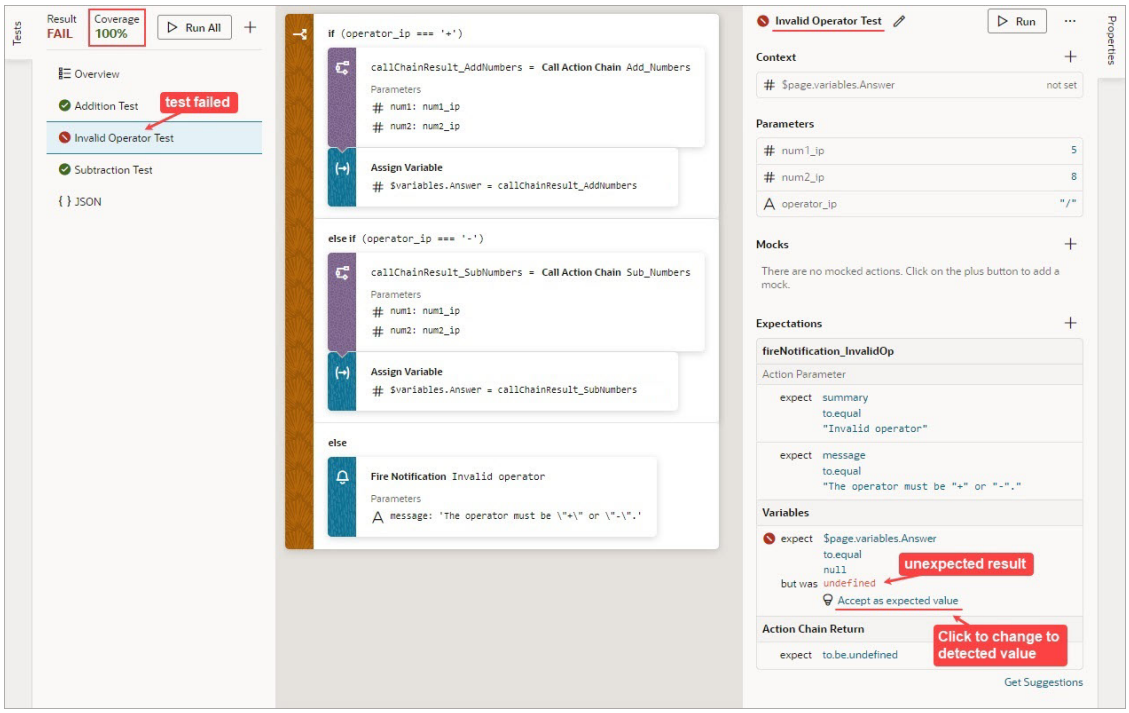
Run the Tests

Once you've defined the tests, you can run them individually or all at once using the **Run** or **Run All** button.

You can also run these tests using the vb-test Grunt task (see Test Action Chains Using the `vb-test` Grunt Task).

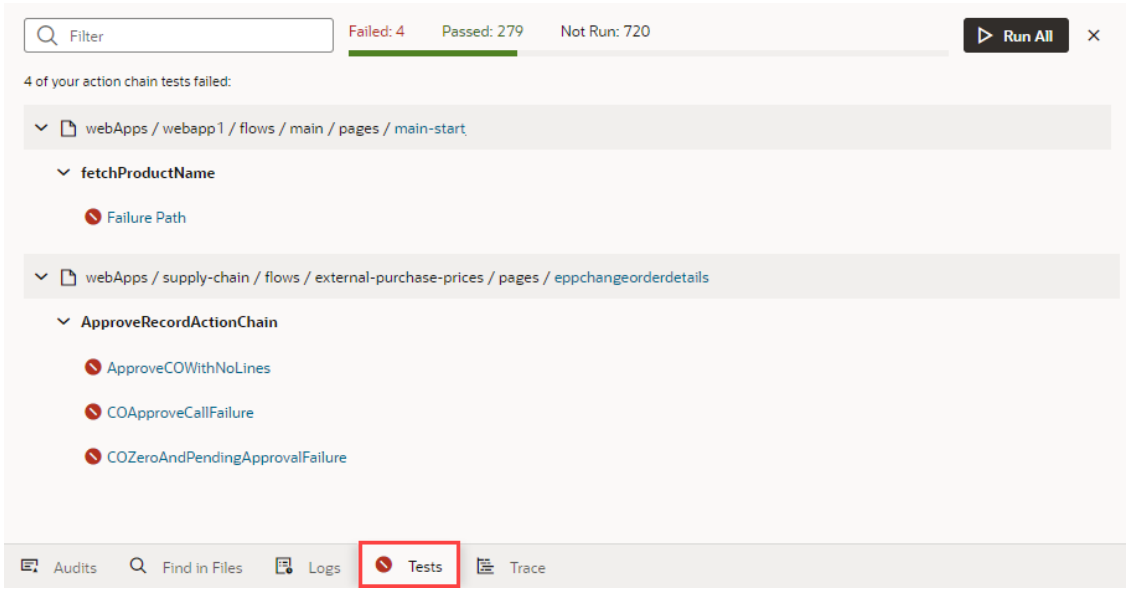
After running the tests, a green icon beside a test indicates that its expectations were as expected and a red icon indicates an unexpected result. The reason for the failure is shown in the Expectations section.

If you incorrectly set an expected value and the detected expected value is correct, click the **Accept as expected value** link to change the expected value to the detected value:



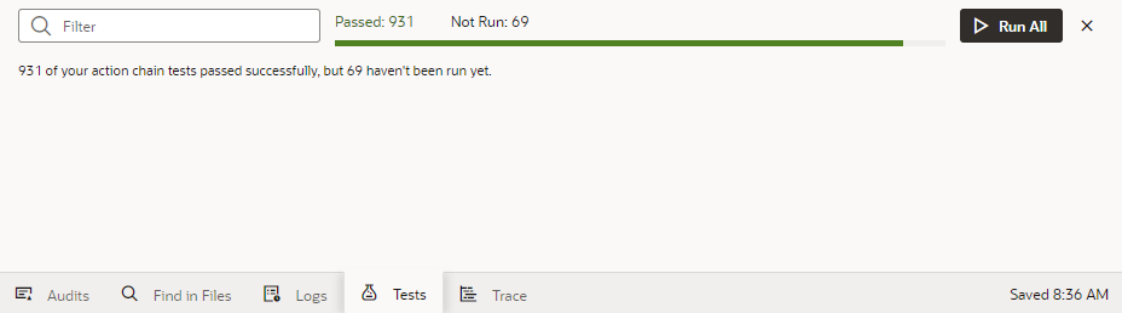
Use the Tests Footer in a Visual Application

When you define multiple tests for each action chain in an app, it might be easier to manage tests for all apps in a visual application, rather than for each app. You can do this using the **Tests** footer at the bottom of your browser.



This aggregate view helps you get a quick look at the status of all action chain tests in a visual application. When tests fail, you can use this view to quickly access the editor for each failed test and take action as needed.

While you can also run all tests in your application from here, it isn't really required if you've already triggered them. Action chain tests run in the background, even when you're not actively working on your application, and the results are saved. So if you are working on an application, only tests impacted by your changes (for example, if you added a new variable or updated an existing function) are scheduled to run again. You'll likely see something similar to this image until Visual Builder actually runs those tests for you (after 10 seconds of inactivity):



This way, your test results are always available and up-to-date, and you can rely on them to identify and fix code-breaking changes.

Test Action Chains Using the `vb-test` Grunt Task

Visual Builder's `grunt-vb-build` NPM package includes a `vb-test` Grunt task that you can use to run the action chain tests in your visual application on your computer.

To use the `vb-test` Grunt task, you must set up your computer to build the visual application by installing Node.js and its package manage (npm). See Build Your Application Locally. Once you have installed the necessary tools, you need to save the sources for the visual application to your computer. You can get the sources of your visual application in one of the following ways:

- Cloning the Git repository containing the sources
- Exporting the visual application from Visual Builder and extracting it to your local system

To test action chains using the `vb-test` Grunt task:

1. In the command-line interface, navigate to the folder on your local system containing the `package.json` and `Gruntfile.js` files.
2. Enter `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`.
3. Enter the task names in the command-line interface to process your application sources, and then run the suite of tests that you defined in Visual Builder.
   The following example shows how you execute these tasks, along with some of the parameters that they support:

```
# First build application sources. This creates a build/processed
directory with the built application assets.
./node_modules/.bin/grunt vb-process-local

# Run the suite of action chain tests that you defined using one of the
following options:
## Headless mode:
grunt vb-test
```

```
## Test in the Chrome browser and set a timeout value:
grunt vb-test --karma-browser=Chrome --mocha-timeout=60000
```

The command-line options include the following:

- `karma-browser`

  By default the tests run in headless Chrome, but you can pass Chrome to use the UI (window) mode instead.

  Example to run tests in Chrome UI:

  ```
  grunt vb-clean vb-process-local vb-test--karma-browser=Chrome
  ```

- `karma-debug`

  Runs tests in Chrome UI mode. Suspends execution until you click the **DEBUG** button, at which point you can debug the app tests using Chrome DevTools. The Default value is `false`.

  Example to run tests in Chrome UI and debug mode:

  ```
  grunt vb-clean vb-process-local vb-test --karma-browser=Chrome --karma-
  debug
  ```

- `karma-log-level`
  Sets the karma logging level. The default level is `INFO`, though you can change this to `DEBUG`, `WARN`, `ERROR`, or `DISABLE`.

  Example to run tests in Chrome UI with increased verbosity:

  ```
  grunt vb-clean vb-process-local vb-test --karma-browser=Chrome --karma-
  log-level=DEBUG
  ```

- `mocha-timeout`
  Sets the timeout for Mocha tests (in milliseconds).

  Example to run tests in Chrome UI with a timeout for Mocha tests:

  ```
  grunt vb-test --karma-browser=Chrome --mocha-timeout=60000
  ```

For more information on the supported command-line options, see vb-test.

4. Check test results and code coverage reports in the `build/tests/results` directory.

11

Work With JSON Action Chains

Action chains determine what happens when users interact with pages or components in your user interface, for example, when they select a row in a table or click a button on a page. Each action chain defines a sequence of **actions** (for example, navigating to a page, calling a REST endpoint, assigning data to a variable, and so on) and is started by an **event listener** when an **event** occurs.

Say you want users who click an Edit button on a List of Employees page to be taken to a page that lets them change employee details: you'd define an `ojAction` event for the Edit button, create an event listener that listens for the `ojAction` event, and select an action chain with a Navigate action to open the Edit Employee page. Now whenever the `ojAction` event occurs, the event listener triggers the action chain to navigate to the edit page.

> **Note:**
>
> All new action chains are now created using JavaScript, which offers significant advantages over JSON. Existing JSON chains can be edited, but no new JSON chains or tests for JSON chains can be created. For a list of benefits to using JavaScript, see the following subsection.

JavaScript and JSON Action Chains

You can call a JSON action chain from a JavaScript action chain using the Call Action Chain action; however, you can't call a JavaScript action chain from a JSON action chain.

We recommend that you use JavaScript action chains (rather than JSON), as they provide a number of benefits, including:

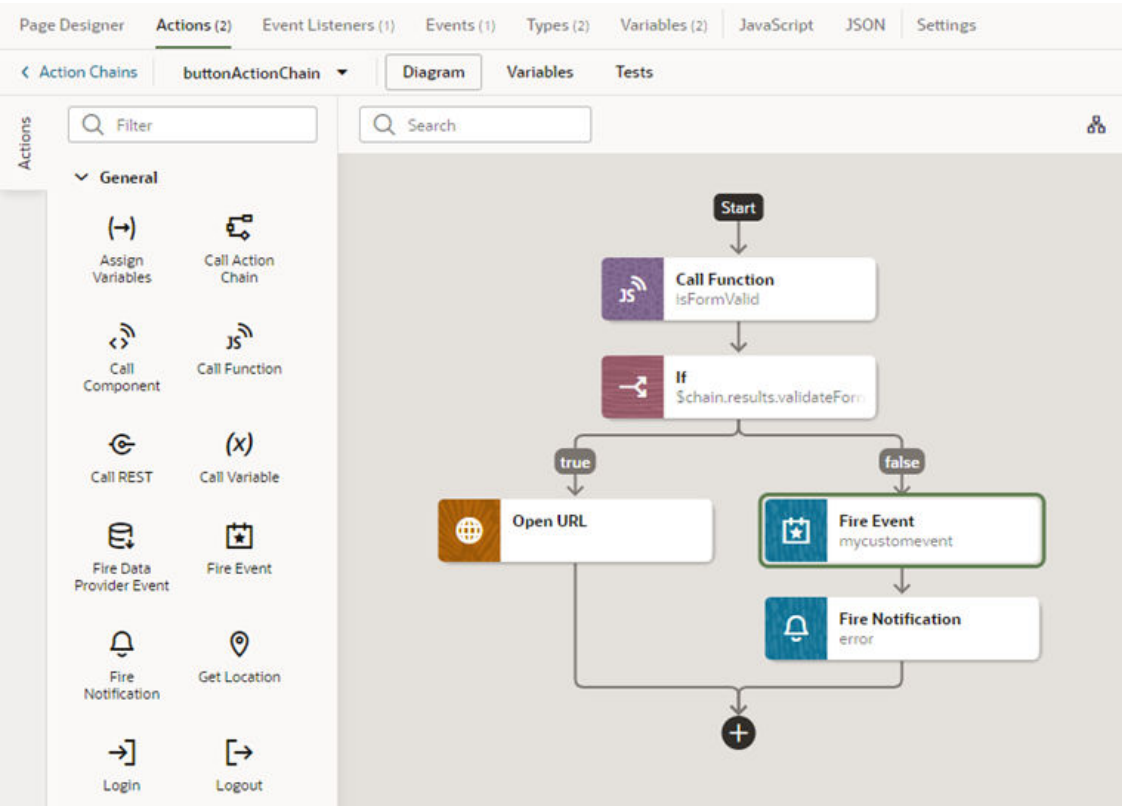
- The JavaScript Action Chain visual editor offers a helpful Structure pane.
- The JavaScript code editor includes an Actions palette, a Structure pane, and a Properties pane to assist with writing code.
- JavaScript action chains can use local functions to improve modularization and keep related logic together.
- Using JavaScript offers numerous advantages, including:
  - Easier debugging, since you can use your browser's Developer tools.
  - Easier management of code through Git operations, such as merge.
  - Self-contained action chains, eliminating the need for external module functions to handle complex code.
  - Built-in types with extensive methods and properties for simplifying common tasks.
  - Powerful data manipulation capabilities, including complex transformations, filtering, and mapping.
  - A rich standard library to leverage.
  - A comprehensive range of control flow and transfer statements, like while loops and try-catch statements, for implementing complex logic and error handling.

About Action Chains

An action chain determines what happens when, for example, you click a button on a page. To configure the button's behavior, you could define an `ojAction` event for the button, create an event listener that listens for that `ojAction` event to occur, and select the action chain that the event listener will start. When the button's `ojAction` event occurs, the event listener starts the action chain.

An action chain might be a short sequence of a few actions, but it could contain many actions as well as logic for determining what happens in the sequence. It might contain actions such as assigning data to a variable, sending data to a database, navigating to another page, even starting other action chains.

This image shows what an action chain that opens a URL might look like:



Much like variables, the scope of an action chain depends on where it's created. An action chain created in a page's Action Chain editor can only be used in that page, and can only access variables defined in that page. An action chain that you create in the Action Chain editor for a flow can only be used within that flow. For action chains that you want to use in multiple pages of your application, such as one that navigates to the start page, you can create an action chain at the application level.

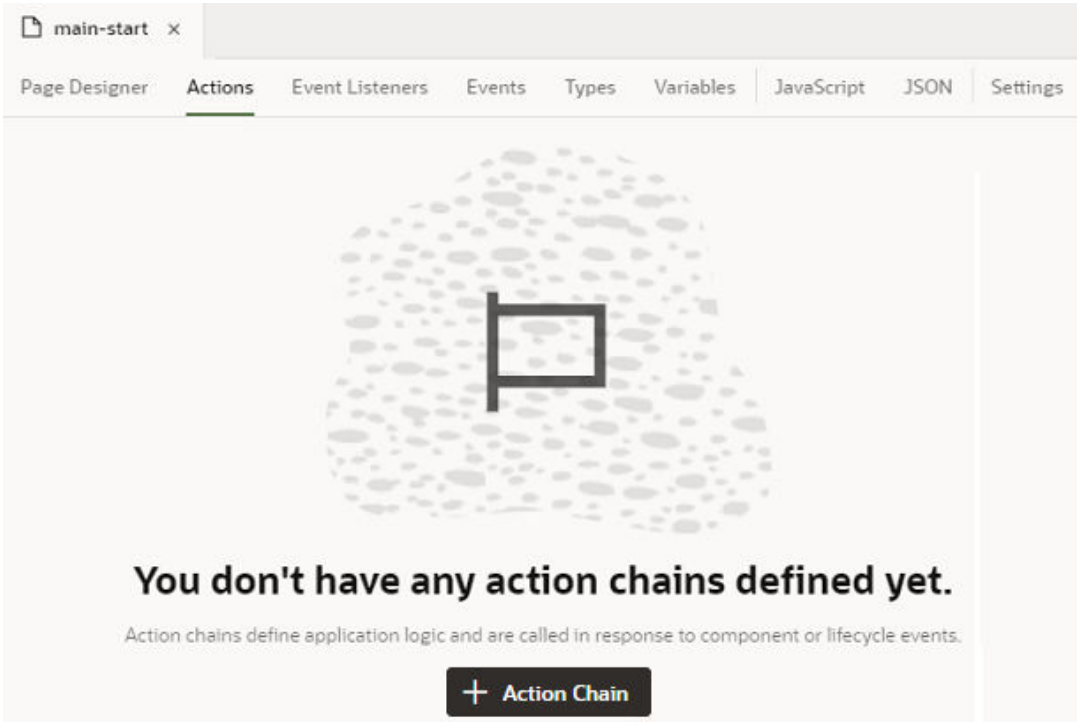
Create an Action Chain

Create action chains by assembling several, individual actions into a sequence in the Action Chains editor. The Actions palette contains a list of built-in actions that you can drag on to the canvas to create your sequence.

To create an action chain:

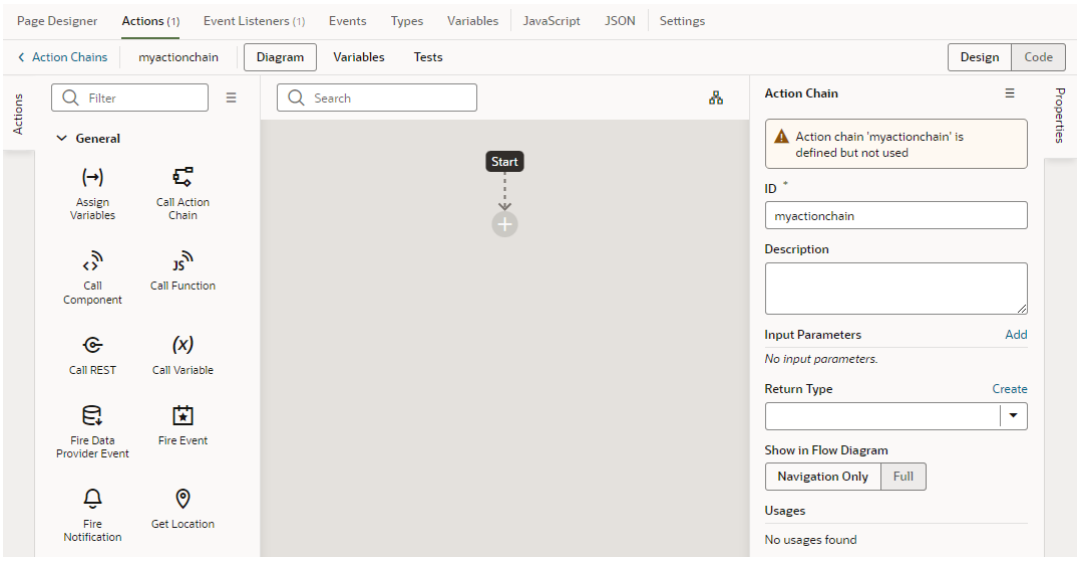
1. Open the **Actions** tab, for example, at the page level.

   The Actions tab 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 and, optionally, a description. Click **Create**.

   The new action chain opens in the editor:

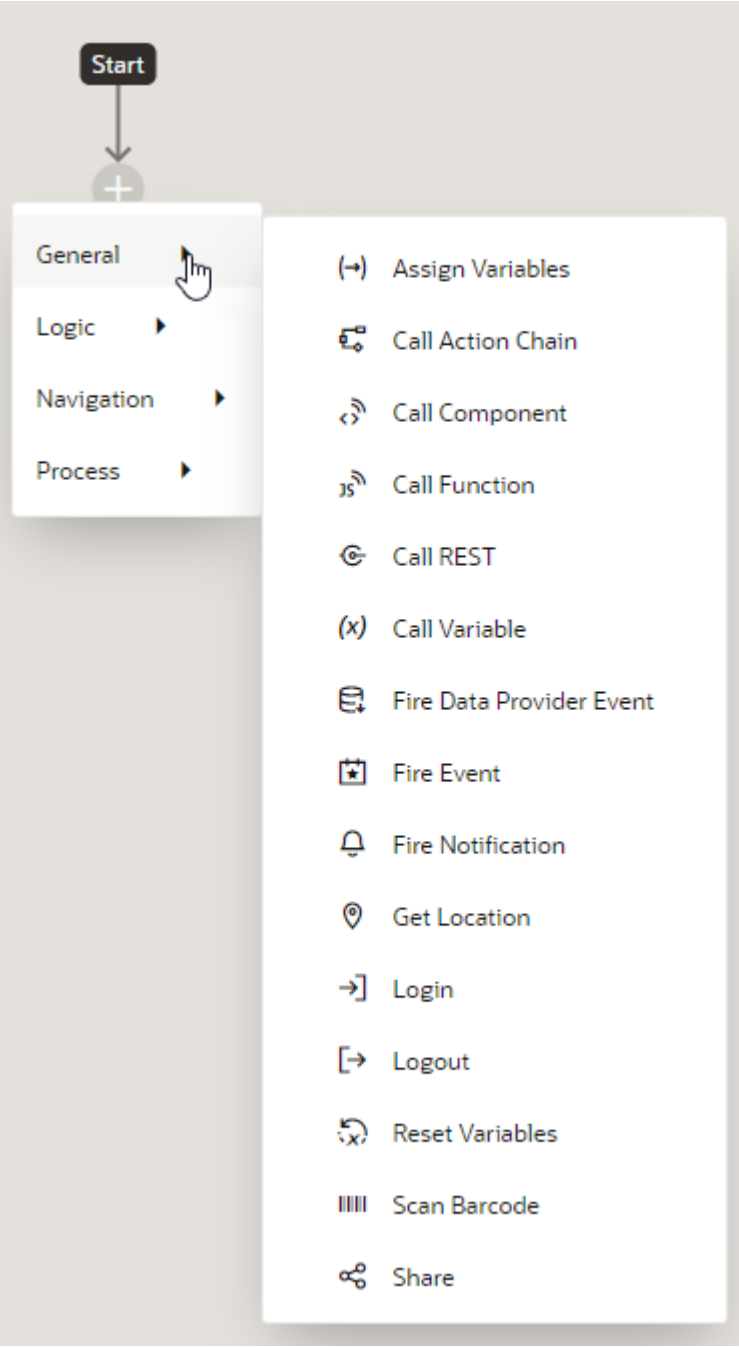


The editor contains the palette of built-in actions (grouped by type), a canvas, and a Properties pane. The Start icon in the canvas area indicates the starting point for your action chain; the Add icon ( ) is a placeholder where you add an action to the chain. The Properties pane shows the properties of what's selected on the canvas.

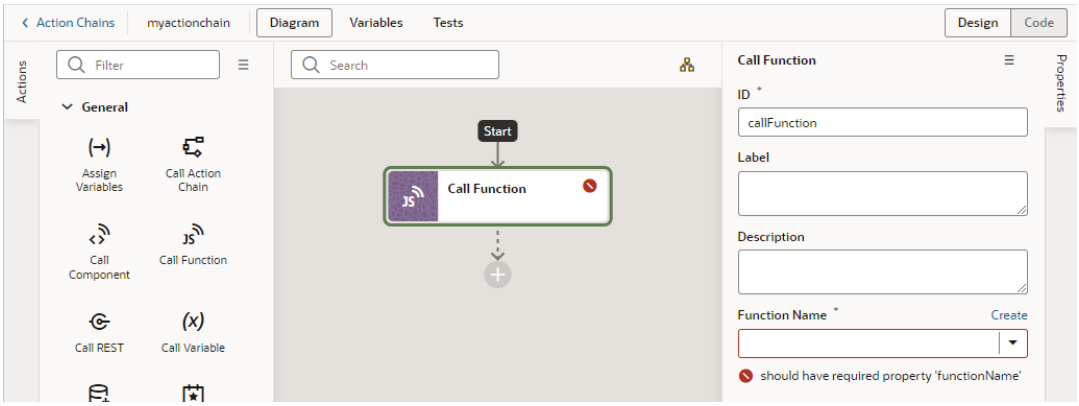
If you prefer to wire up your action chain manually, you can use Code view to directly edit the action chain's source code. For supported syntax, see Actions and Action Chains in the *Oracle Visual Builder Page Model Reference*.

> **Tip:**
>
> It's possible to declare local variables that are only available within the scope of your action chain. To do this, click the **Variables** tab in the Action Chains editor and create your variable. These variables are internal to the action chain and can be used internally by actions in the chain. You can also pass them as input parameters to the action chain.

4. From the actions palette, drag an action and drop it on the Add icon ( ). You can also click the Add icon ( ) in the chain and select an action in the pop-up menu.

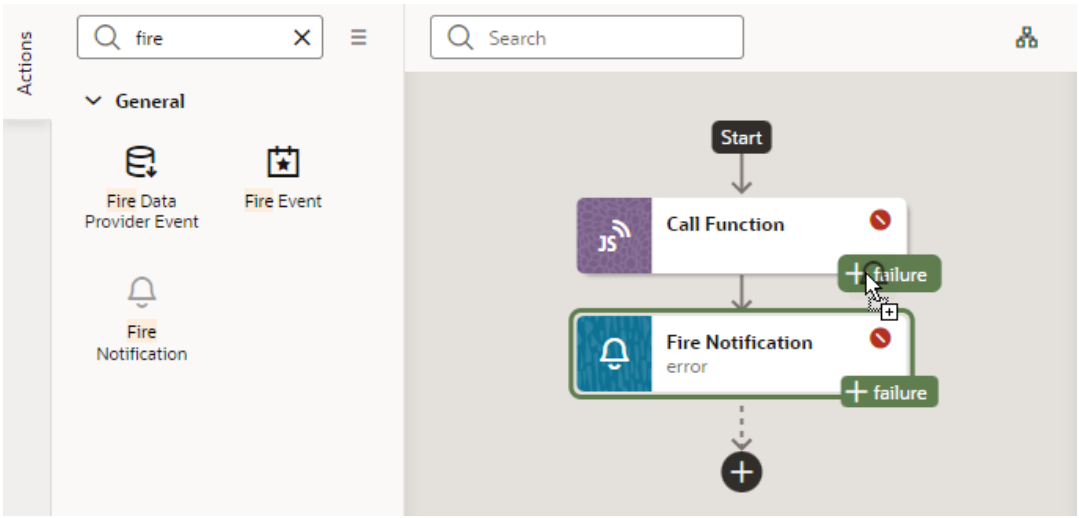


The new action is added to the chain and is selected by default. The Properties pane displays the properties that you can specify for the action. For example, here's what the editor looks like when you add the Call Function action in the Design view:

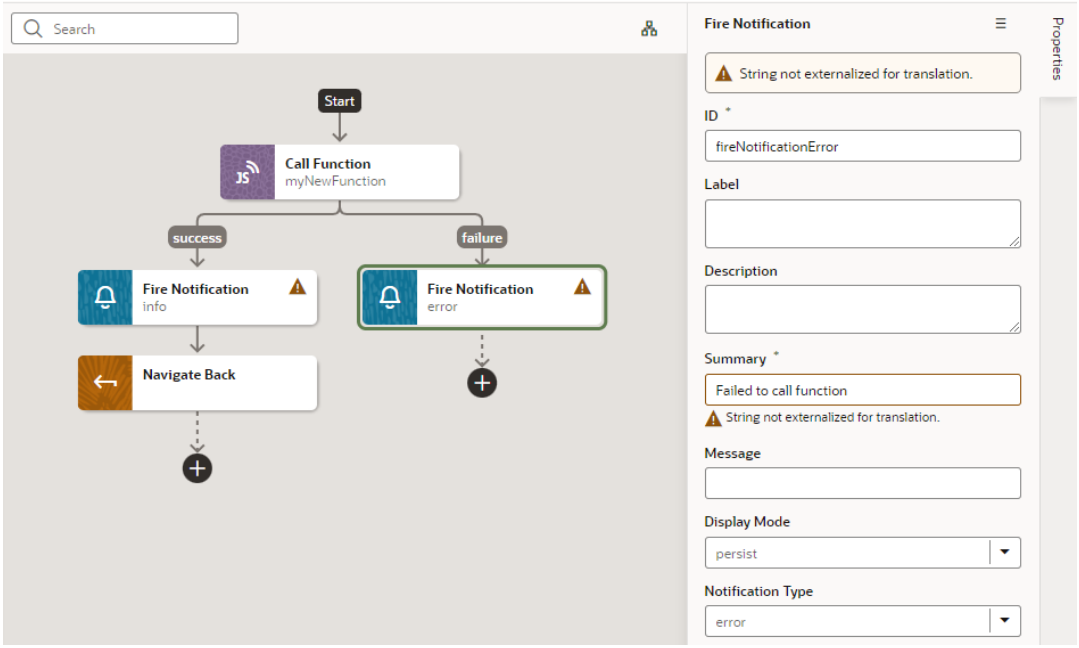


The action is usually flagged with a warning icon when a required field isn't set (in this case, because a JavaScript function hasn't been selected yet). Specify the action's properties as required in the Properties pane. For details specific to an action, see Built-in Actions.

5. 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.



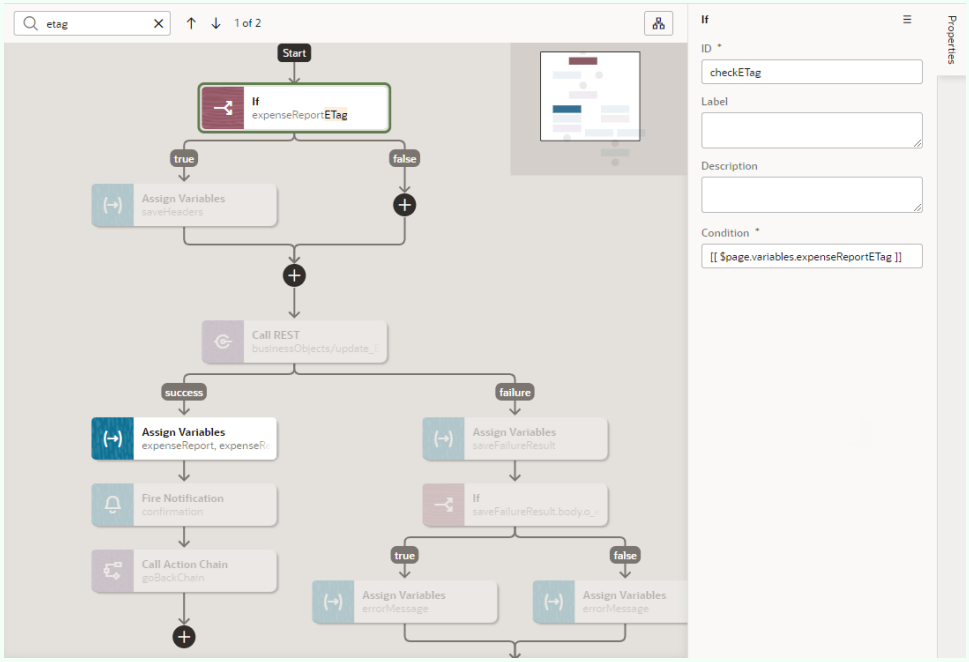
6. Repeat step 4 (and optionally step 5) until your action chain is complete. The action chain is saved automatically.



If you want to remove an action from the chain, select the action on the canvas, right-click, and select **Delete** ( ). You can also click **Delete** in the Properties pane's options menu.

Tip:

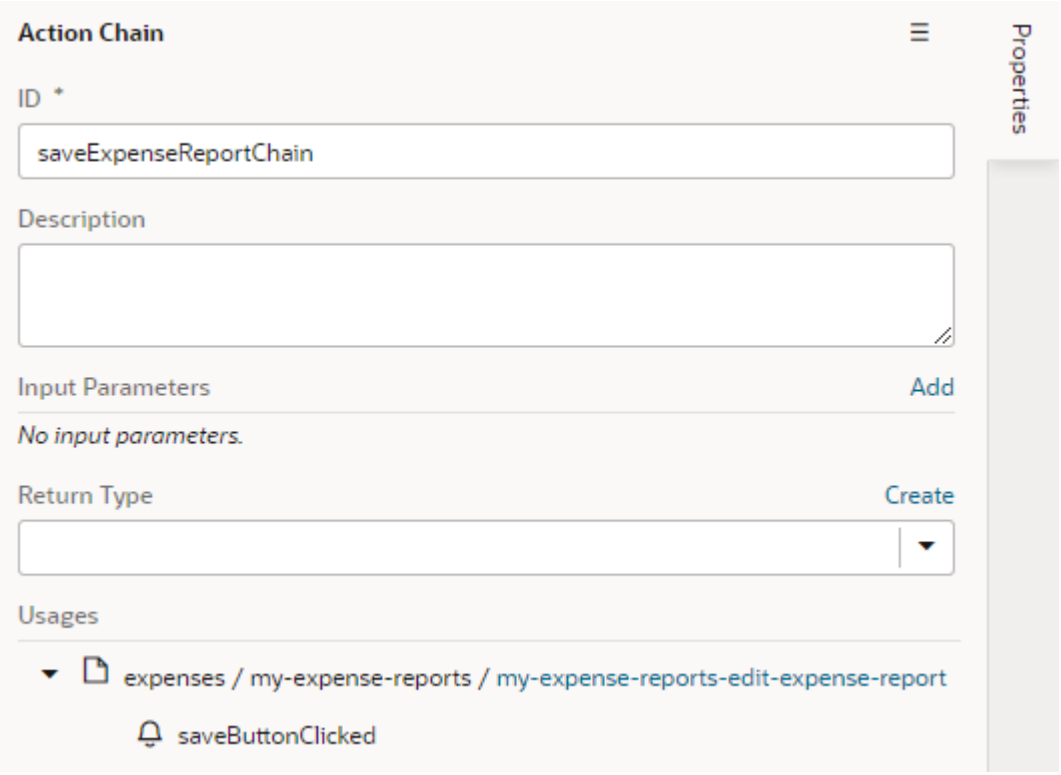
When your action chain includes a large number of actions, you can use search to quickly find what you're looking for. In the Search text box, enter any text—variable name, endpoint ID, or even an action ID from the console log. All actions that match the text you enter will be highlighted, along with navigation arrows that you can use to jump from one highlighted action to the next.
Here's an example of using "`etag`" to find the `expenseReportEtag` variable:



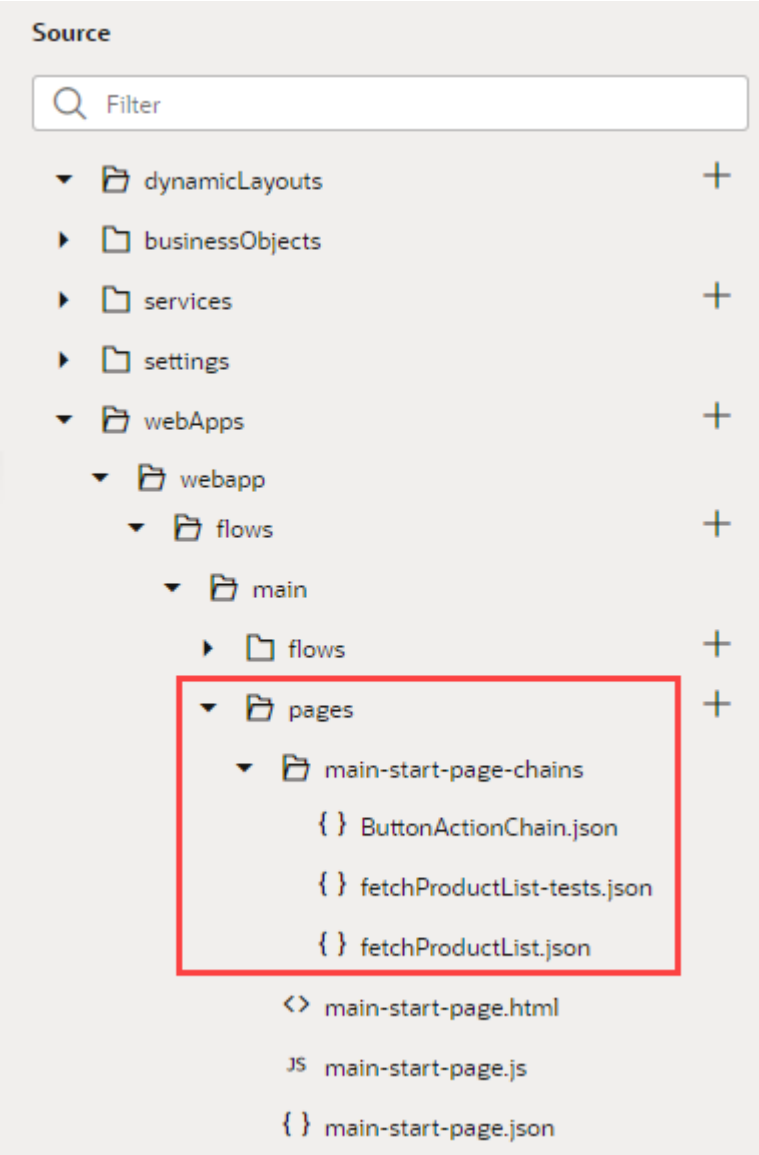
You can also click the Show Overview icon ( ) to view a visual representation of the action chain's flow. In combination with search, the overview diagram can help you know where the highlighted action is within the overall flow. Toggle the icon to show or hide the diagram as required.

You can open your action chain at any time from the **Actions** tab and edit it as necessary. When your action chain is complete, you can call it in response to a component event, a lifecycle event, or from another action chain. You can also trigger it when a variable changes.

If you want to view usage details for your action chain (for example, to see which pages use the action chain), look under **Usages** in the action chain's Properties pane. Click a usage to readily navigate there. The event listener tied to the event that calls the action chain is also listed, as shown here:



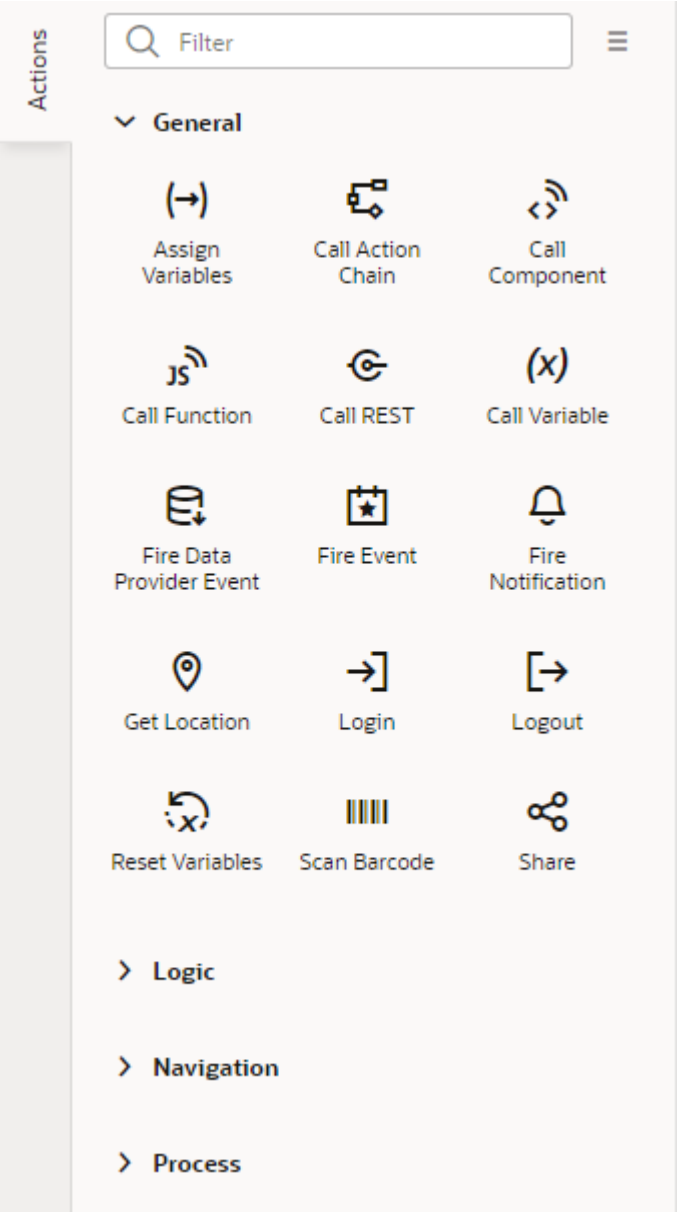
An action chain's source code is stored in its own JSON file. This helps to optimize performance by reducing the size of the artifact JSON and to reduce the potential for merge conflicts when multiple action chains for an app integrated with a Git repo are edited. To view and edit an action chain's JSON file, it's simplest to use the **Code** editor in the Diagram view, though you can always view files using the Navigator's **Source** view. For action chains in applications, flows, and layouts, look in the artifact's `chains` folder. For action chains in pages, look in the *`pagename`*`-page-chains` folder under `pages` that's at the same level as the page JSON file:



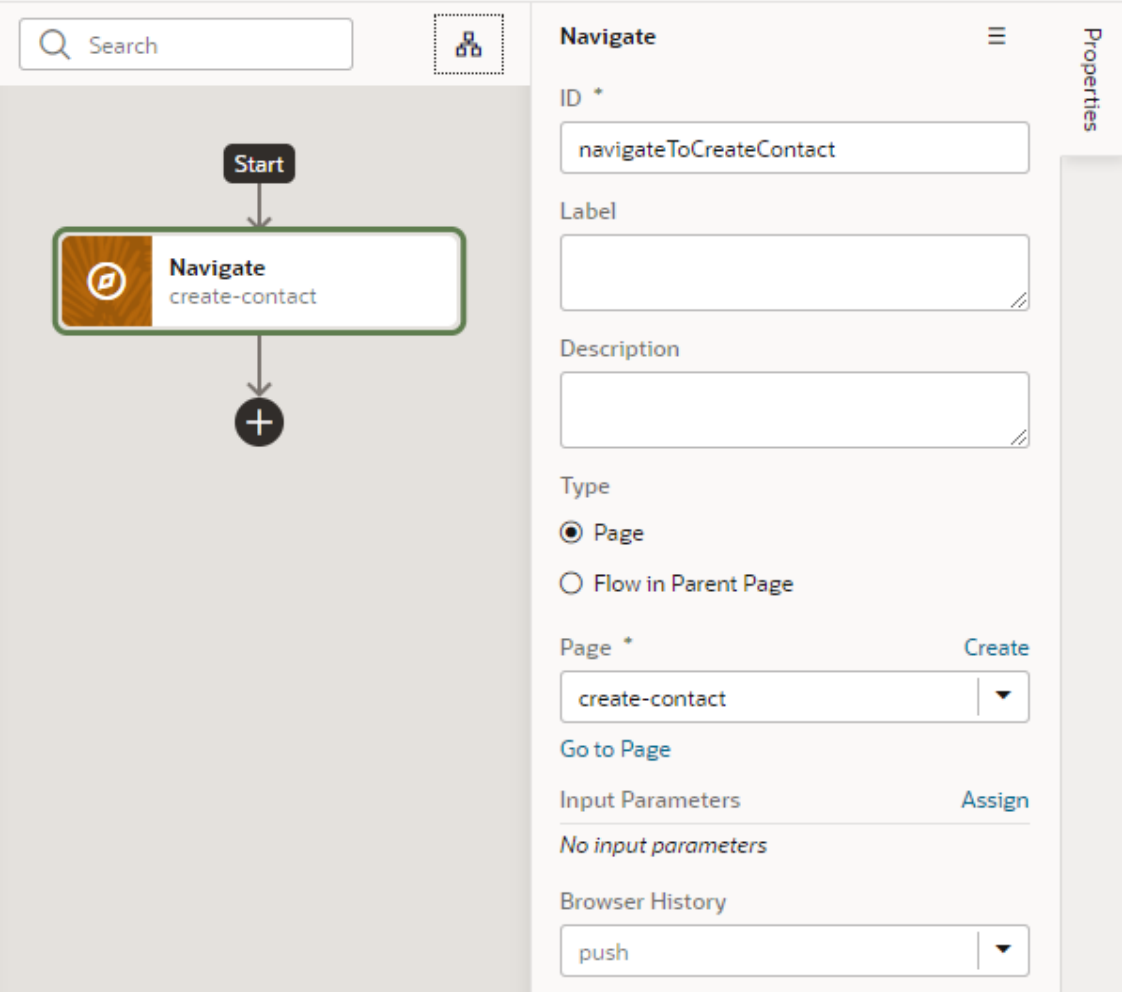
If you create tests for your action chain, those will be stored in another JSON file, distinct from the action chain's file.

Built-in Actions

Visual Builder provides a set of built-in actions that you use to create your action chain.



Each action performs a specific function and requires you to set different properties. 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 to the Call REST endpoint action. Similarly, when you add the Navigate action to an action chain, you are required to select a page in your application that the action navigates to, as shown here:



Regarding an action's output, an action can have multiple potential outcomes (such as success or failure, or a branch), and it can return results. For more details, refer to Action Results in the *Oracle Visual Builder Page Model Reference* guide.

Use this section to learn more about the steps particular to a built-in action.

> **Note:**
>
> Some built-in actions might be deprecated over time. To view actions deprecated in the latest release of Visual Builder, use the **Show Deprecated** option in the Actions palette's menu. The actions will show up in the actions palette, but won't be updated any more. The **Show Deprecated** option gives you time to move away from deprecated actions in your action chains.

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.

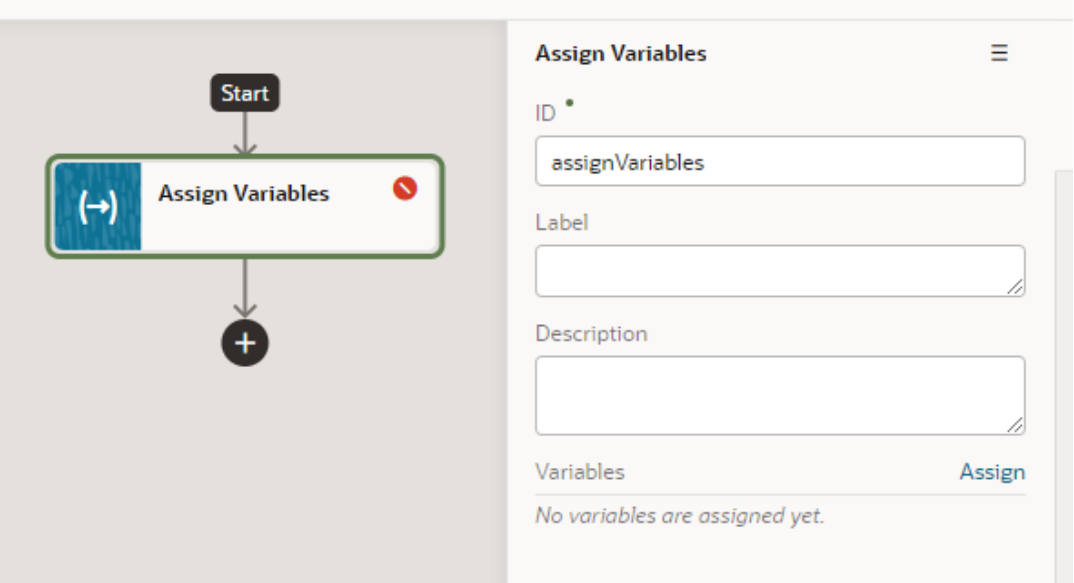
For example, if your action chain sends a request to a GET endpoint, you can use the Assign Variables action to map the response to a page variable bound to a page component. Or,

suppose 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 Action Chain 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.

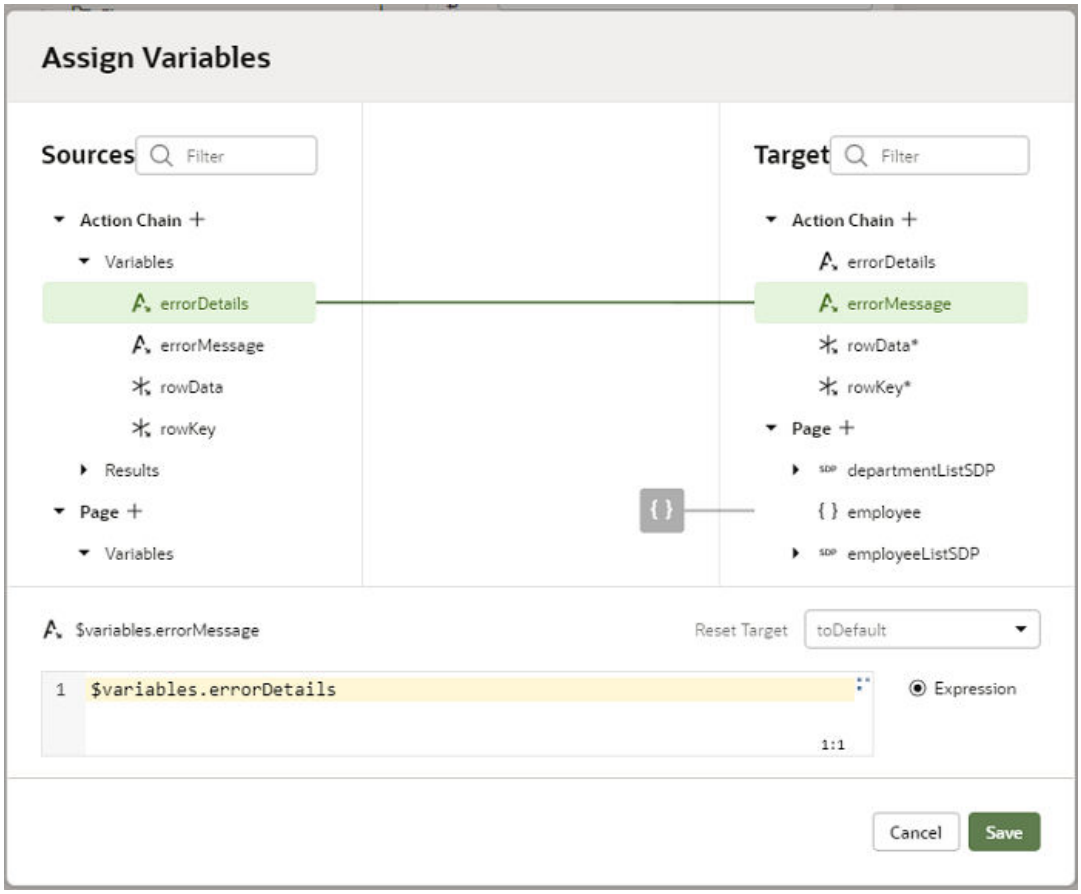
   You can drag the action onto the Add icon ( ) in the action chain, or between existing actions in the chain. The properties pane opens when you add the action to the chain.



   The Assign Variables action is badged with a warning icon when no variables have been assigned.

4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. Click **Assign** in the properties pane 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.

   If you need to define the variable, use the **+** icon to open a dialog where you can define a variable for the artifact (action chain, page, flow, or application).



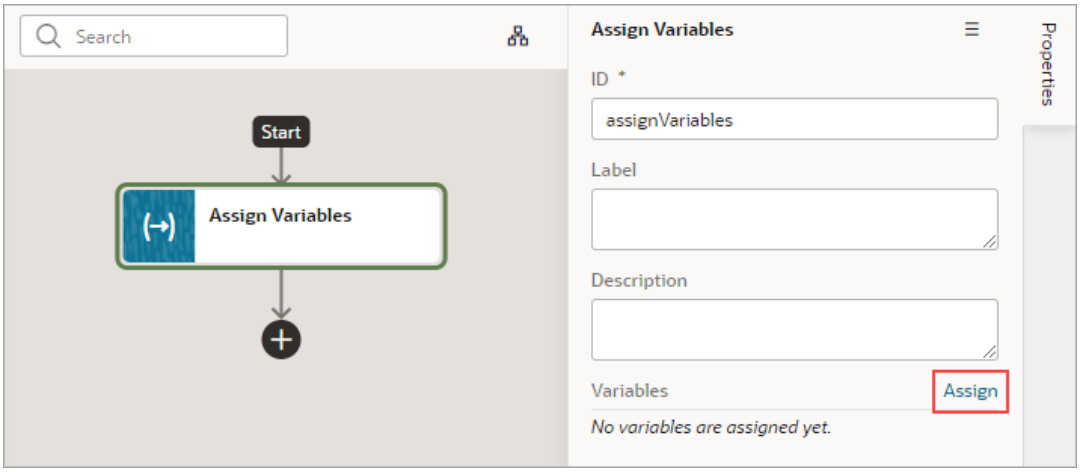
Use Filter Builder to Create Filter Criteria for an SDP

If you're using an SDP to provide a table or list's data, and you'd like to filter out rows, you can use the Assign Variable action to create and assign the filter criteria to the SDP's filterCriterion property. For further details about using an SDP to filter a table or list's rows, see Filter Data by Filter Criteria

When the Assign Variable action's Variable property is set to an SDP's filterCriterion property, the Filter Builder appears under the Variable property for you to create the filter criterion.

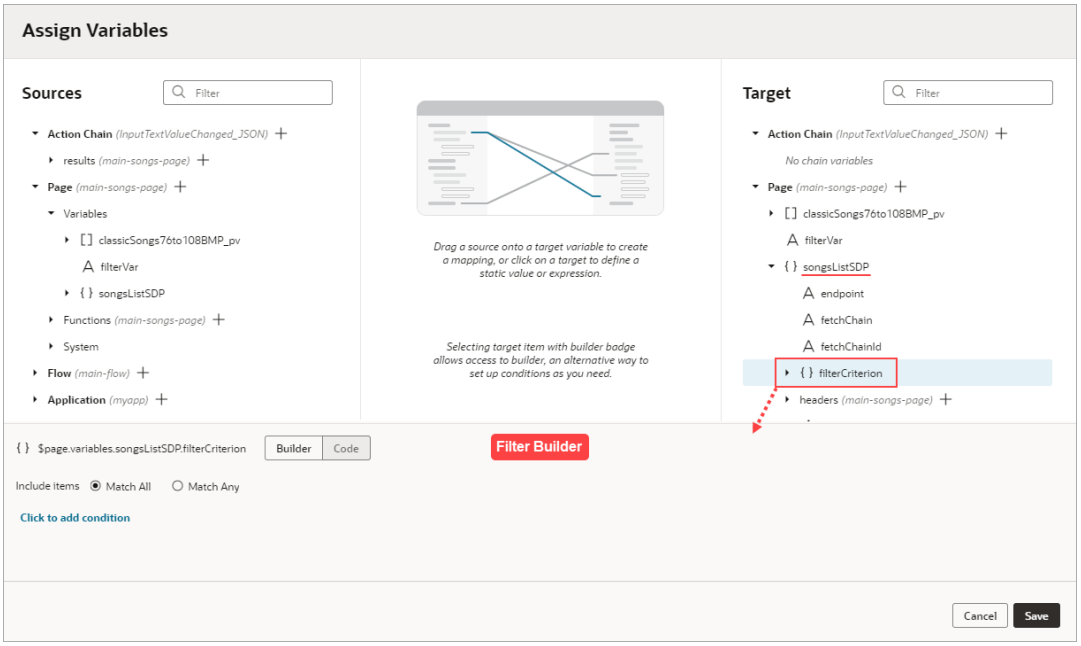
To set the action's Variable property to an SDP's filterCriterion property:

1. In the Properties pane, click **Assign** to open the Assign Variables window:



2. In the Target pane, open the node for the SDP that's connected to your table or list, and select its **filterCriterion** property.
When an SDP's filterCriterion property is selected in the Target pane, the Filter Builder appears for you to create the filter criterion. Alternatively, you can expand the SDP's filterCriterion property in the Target pane and build your filter by specifying values for the `attribute`, `op`, and `value` properties.

   To directly work with the code, click the **Code** button. For details, see Filter Builder's Code Editor.

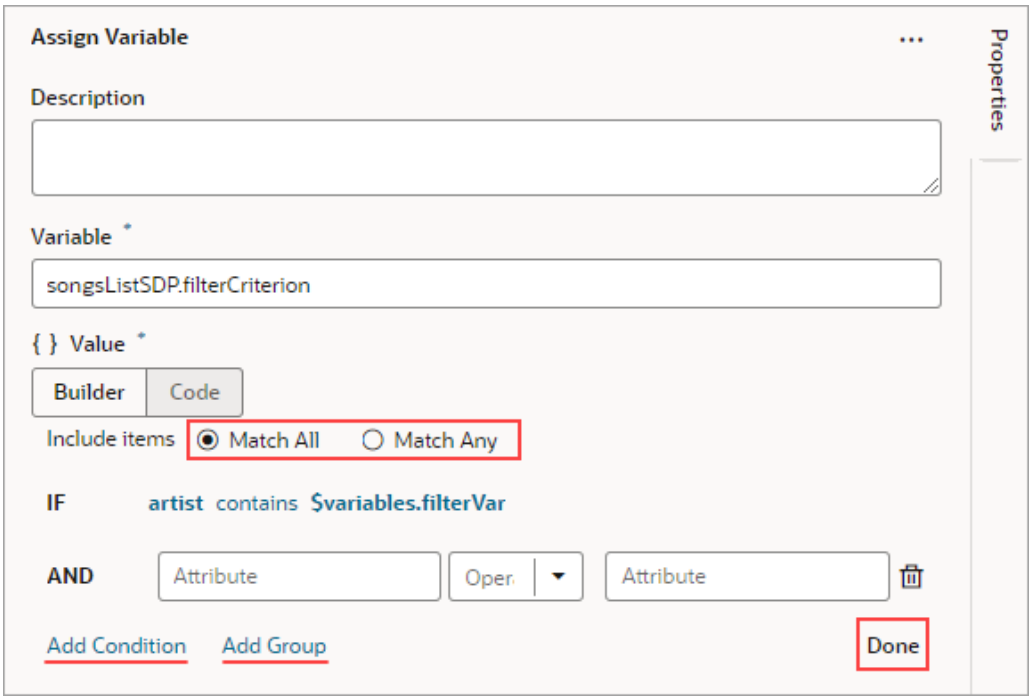


To use the Filter Builder to create the filter criterion for the SDP:

1. Click the Filter Builder's **Click to add condition** link. To create the filter criterion:
   a. For the first Attribute textbox, enter the name of the column (record field) that you want to compare its values against the user's inputted value.
   b. For the Operator drop-down list, select the operator for the criterion.

c. For the second Attribute textbox, select the page variable that was bound to the Input Text component.

d. To add another condition, click the **Add Condition** link to add a condition with an AND or OR operator, or click the **Add Group** link to add a group of conditions that are to be evaluated together (conditions enclosed in brackets). To combine conditional expressions with the AND operator, select **Match All**, and to combine them with the OR operator, select **Match Any**:



e. Click **Done** when you're finished.

Filter Builder's Code Editor

You can use the Filter Builder's Code tab to view and edit the filter's code. After defining a condition on the Builder tab, you will see that the Code tab contains an `attribute`, `op` and `value` property.

Here's an example of a filter with two conditions combined by an AND operator:

```
{
 "op": "$and",
 "criteria": [
```

```
  {
   "op": "$eq",
   "attribute": "name",
   "value": "{{ $page.variables.filterVar }}"
  },
  {
   "op": "$eq",
   "attribute": "id",
   "value": "{{ $page.variables.idVar }}"
  }
 ]
}
```

In this example:

- The Oracle JET operator is "`$eq`" (it must include the dollar sign (“`$`”)).
- The `attribute` property is set to the name of the field (column) that you want to be evaluated against the `value` property.
- The `value` property (`$page.variables.customerListSDP.filterCriterion.criteria[0].value`) is mapped to a page variable (`$page.variables.filterVar`) that holds the value to be evaluated against each field (column) value.

Add a Call Action Chain Action

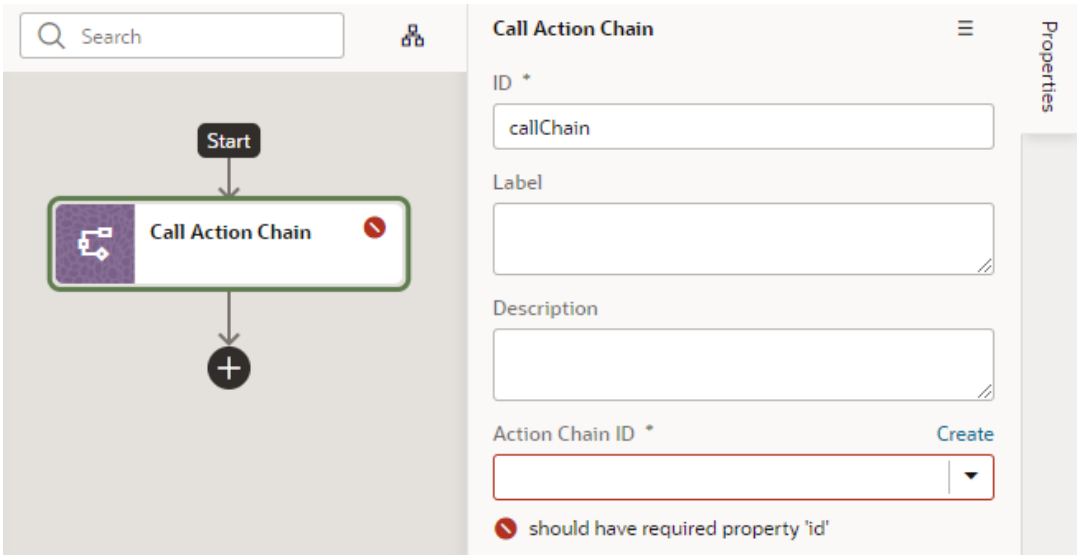
You add a Call Action Chain action to an action chain to start a different action chain. The action can call other action chains defined in the same page, parent flow, or application.

> **Note:**
>
> You can call a JSON action chain from a JavaScript action chain using this action; however, you can't call a JavaScript action chain from a JSON action chain.

To add a Call Action Chain action:

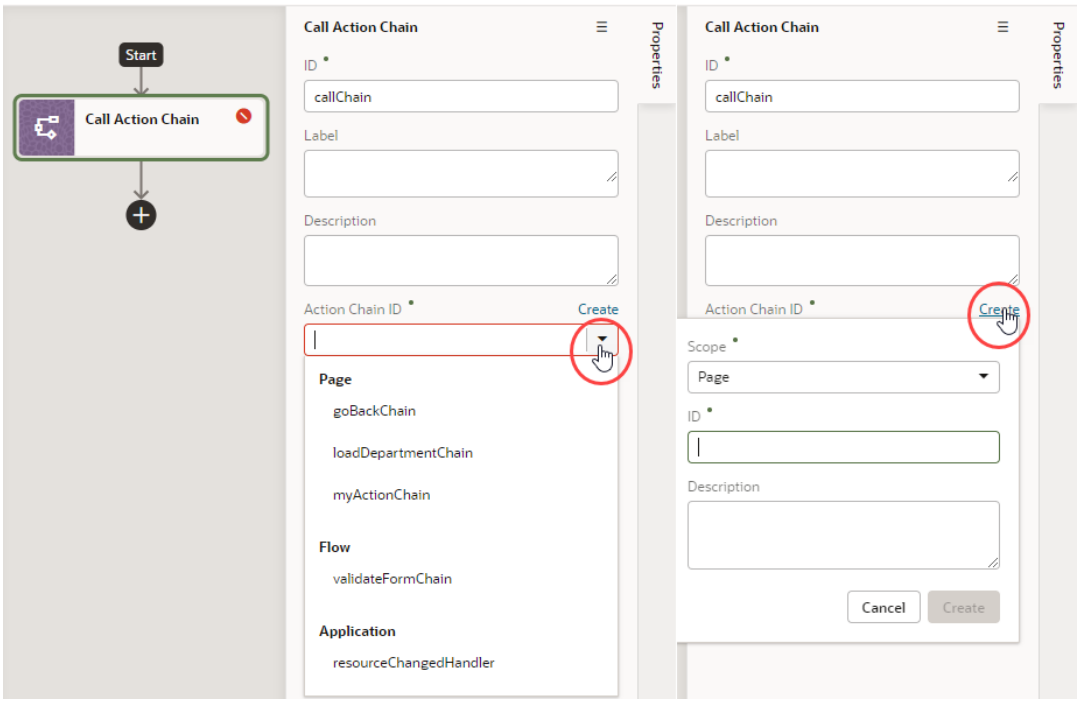
1. Open the Actions editor for the page.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Call Action Chain** from the Actions palette into the action chain.

   You can drag the action onto the Add icon ( ) in the action chain, or between existing actions in the chain. The properties pane opens in the editor when you add the action to the chain.



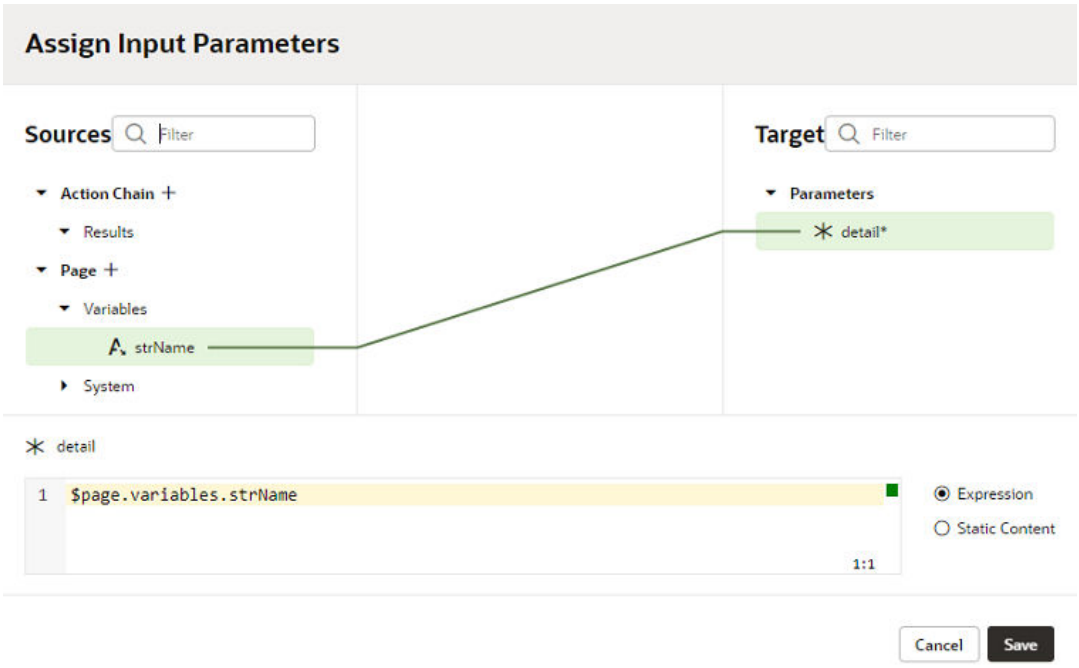
4. Select an existing action chain from the drop-down list of available action chains, or click **Create** to create a new action chain.

   The dialog where you create the new action chain to call displays a drop-down list where you choose where to define the scope of the new action chain (**Page**, **Flow**, or **Application**). Depending on where you are creating the action chain, the drop-down list might have entries for action chains defined in the page, in the current flow, or 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 an entry for Page action chains.



5. Optional: If the action chain that is called requires input parameters, click **Assign** in the Input Parameter section of the properties pane to map the input parameter to a variable.

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. If a suitable variable does not exist, use the **+** icon beside the relevant node (Action Chain, Page, and so on) to create a new variable. Click **Save**.



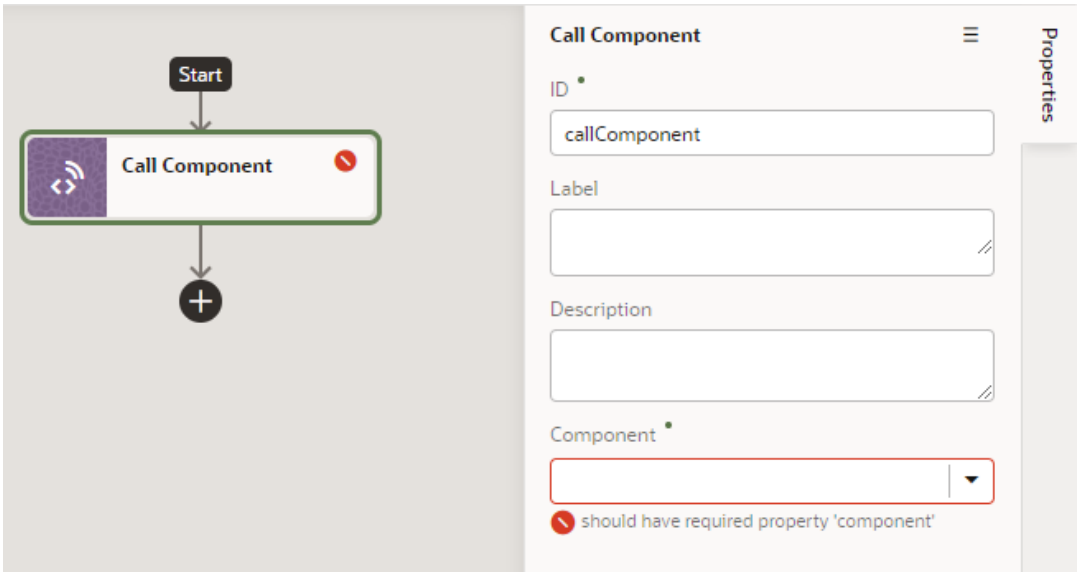
Add a Call Component Action

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

To add a Call Component action to an action chain:

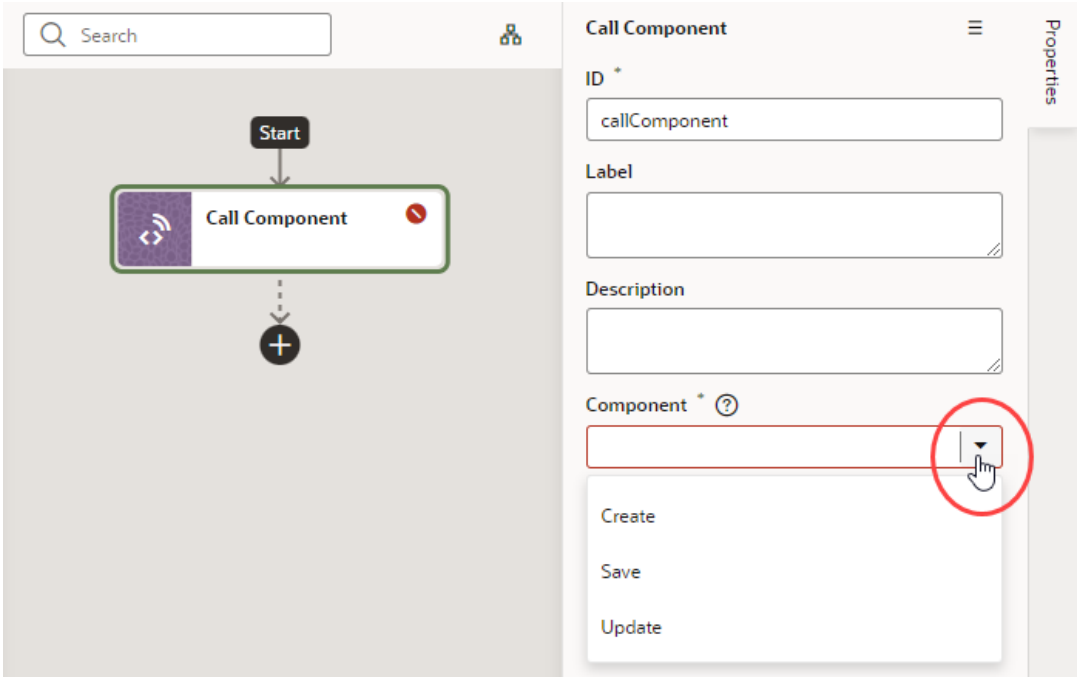
1. Open the Actions editor for the page or application.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Call Component** from the Actions palette into the action chain.

   You can drag the action onto the Add icon ( ) in the action chain, or between existing actions in the chain. The properties pane opens when you add the action to the chain.

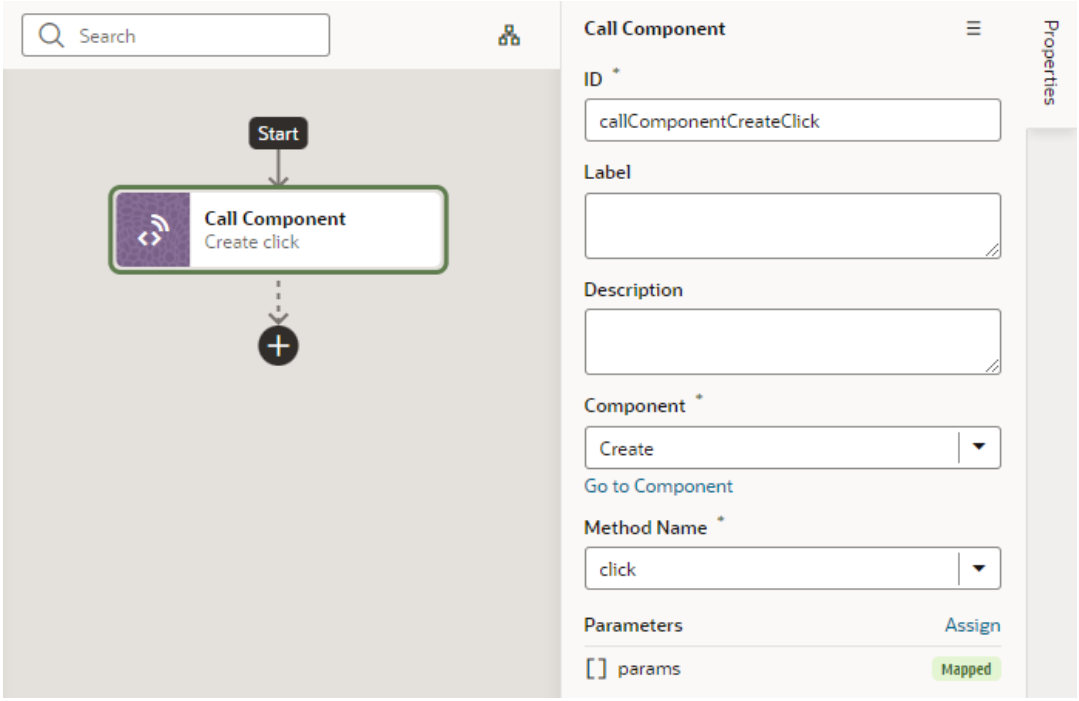


4. In the Properties pane, select the component name in the **Component** drop-down list.

   For example, if your page contains three buttons whose IDs are `Create`, `Update`, and `Save`, you'll see those options available for selection in the drop-down list:



5. With the component selected, select or enter the **Method Name**, then click **Assign** to map the parameters required by the method.



Add a Call Function Action

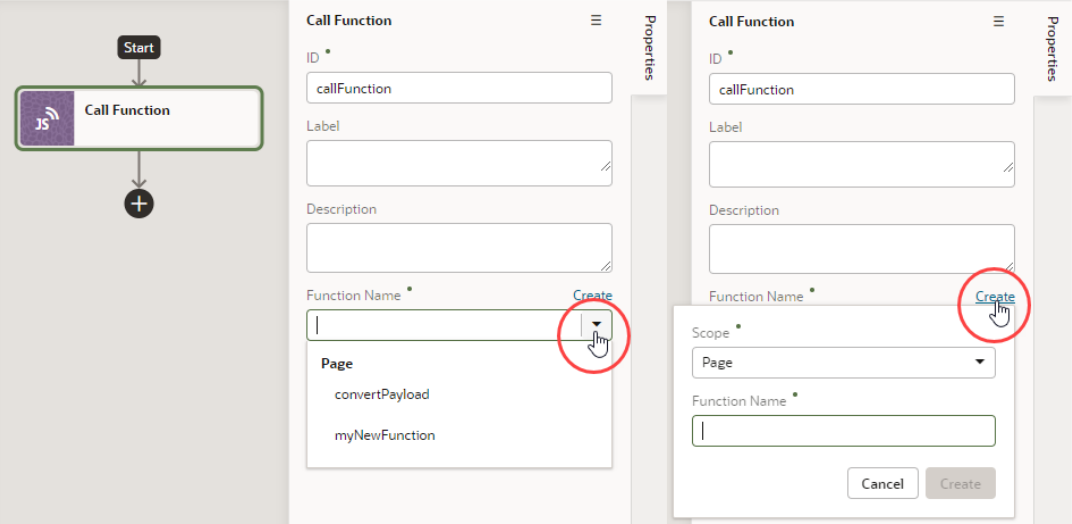
You add a Call Function action to an action chain to call a function defined for the current page, current flow, or the application. You create and edit module functions in the JavaScript editor.

To add a Call 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 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 properties pane opens when you add the action to the chain.
4. In the Properties pane, select an existing function from the drop-down list of available functions, or click **Create** to create a new function.

   You can select functions that are defined for the current page, the current flow, or for the application.



5. Click **Go to Module Function** to go to the JavaScript editor where you write or modify code for the function.
6. Specify any input parameters and return type for the function in the properties pane.

   You can click **Assign** to map variables to the parameters. If a suitable variable does not exist, use the **+** icon beside the relevant node (Action Chain, Page, and so on) to create a new variable.

Add a Call REST Action

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

When you add the Call REST 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. 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. If you did not create the variables before creating the action chain, you can create a variable during the process of creating the action chain; you can also edit the action chain after creating the variables you need.

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.

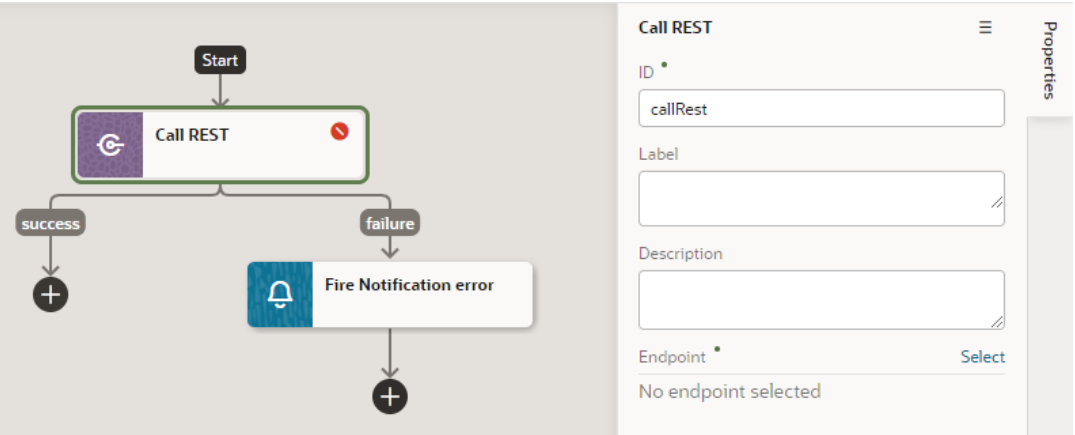
Type of Endpoint	Typical Requirements
POST	When you call a POST endpoint, you will typically need: • Parameters: The page variable for the data needs to be mapped to the parameters of the payload of the POST call. • No input parameter is required.

Type of Endpoint	Typical Requirements
GET	When you call a GET endpoint, you will typically need: • 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: • 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: • 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:

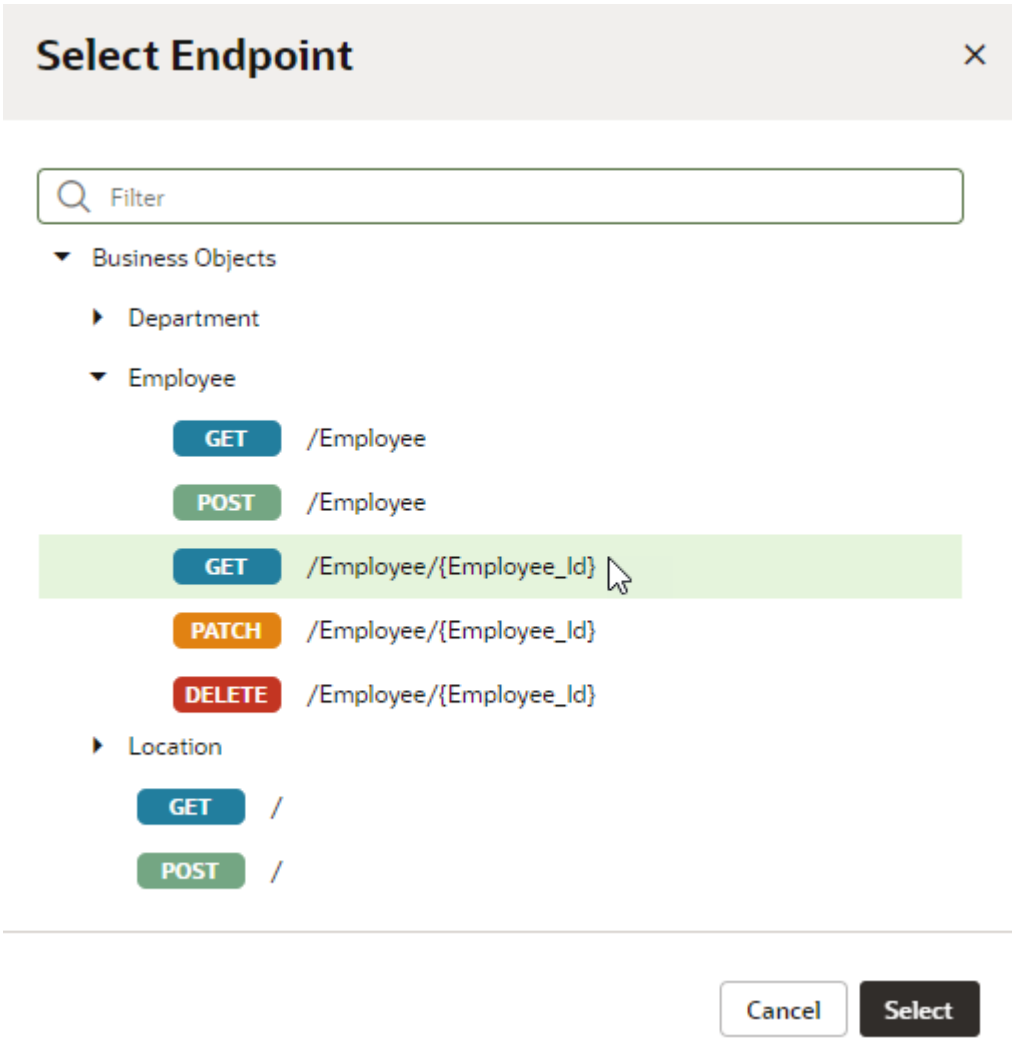
1. Open the Actions editor for the page.
2. Click the action chain in the list to open it in the Action Chain editor.
3. Drag **Call REST** from the Actions palette into the action chain.

   You can drag the action onto the Add icon ( ) in the action chain, or between existing actions in the chain. The properties pane opens when you add the Call REST endpoint action to the action chain.



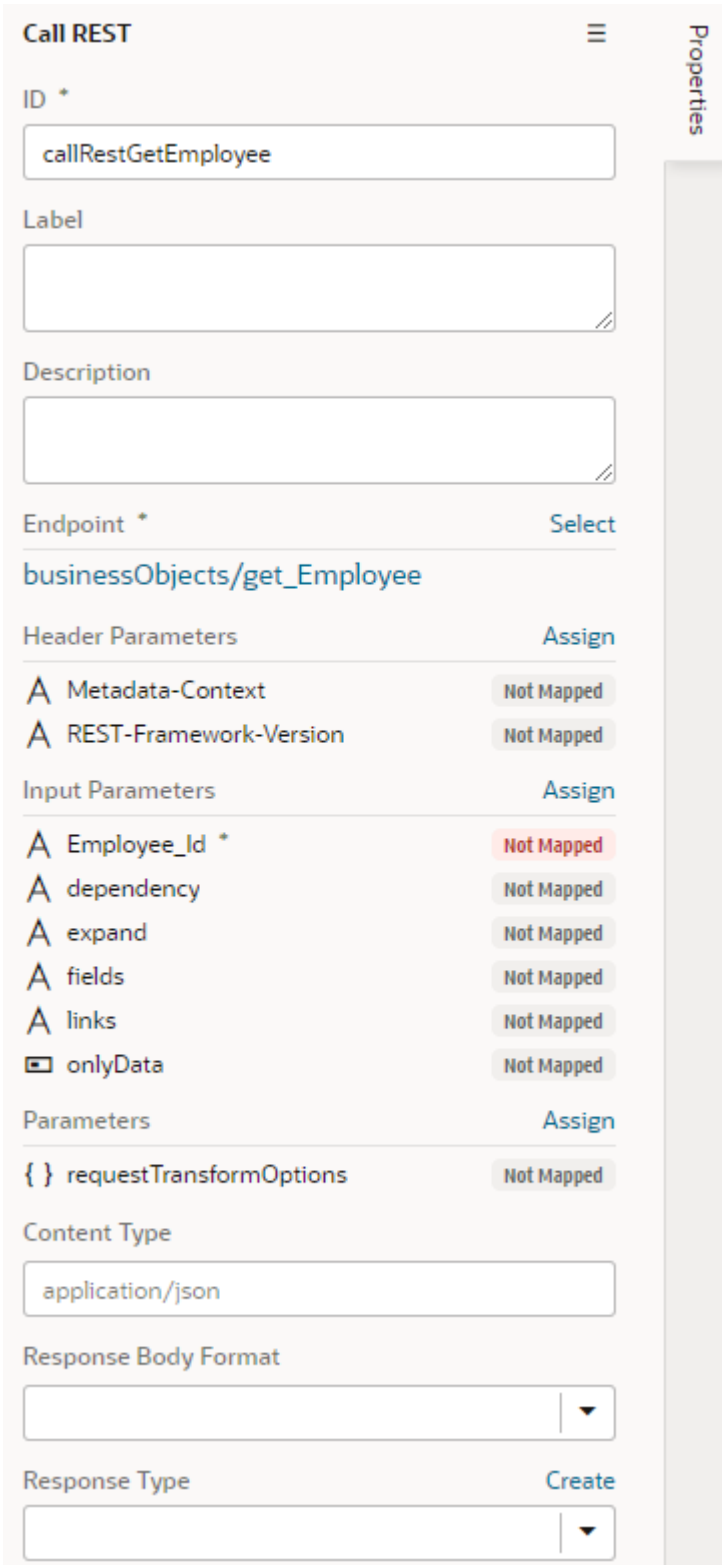
4. Click **Select** beside the Endpoint property in the properties pane.

   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 properties pane.

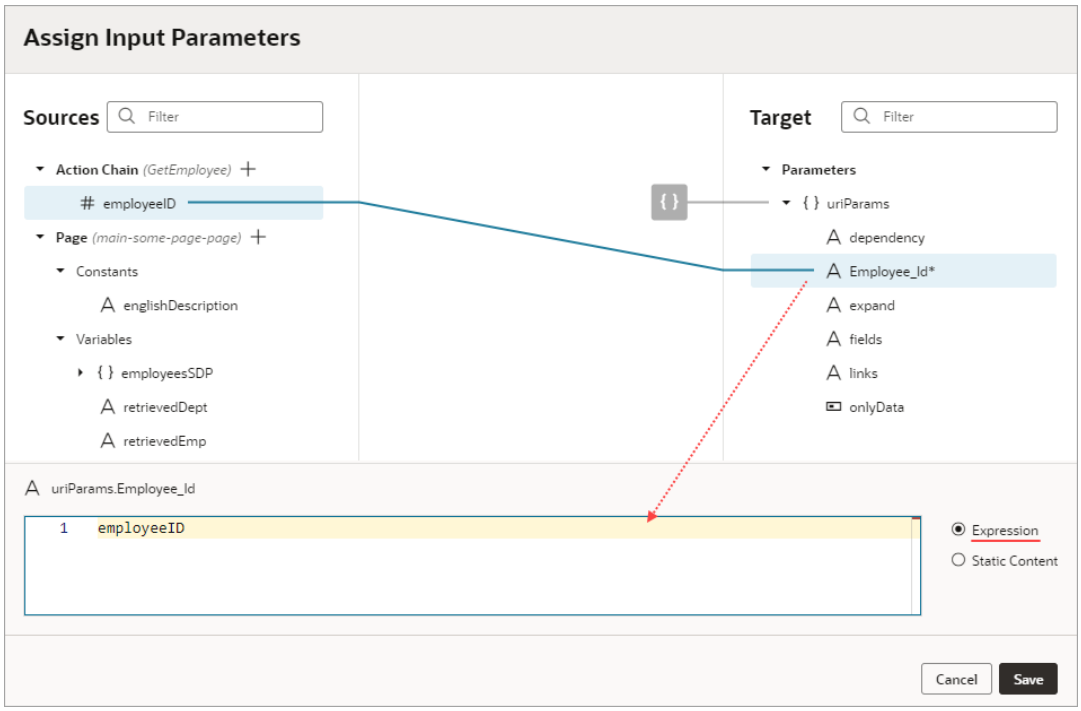
   The properties pane is displayed when the action is selected on the canvas.



7. Optional: If the REST call requires input parameters, click **Assign** next to Input Parameters to map the variable for the input value to the action's parameter. Click **Save**.

   You map variables to parameters in the Assign Input Parameters 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, use the **+** icon beside the relevant node (Action Chain, Page, and so on) to create a new variable.

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 Call Variable Action

You add a Call Variable action to an action chain to call a method on an InstanceFactory variable defined for the current container (flow, page, or application). You can use this action to call any method on the current instance associated with the InstanceFactory variable, including asynchronous ones.

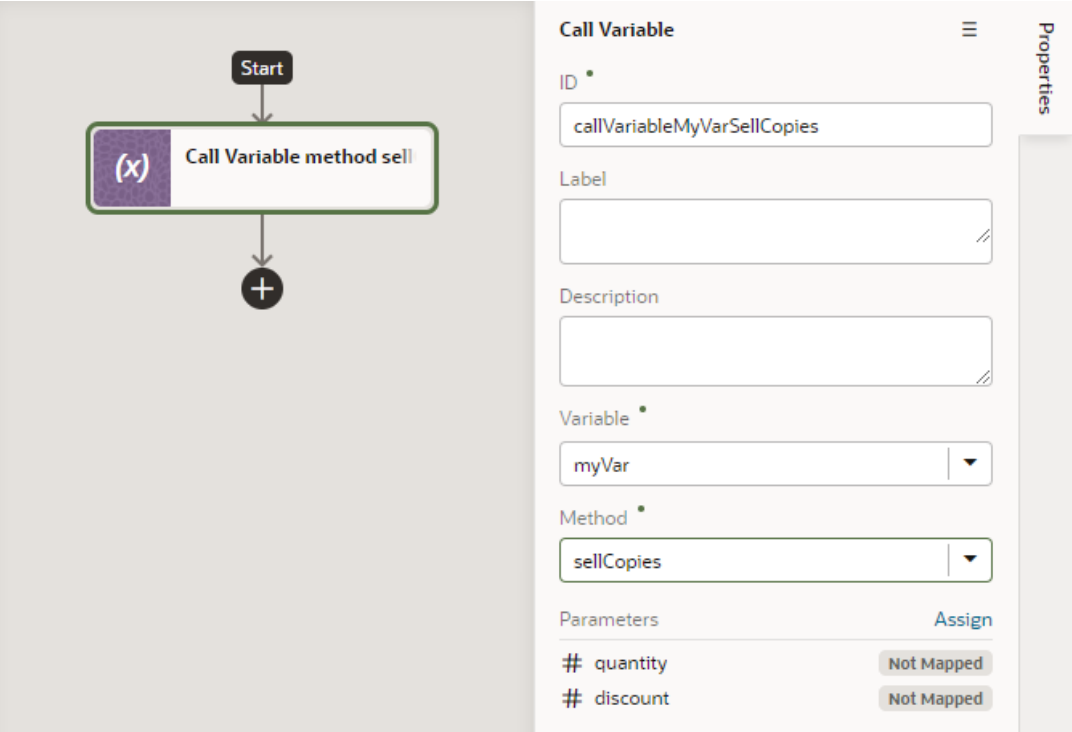
> **Note:**
>
> Because actions are by design synchronous, it will wait for the asynchronous call to resolve before proceeding to the next action in the chain.

Before you use a Call Variable action in an action chain, make sure an InstanceFactory type variable is already defined for the application. See Create a Type From Code.

To add a Call Variable action to an action chain:

1. Open the Actions editor for the application.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Call Variable** 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 properties pane opens when you add the action to the chain.
4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. From the Variables drop-down list, select an InstanceFactory type variable defined for the application.
6. In the Method field, select the method you want to call. The available methods are based on the definition file imported for the type.



7. Click **Assign** to open the Assign Parameters window, then map variables to the action's parameters by dragging the variable in the Sources pane onto the parameter in the Target pane. If a suitable variable does not exist, use the **+** icon to create a new variable.

   The method's return value will be part of the outcome passed to the subsequent chain.

Add a Fire Data Provider Event Action

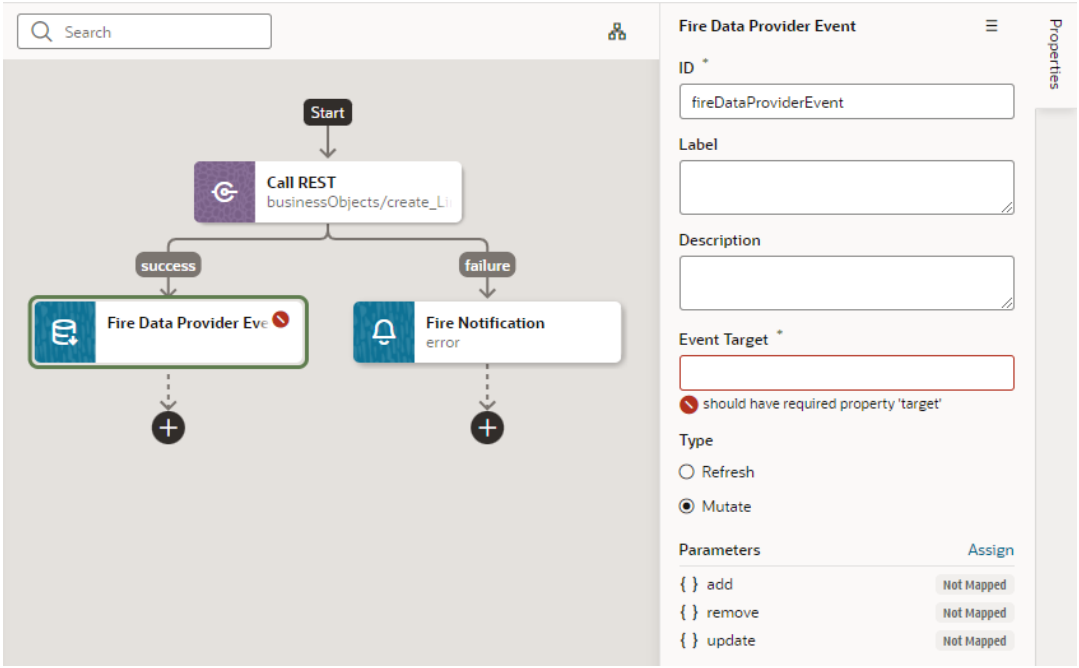
You add a Fire Data Provider Event action to dispatch an event on a data provider to reflect changes to your data. For example, a component using a particular ServiceDataProvider may

need to render new data because new data has been added to the endpoint used by the ServiceDataProvider.

To add a Fire Data Provider Event action to an action chain:

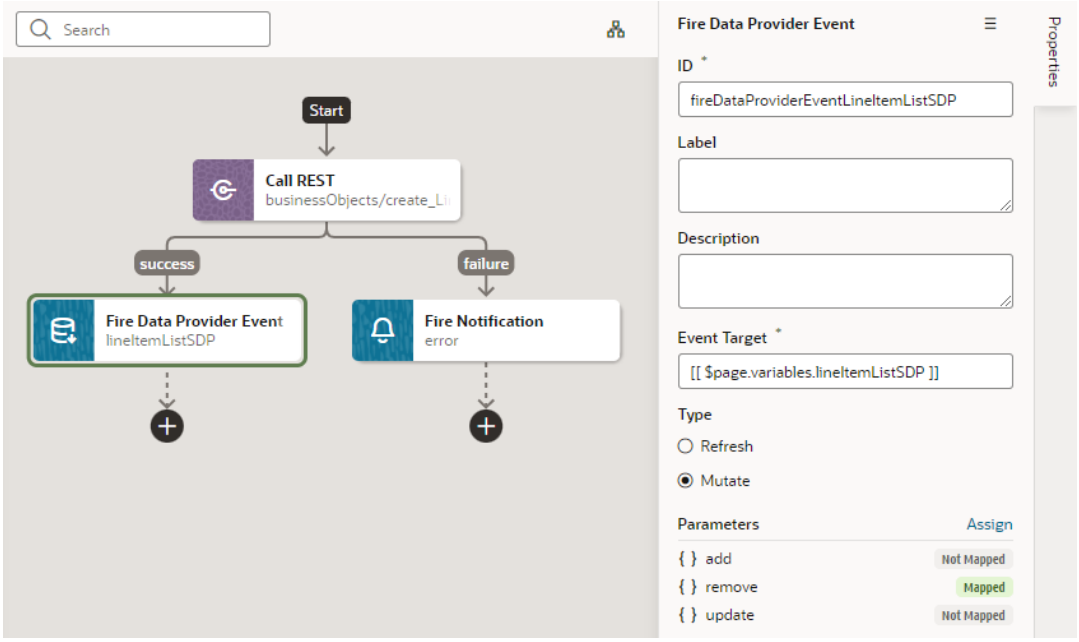
1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Fire Data Provider Event** from the Actions palette into the action chain.

   You can drag the action onto the Add icon ( ) in the action chain, or between existing actions in the chain. The Properties pane opens when you add the action to the chain.



4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. Set the target of the event. Usually, this is a variable of type ServiceDataProvider or ArrayDataProvider.
6. Select the type of event you want to dispatch:
   - **Refresh**: Indicates a refresh event needs to be dispatched to the data provider identified by the target.
   - **Mutate**: Indicates a mutation event needs to be dispatched to the data provided identified by the target. Generally, a mutation event is raised when items have been added, updated, or removed from the data that the data provider represents.
7. If you chose a Mutate event, click **Assign** to map variables for the add, remove, and update operations.

   A mutation event can include multiple operations (add, update, remove) as long as the id values between operations do not intersect.



Add a Fire Event Action

You add a Fire 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 Events editor displays a list of the custom events available in the context.

To add a Fire 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 Event** from the Actions palette into the action chain.
4. In the Properties pane, select an existing custom event from the drop-down list of available custom events, or click **Create** to create a new custom event.

   The drop-down list displays the custom events that are available in the current context.
5. Click **Assign** to open the Mapper and define the event's payload.

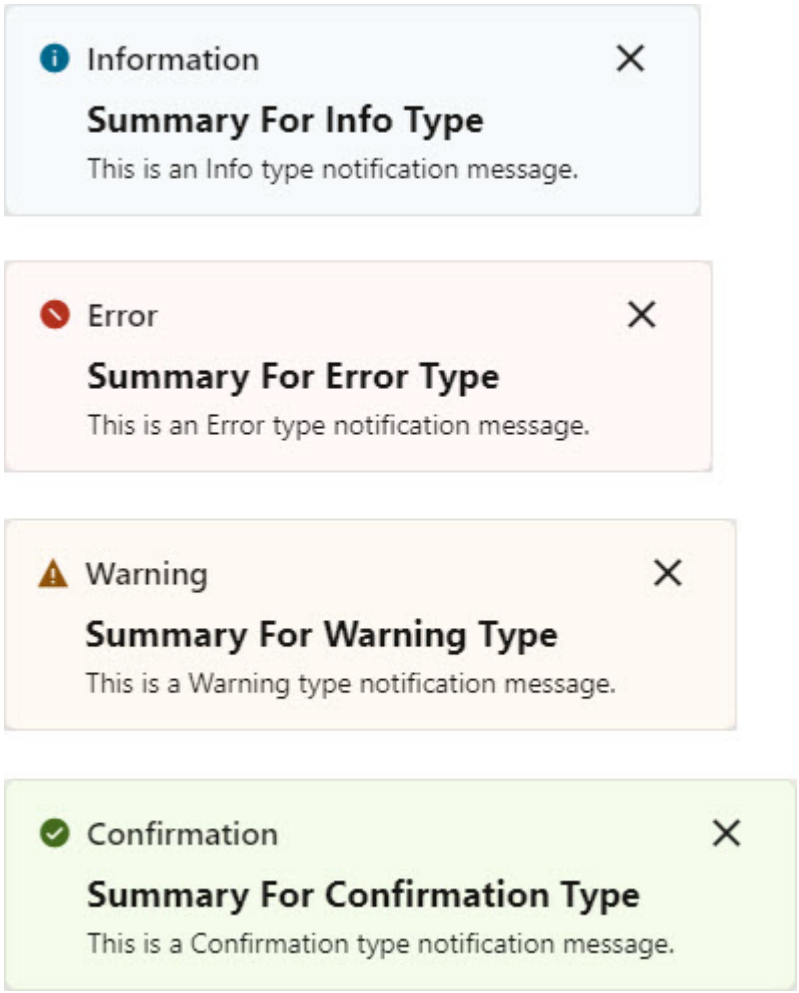
   The event payload depends upon how the custom event is defined. You can use the Mapper to map the payload to a source, such as a page variable, or define a specific value or expression.

   If you need to define the variable, use the **+** icon to open a dialog where you can define a variable for the artifact (action chain, page, flow, or application).

Add a Fire Notification Action

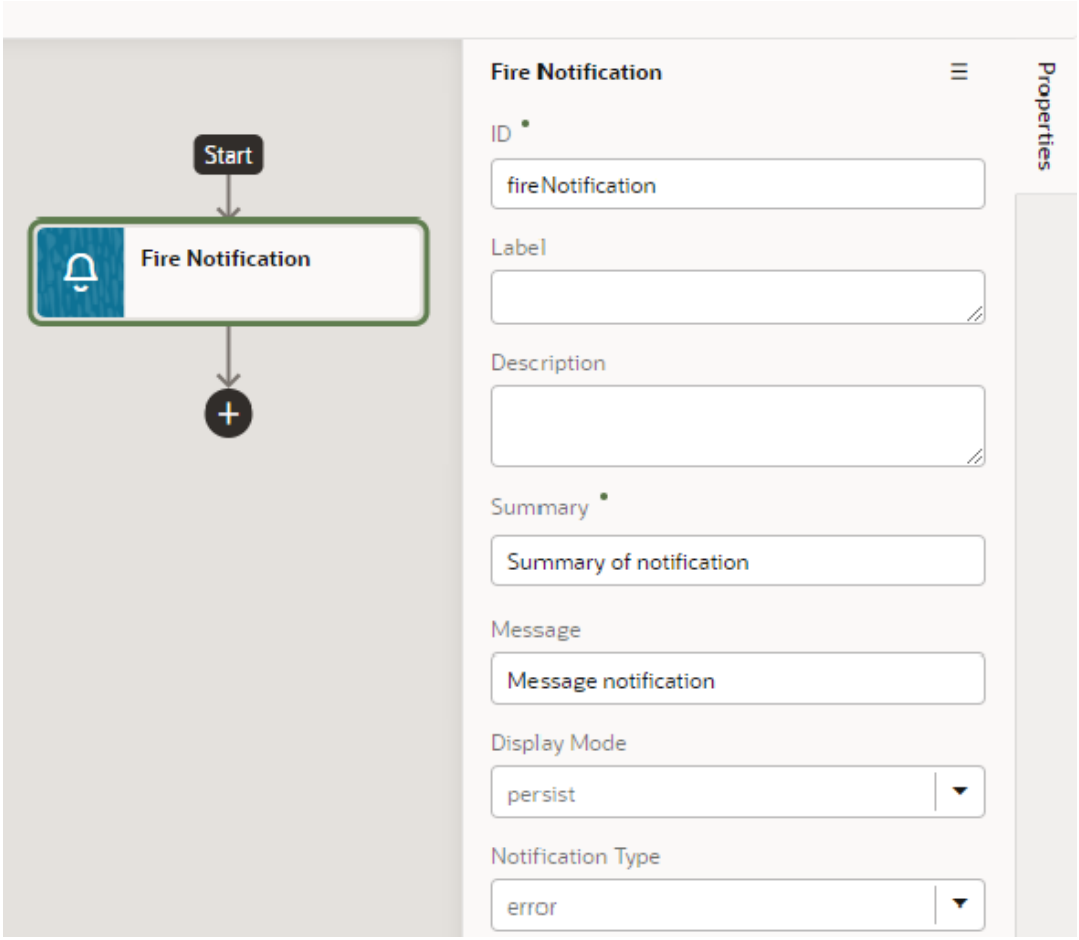
You add a Fire Notification action to display a notification to the user in the browser window.

There are four types of notifications: Info, Error, Warning and Confirmation. The notifications display a summary and a message underneath:



To add a Fire Notification action:

1. Drag **Fire Notification** from the Actions palette onto the empty canvas, onto an Add icon ( ), or between existing actions in the chain. The Properties pane displays the action's parameters:



2. Update the **ID** field in the Properties pane to make the action more easily identifiable.
3. Enter a summary of the notification in the **Summary** field.
4. Enter the message you want to display in the **Message** field.

   The message can be a static string (`The name was updated.`) or can contain variables (`{{ 'Could not create new Contacts: status ' + $chain.results.createContacts.payload.status }}`).
5. For **Display Mode**, specify how the notification is to be dismissed. Choose **Transient** for the notification to go away on its own after a few seconds, or **Persist** for the notification to stay until the user closes it.
6. Select a **Notification Type** to specify the look of the notification window.
7. Select the **Target** to specify where you want the event to be fired. Choose `current` to have the event fire where it is executed and then all the way up the hierarchy. Choose `leaf` (or leave the setting undefined) to have the event fire at the bottom of the hierarchy.

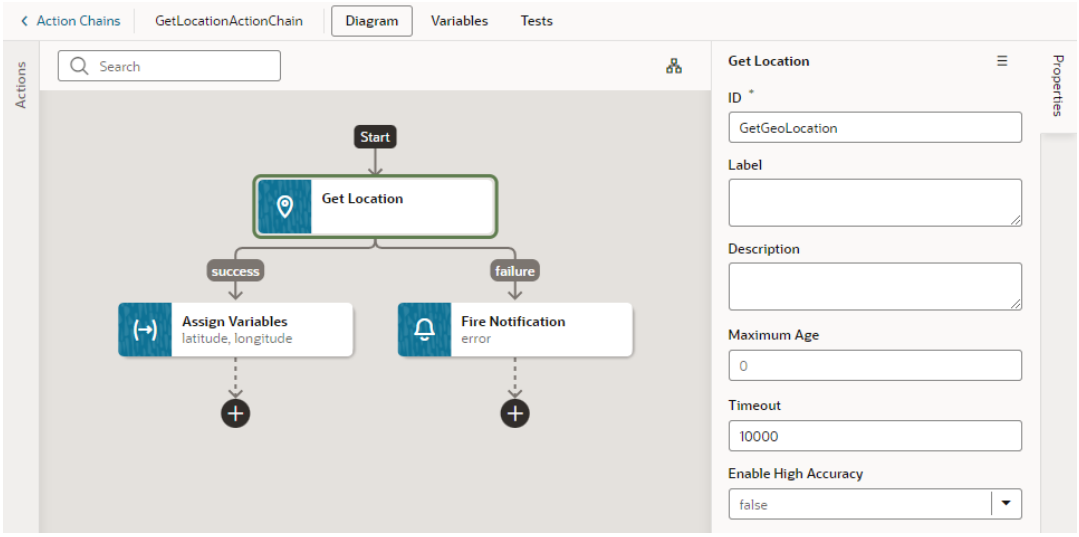
Add a Get Location Action

You add a Get Location action to get a user's live location. This action requires the user's consent. As a best practice, it should only be fired on a user gesture, so users can associate the system permission prompt for access with the action they just initiated.

To add a Get Location action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Get Location** 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 Properties pane opens when you add the action to the chain.



4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. Set the Maximum Age (in milliseconds) of a possible cached position that is acceptable to return. If set to `0` (default), 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.
6. Set the Timeout value, representing the maximum length of time (in milliseconds) that the device is allowed to take in order to return a position.
7. Set the Enable High Accuracy value that indicates whether 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` (default), the device can save resources by responding more quickly or using less power. For mobile devices, you should set this to true in order to use GPS sensors.

Add a Reset Variables Action

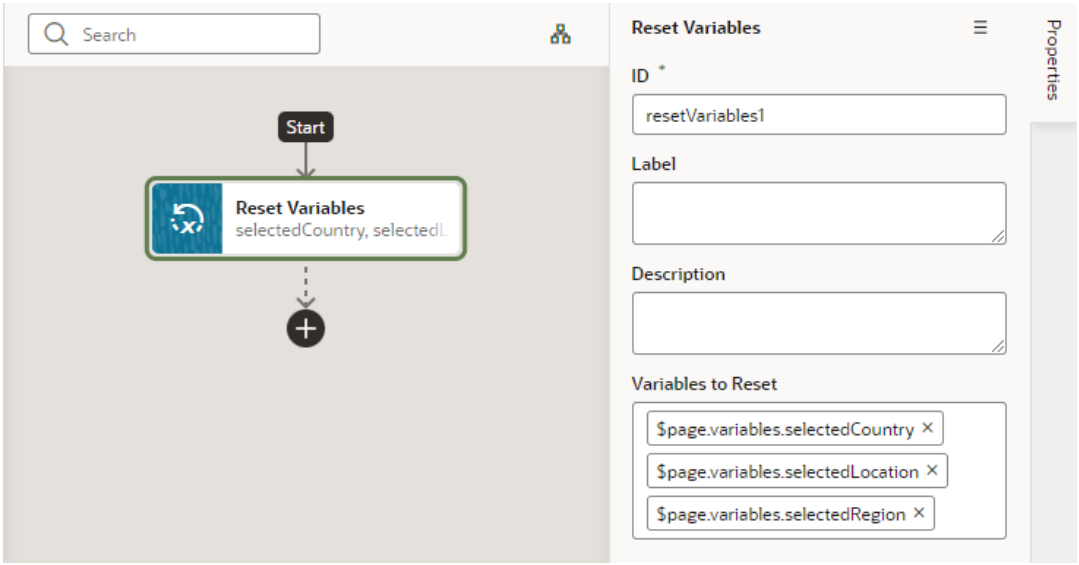
You add a Reset Variables action to reset variables to their default values, as specified in the variable definitions.

To add a Reset Variables action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Reset 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 Properties pane opens when you add the action to the chain.

4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. From the Variables to Reset list, select the variables you want to reset.



Add a Login Action

You can add the Login action to any page component to launch the mechanism that enables your users to sign in to an application.

To add a login action 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 the **Login** action from the Actions palette into the action chain to the Add icon ( ) in the action chain.
4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. In the Return Path field, specify the path of the page to go to when login is successful. If a value isn't defined, the user will be taken to the application's default page. For more information, see Login Action in the *Oracle Visual Builder Page Model Reference*.

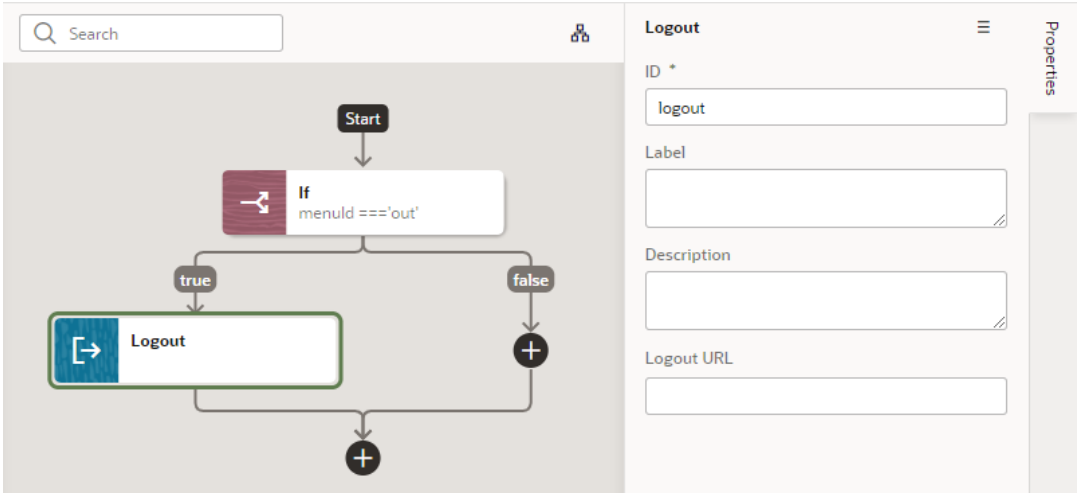
Add a Logout Action

You can add the Logout action to any page component to launch the logout mechanism that enables your users to sign out of an application.

To add a logout action 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 the **Logout** action 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 Properties pane opens when you add the action to the chain:



4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. If you're using an external identity provider, enter the provider's logout endpoint URL in the **Logout URL** field, something like `https:***/oam/server/logout?end_url=https://****/oamwebsso/logout-success.jsp`.

   If you're using IDCS for user authentication, you don't need to specify the logout URL. In this case, the URL defined by the default Security Provider configuration is used. After the user is logged out, the application continues to the default page of the application.

   For more information, see Logout Action in the *Oracle Visual Builder Page Model Reference*.

Add a Scan Barcode Action

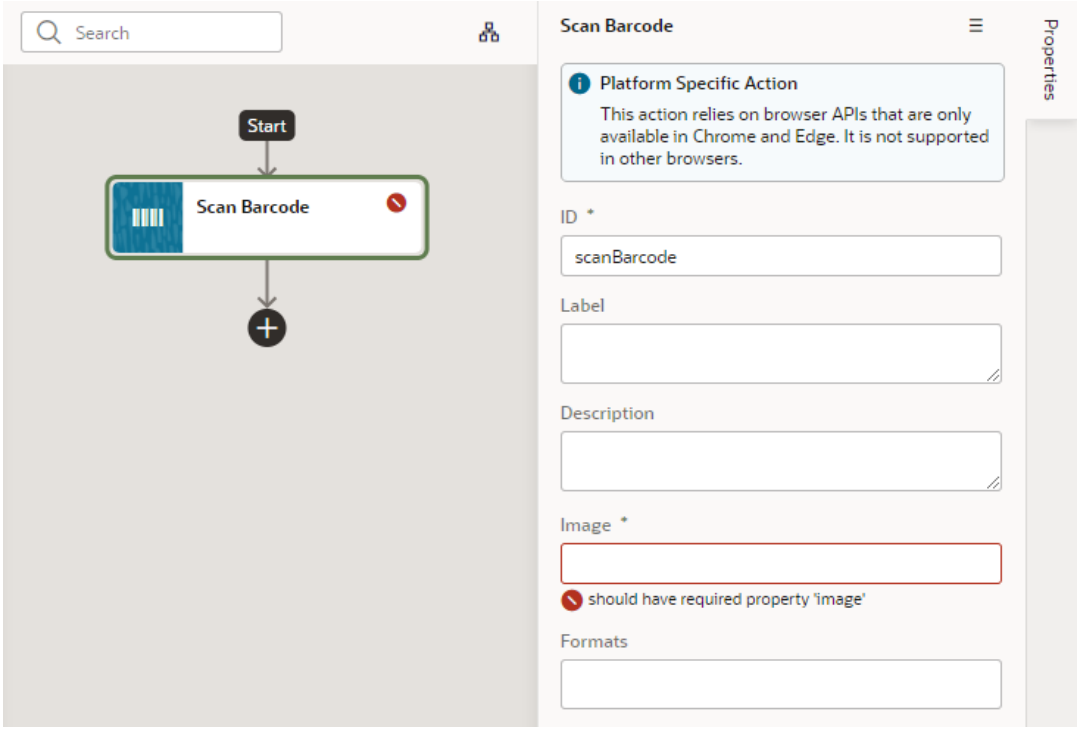
You can add the Scan Barcode action when you want your application to decode information such as URLs, Wi-Fi connections, and contact details from QR codes and barcodes.

> **Note:**
>
> The Scan Barcode action relies on browser APIs and is supported only on Chrome for Visual Builder apps.

To add a scan barcode action 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 **Scan Barcode** 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 Properties pane opens when you add the action to the chain.



4. Specify the action's properties in the Properties pane:
   a. Update the **ID** field to make the action more easily identifiable.
   b. In the **Image** field, enter an image object (either a CanvasImageSource, Blob, ImageData, or an `<img>` element) to decode.

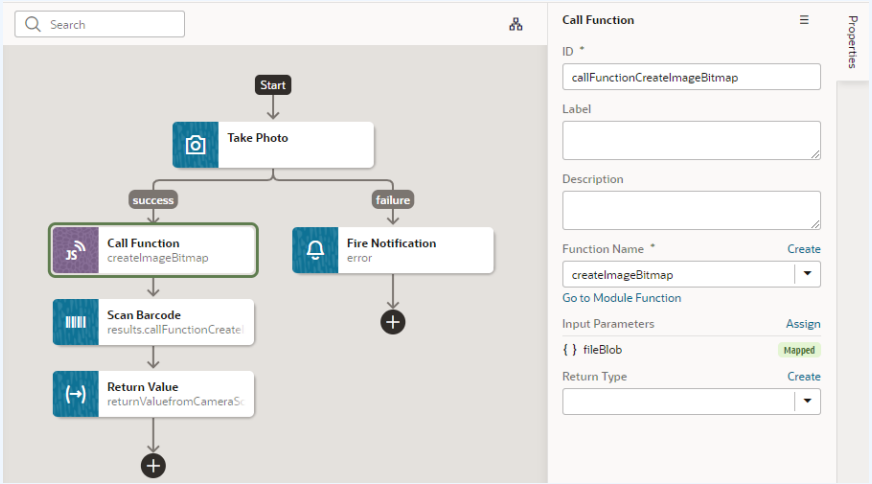
Note:

If you're using the Take Photo Action or the camera component to pass a Blob to the Scan Barcode action, you might run into the `Failed to execute 'detect' on 'BarcodeDetector` error. To get around this error, convert the Blob to an ImageBitmap before passing it to the Scan Barcode action. For example:

i. Add a function to do the image conversion, something like:

```
// Convert Blob to ImageBitmap
//
PageModule.prototype.createImageBitmap =
function(fileBlob) {
   return window.createImageBitmap(fileBlob);
};
```

ii. Add a Call Function action to the action chain, similar to:



iii. Pass the converted ImageBitmap as the Image parameter for the Scan Barcode action, for example:

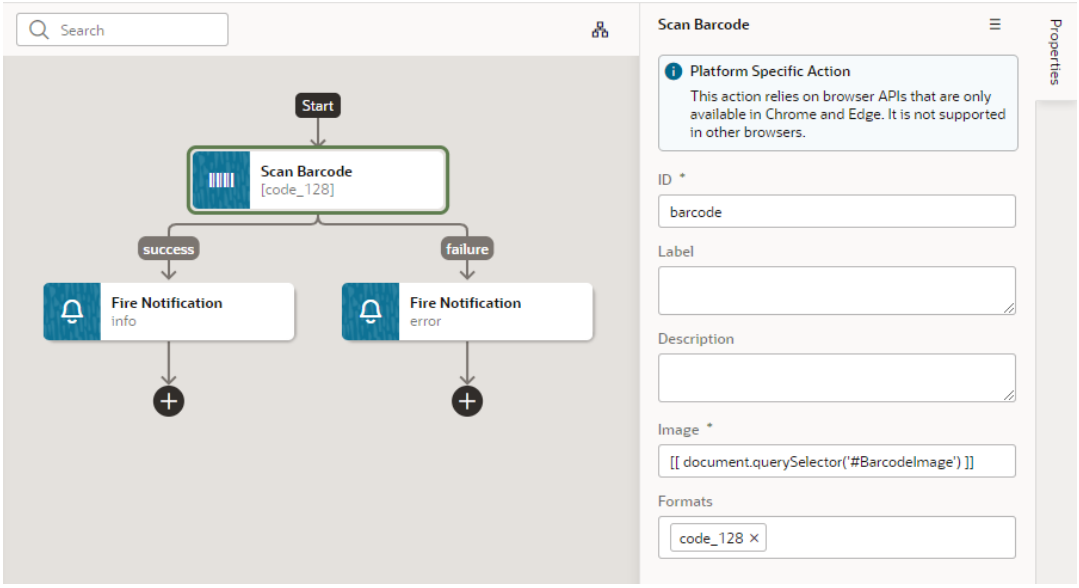
```
[[ $chain.results.callFunctionCreateImageBitmap ]]
```

c. Optional: In the Formats field, select the barcode formats you want the browser to search for.

Barcode formats unlock a variety of use cases. QR codes can be used for online payments, web navigation, or social media connections, aztec codes can be used to scan boarding passes, and shopping apps can use EAN or UPC barcodes to compare prices of physical items.

If `Formats` is not specified, the browser will search all supported formats, so limiting the search to a particular subset of supported formats may provide better performance.

One option when using the Scan Barcode action is to use `document.querySelector` to get the image, as shown here where the first image with the ID `BarcodeImage` will be returned:



Add a Take Photo Action

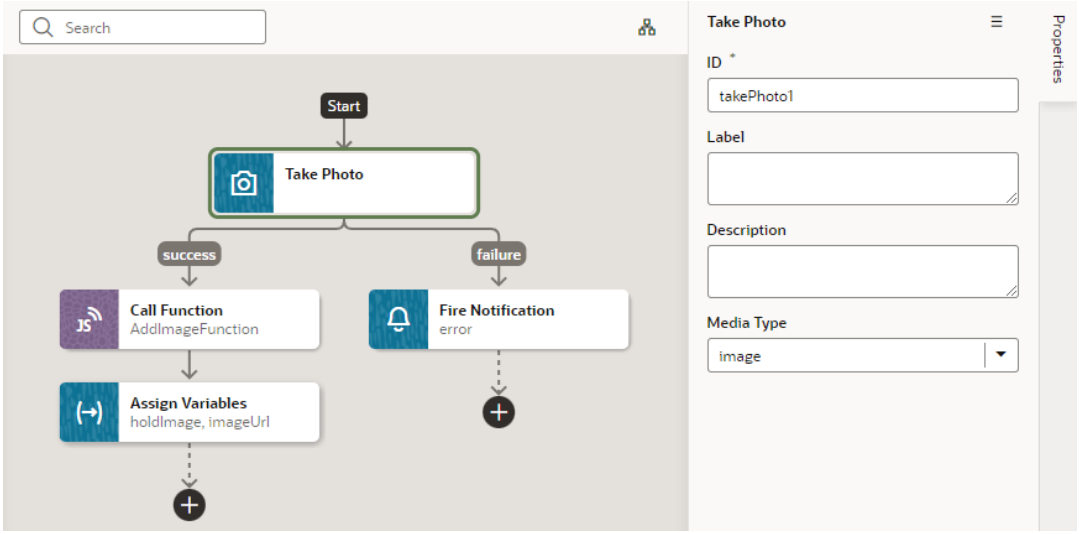
When working with PWA-enabled applications, you add a Take Photo action to access the camera or the image gallery on the device where your application is installed. For PWAs on Android and iOS, this action prompts user with multiple options, such as Camera, Browse, or Like.

To add a Take Photo action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Take Photo** from the Actions palette into the action chain.

   You can drag the action onto the Add icon ( ) in the action chain. The Properties pane opens when you add the action to the action chain.
4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. Select a value for the Media Type property: either **image** (default) or **video**. If Media Type is set to **video**, options to record video using the Camera or to select video files will be provided for PWA apps on iOS and Android.

   Here's an example of a Take Photo action (with the Media Type set to **image**) that calls a custom module function, then maps its output to variables.



Add a Share Action

You add a Share action to share content with other applications, such as Facebook, Twitter, Slack, and SMS, by invoking the native sharing capabilities of the host platform. This action requires the user's consent. As a best practice, it should only be fired on a user gesture, such as a button click.

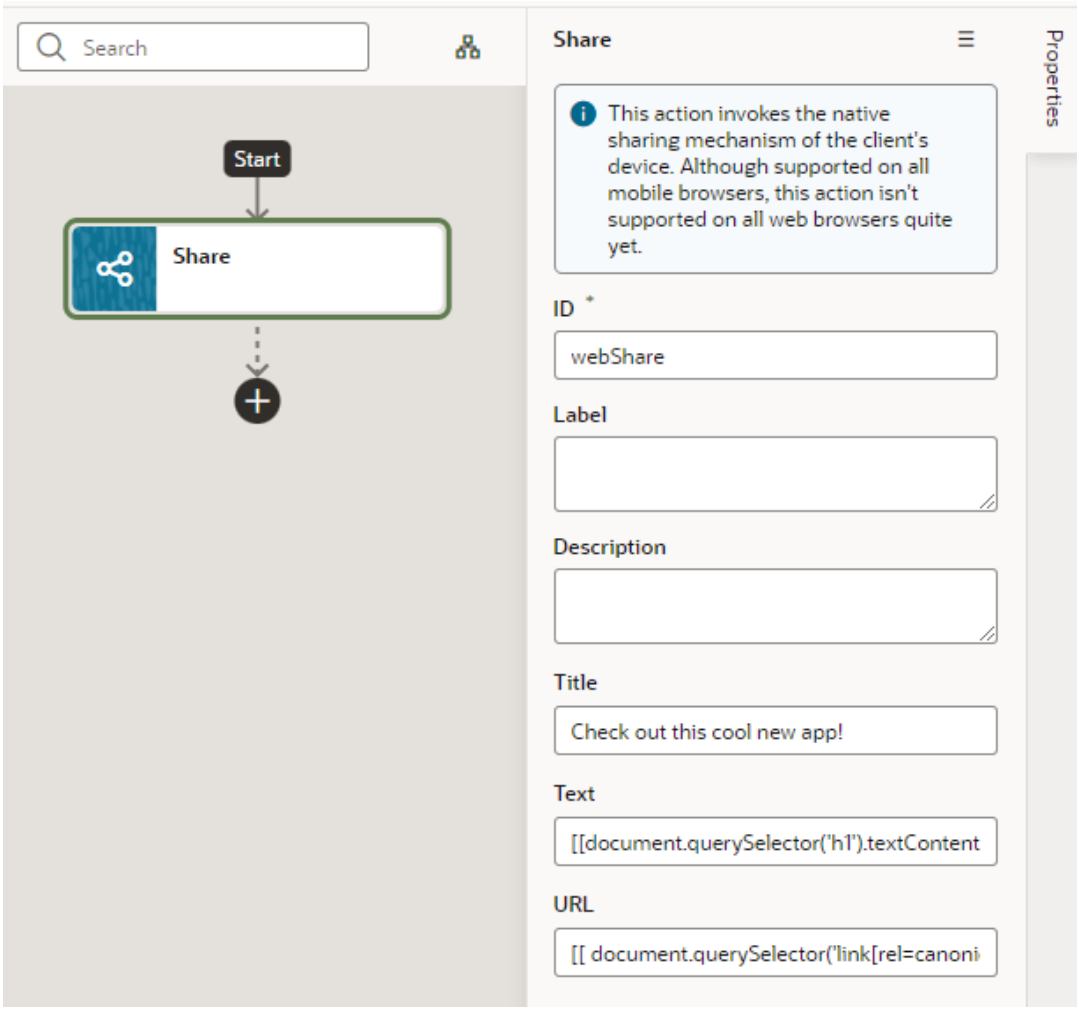
> **Note:**
>
> Web apps require the web browser running the app to support the Share action. Currently, not all browsers support this native feature.

To add a Share action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Share** 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.
4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. Configure the Title, Text, and URL. All parameters are individually optional, but at least one parameter must be specified. Any URL can be shared, not just those under the website's current scope. Text can be shared with or without a URL.
   a. In the Title field, enter the title of the document to be shared.
   b. In the Text field, enter the text that will form the body of the message being shared.
   c. In the URL field, enter the URL that refers to a resource being shared.

   Here's an example that shares the current page's title and URL:



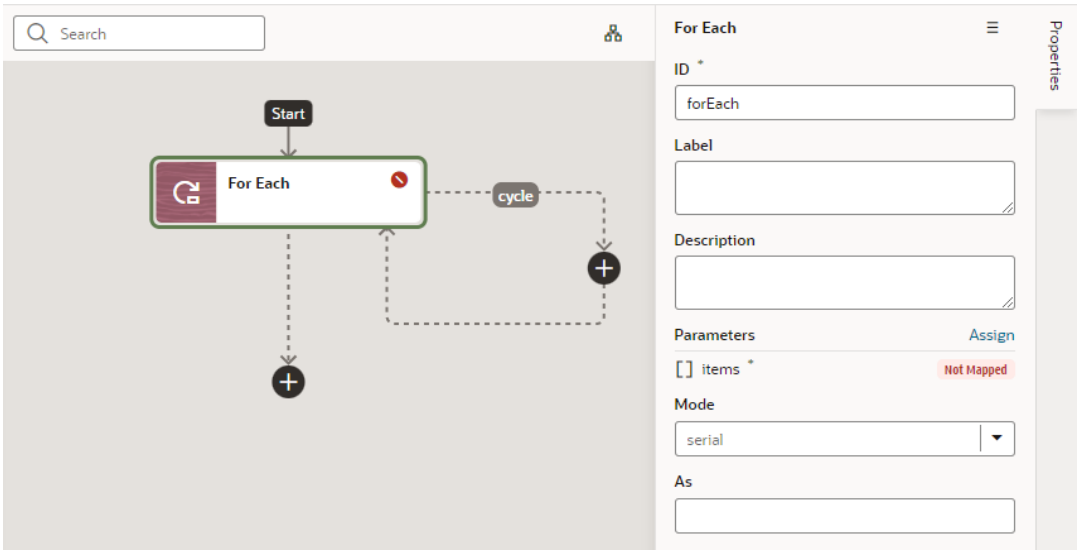
Add a For Each Action

You add a For Each action to execute another action for each item in an array. The action in the loop will be executed once for each item in the array.

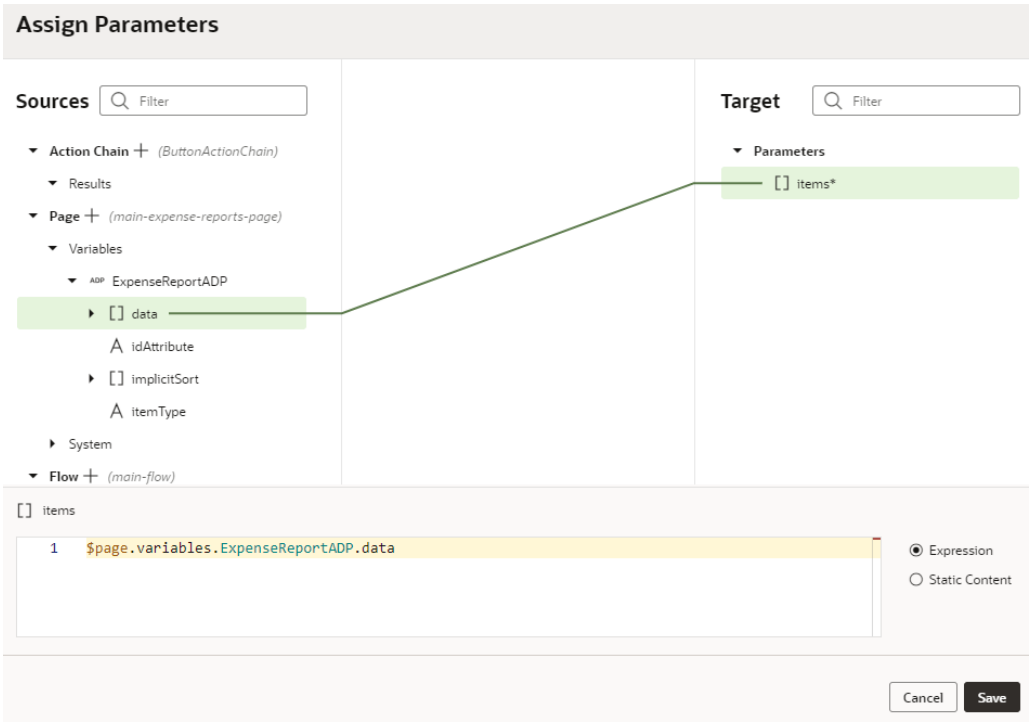
To add a For Each action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **For Each** from the Logic section of 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.



4. Configure the action's properties in the Properties pane:
   a. Update the **ID** field to make the action more easily identifiable.
   b. Click **Assign** next to Parameters to set up an expression for the `items` parameter that evaluates to an array, for example, `$page.variables.ExpenseReportADP.data`:

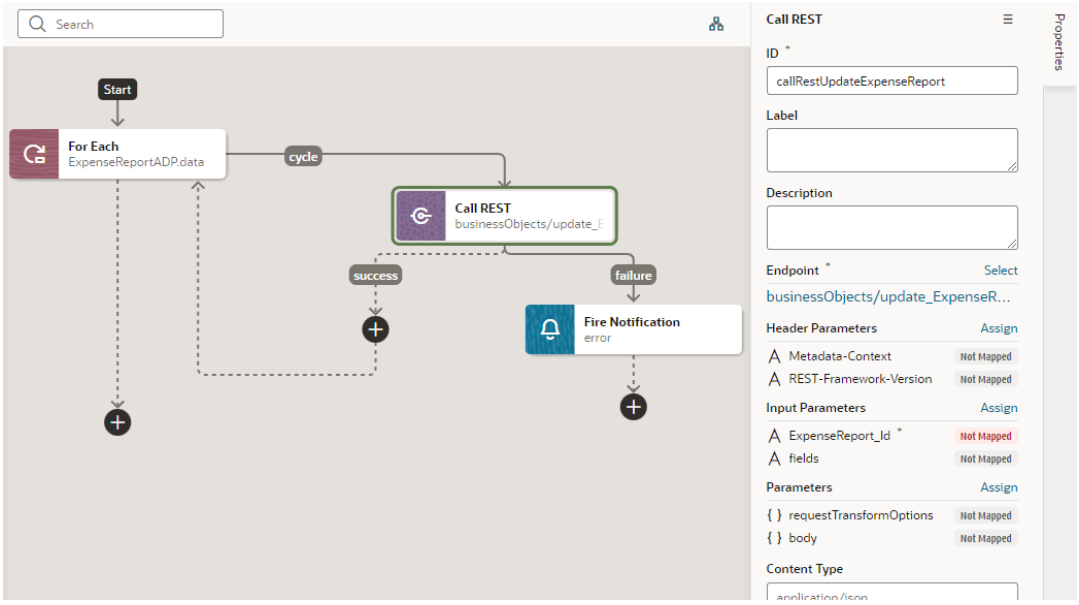


   The For Each action uses 'items' and the 'actionId' and adds a `$current` context variable for the called action to access the current item. You can inject additional properties into the available contexts for the called action to reference in its parameter expressions (as we'll see in subsequent steps).

   c. If you want to use your own context name, enter an alias for `$current` in the **As** field, for example, `foo`. This alias can then be referenced in nested called actions.

d. Define whether your called actions must run serially (default) or in parallel. Regardless of the mode, the For Each action will not complete until the actions for each item in the `items` array are complete.

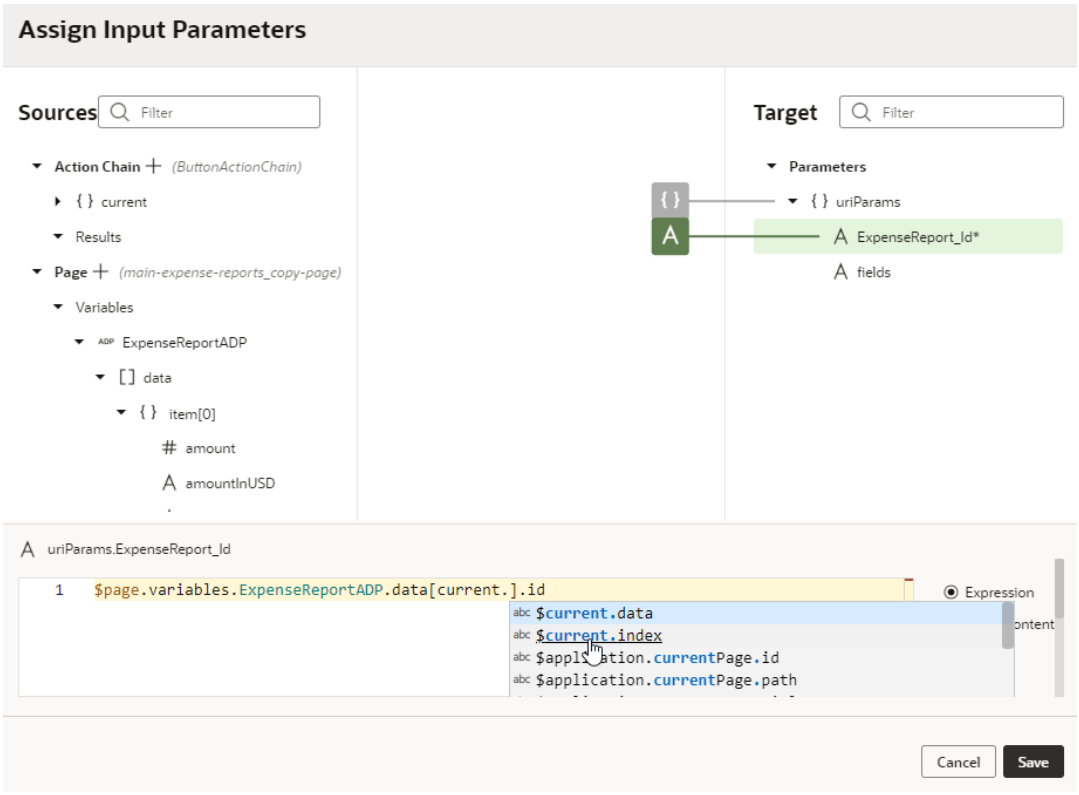
5. Now click the Add icon ( ) inside the cycle loop and add the action you want to loop over the array. Here's an example that adds an action to call the `PATCH /ExpenseReport/{ExpenseReport_Id}` REST endpoint.



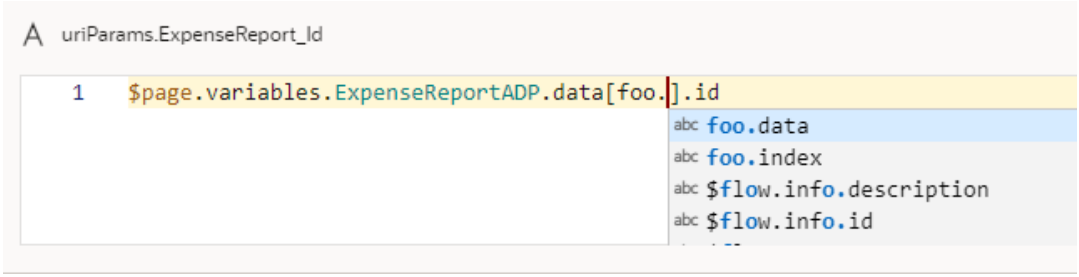
When assigning the results of the REST call to a variable, you can use the following parameter expressions for the called action:

Parameter Name	Description
`$current.data`	The current array item.
`$current.index`	The current array index.
alias`.data`	An alternate syntax for `$current.data`, which allows a reference to `$current` from nested contexts.
alias`.index`	An alternate syntax for `$current.index`, which allows a reference to `$current` from nested contexts.

For example, to pass the ID of the current expense report in the loop, you can use `$current.index` in the source expression:



If you defined a context alias, for example, `foo`, you'd be able to create expressions that reference `foo.data` and `foo.index`:



The outcome of the action is either "success", with an array containing the return value of the last action's results or "failure" if there is some exception/error.

6. As a final step, click the Add icon ( ) to add an action (for example, a Fire Notification action) where the For Each action's loops ends.

Add an If Action

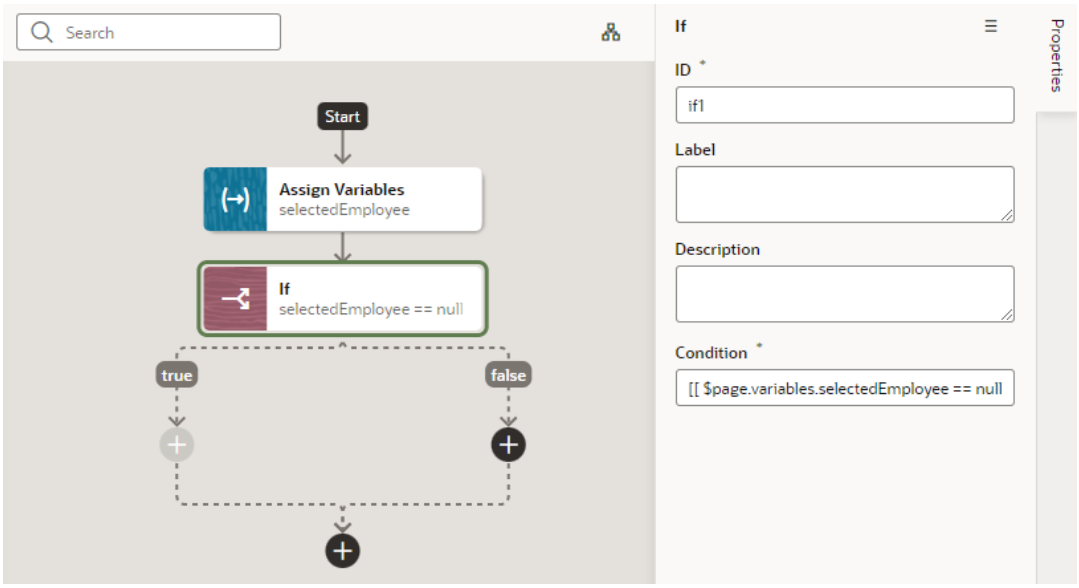
You add an If action to evaluate an expression based on conditions and return a 'true' outcome if the expression evaluates to true, and a 'false' outcome otherwise. You use this action typically to execute custom logic, say to validate data before you actually call REST APIs in your action chain.

To add an If action to an action chain:

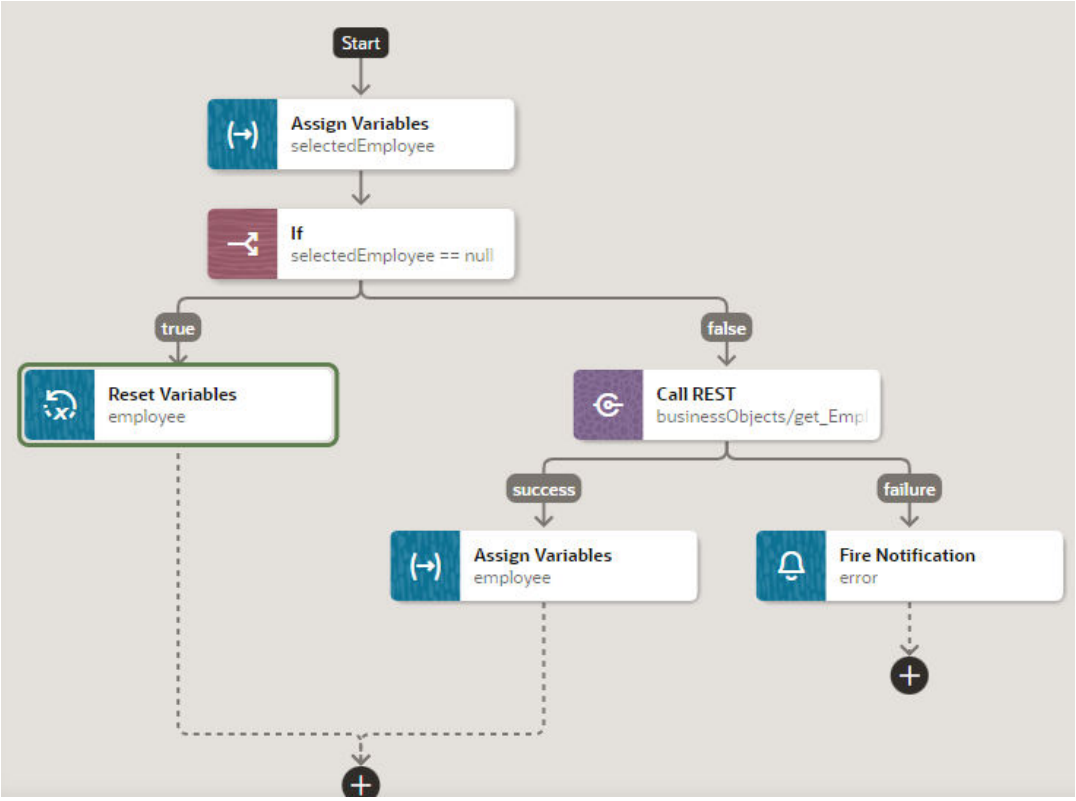
1. Open the Actions editor, for example, at the page level.

2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **If** from the Logic section in 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 Properties pane opens when you add the action to the chain.
4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. In the Condition property, add a condition, for example, `[[ $page.variables.selectedEmployee == null ]]`.



6. Add actions for the **true** and **false** branches to define what should happen when the If action's outcome evaluates to true and false. Here's one possible scenario:



Add a Return Action

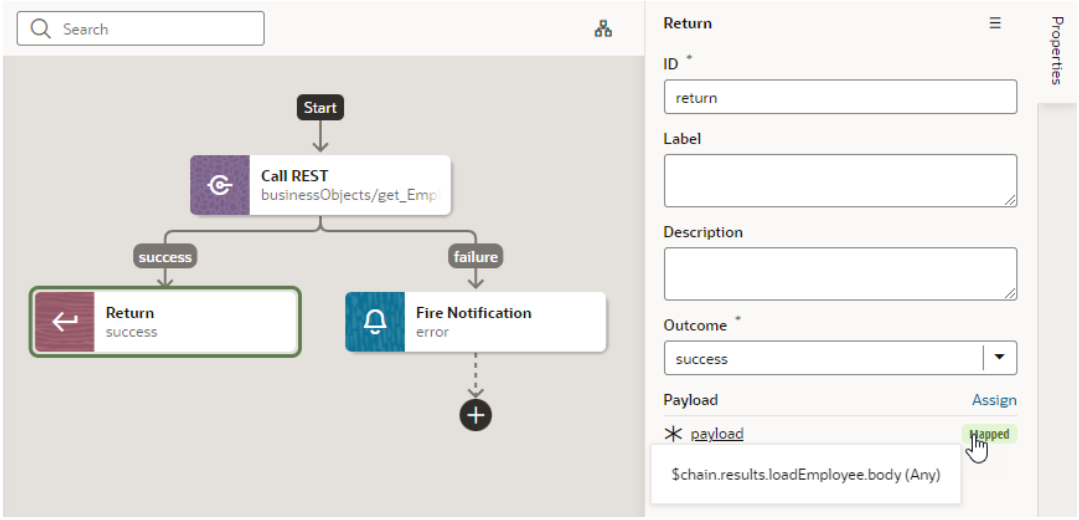
You add a Return action as the final action of a chain to control the outcome and payload of that chain. It's particularly useful in a Call Action Chain action to control the payload resulting from calling that action chain.

To add a Return action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Return** from the Logic section of 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 Properties pane opens when you add the action to the chain.
4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. In the Outcome property, select the outcome to return: **success** or **failure**.
6. Click **Assign** next to Payload to open the Assign Parameters window and map the payload to return from this action.

   Here's an example that uses the Return action on a chain that makes a REST call:



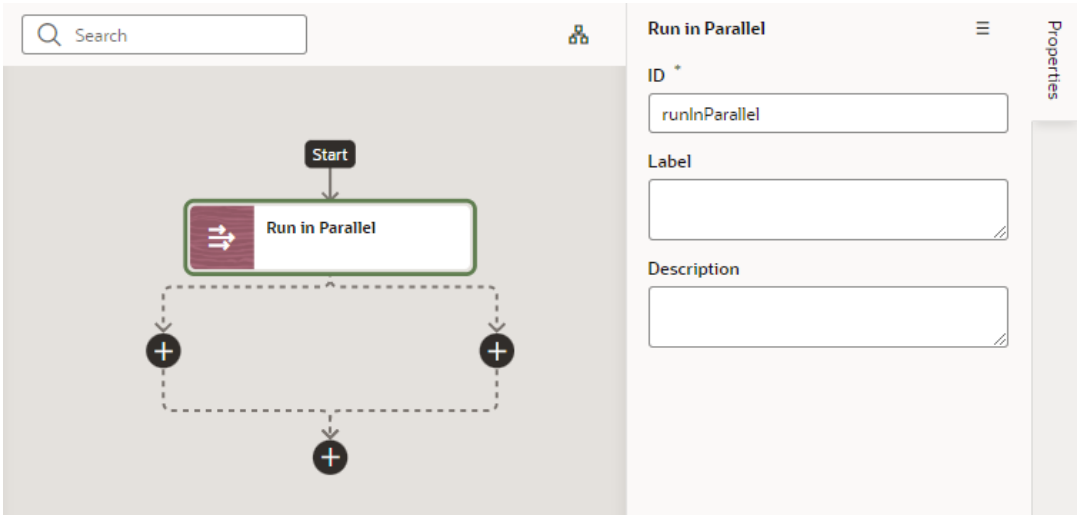
Add a Run In Parallel Action

You use the Run In Parallel action to run multiple action chain paths in parallel, wait for their responses, and produce a combined result.

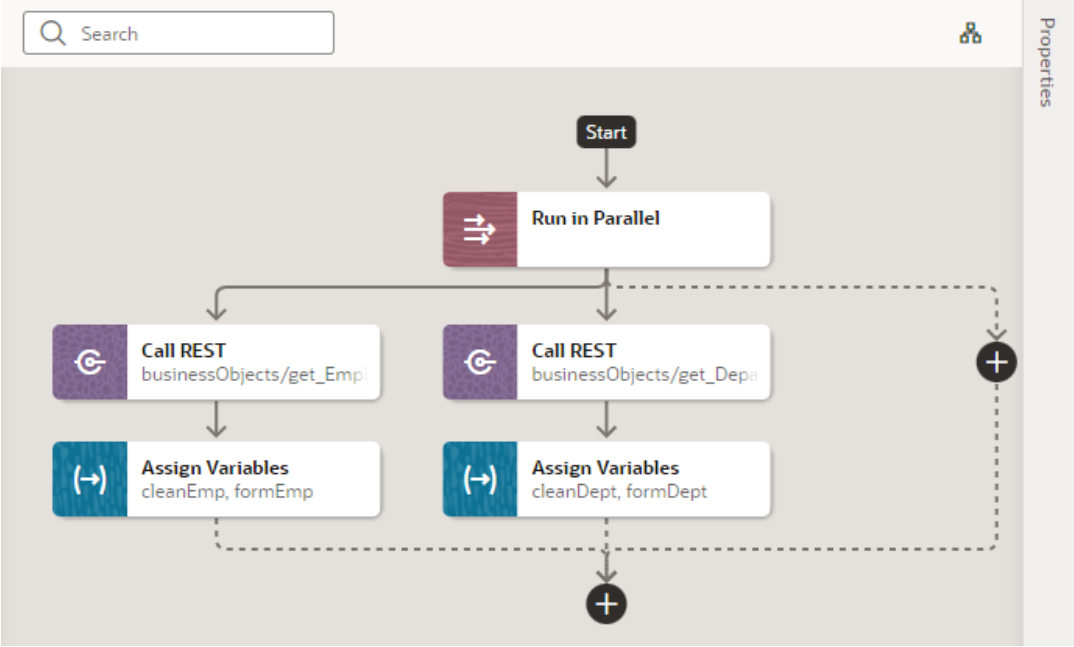
To add a Run In Parallel action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Run in Parallel** from the Logic section of 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.



4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. Click each Add icon ( ) under the Run in Parallel node to define the actions you want to run in parallel, for example, you could make two REST calls, then do some assignments only after they both complete:



Add a Switch Action

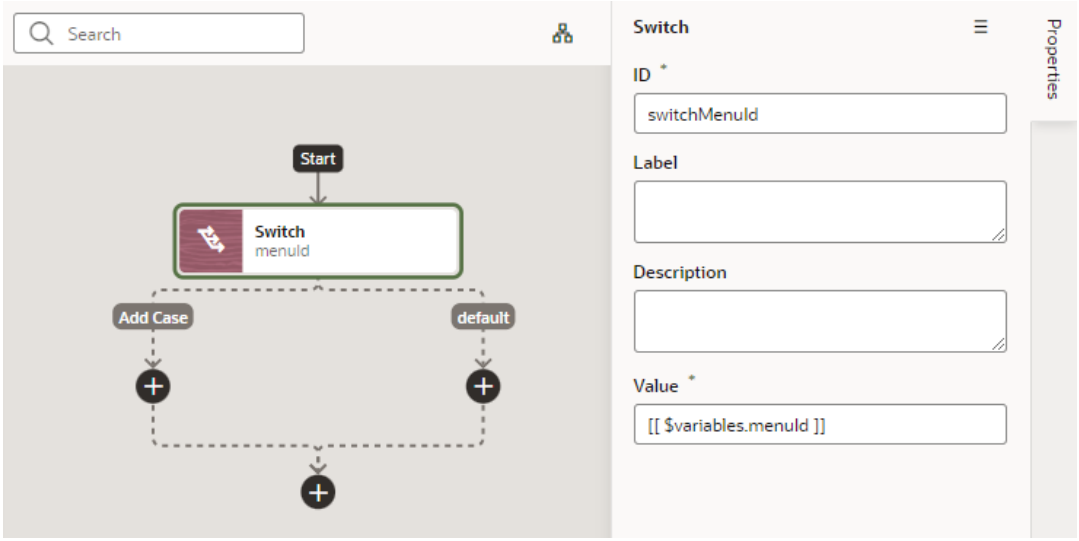
You add a Switch action when you want to evaluate an expression and create an outcome with that value. An outcome of "default" is used when the expression does not evaluate to a usable string.

To add a Switch action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Switch** from the Logic section of 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 Properties pane opens when you add the action to the chain.
4. Update the ID field in the Properties pane to make the action more easily identifiable.
5. In the Value property, enter a value that should be used as the outcome value. If this property is null or undefined, the outcome is "default".

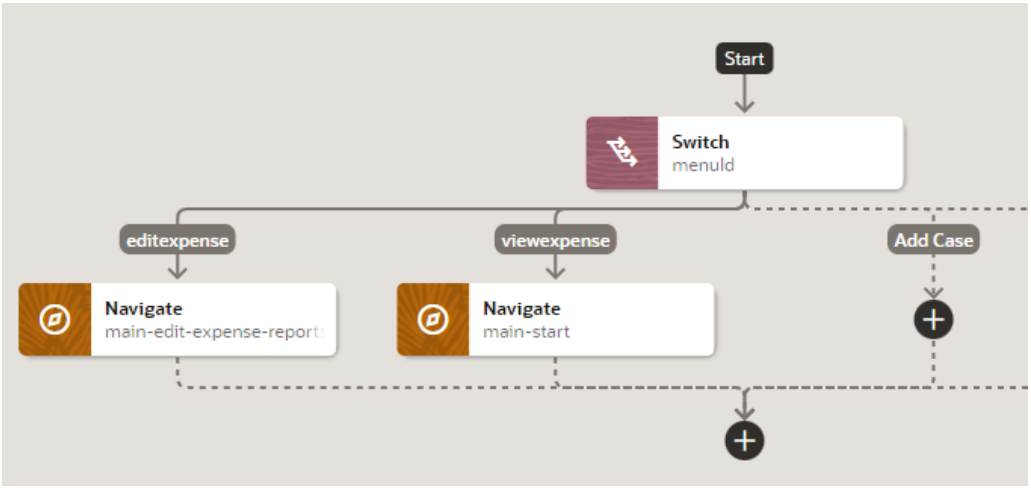
   For example, when a menu defines a set of options, you can use the Switch action to perform actions for each menu item. In this case, you'd pass the `menuId` input parameter containing the selected menu item's ID in the Value property:



6. Optional: You can add further actions for each case. For example, if you want to navigate to a particular page when the menu item is selected, you can add a Navigate action for each menu item.

   a. Click the Add icon ( ) under Add Case.

   b. In the Add Case dialog, enter the case value to be used as the outcome. In our menu example, you would specify the ID of each menu item. Again, if this property is null or undefined, the outcome is "default".



Add a Navigate Action

You add a Navigate action to navigate to a specific page and optionally pass parameters to activate that page.

To add a navigation action to an action chain:

1. Open the Actions editor.
2. Click the action chain in the list to open it in the Action Chain editor.

3. Drag **Navigate** from the Navigation section in the Actions palette and drop it into the action chain.

   You can drag the action onto the Add icon ( ) in the action chain; typically this action will be the final action in the chain. The properties pane opens when you add the action to the action chain.

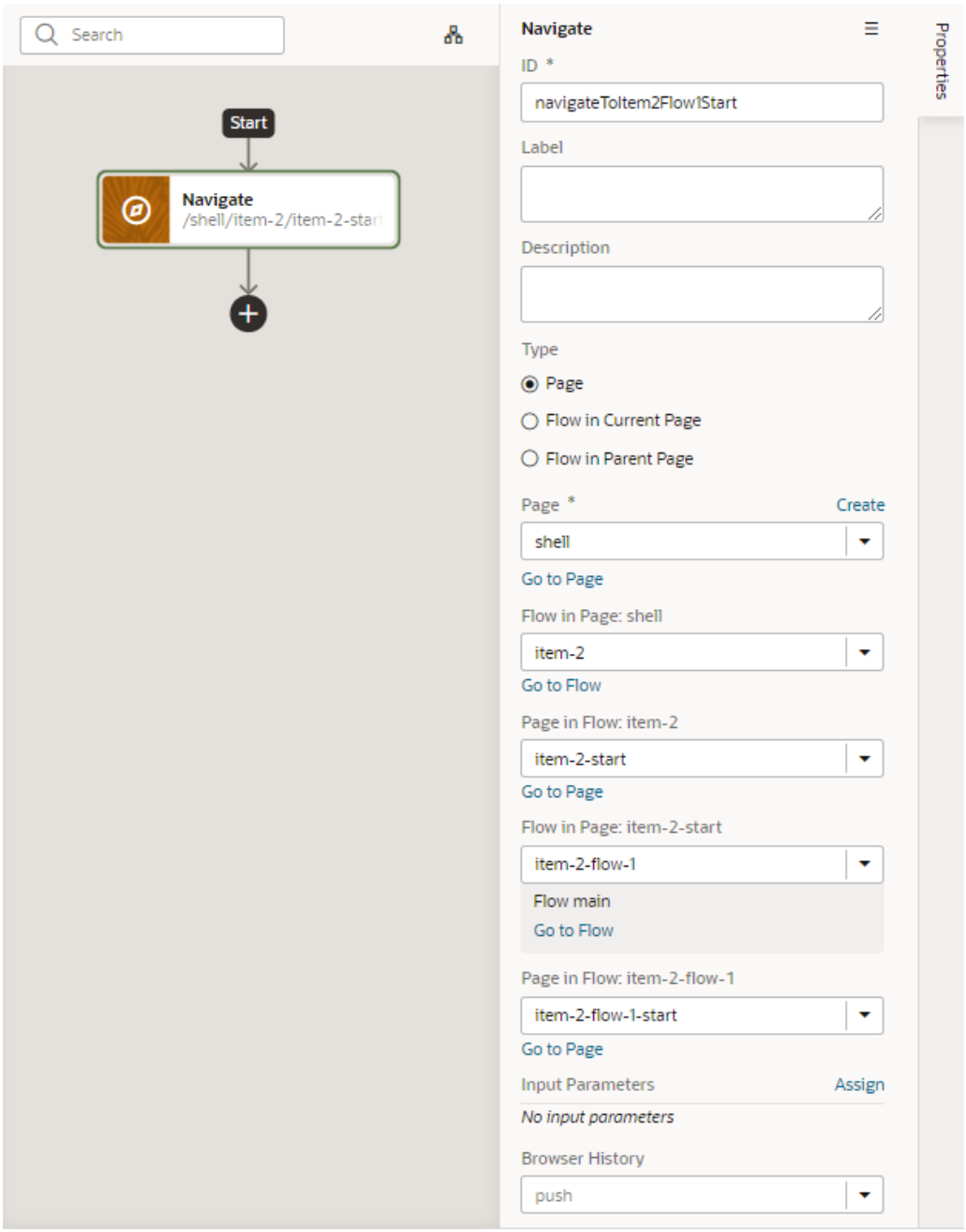
4. Select the type of navigation you want in the Properties pane:

   - **Page**: Enables navigation to a sibling of the current page or a deeply nested page relative to the root of the application or the current page.
   - **Flow in Parent Page**: Enables navigation to a flow of the parent page.
   - **Flow in Current Page**: If the page includes a flow container component, enables navigation to a flow or page within the current page.

   Complete the following steps as it applies to your use case:

   a. Select an existing page from the drop-down list of available pages, or click the **Create** link next to Page to create a new page as the target for the Navigate action.

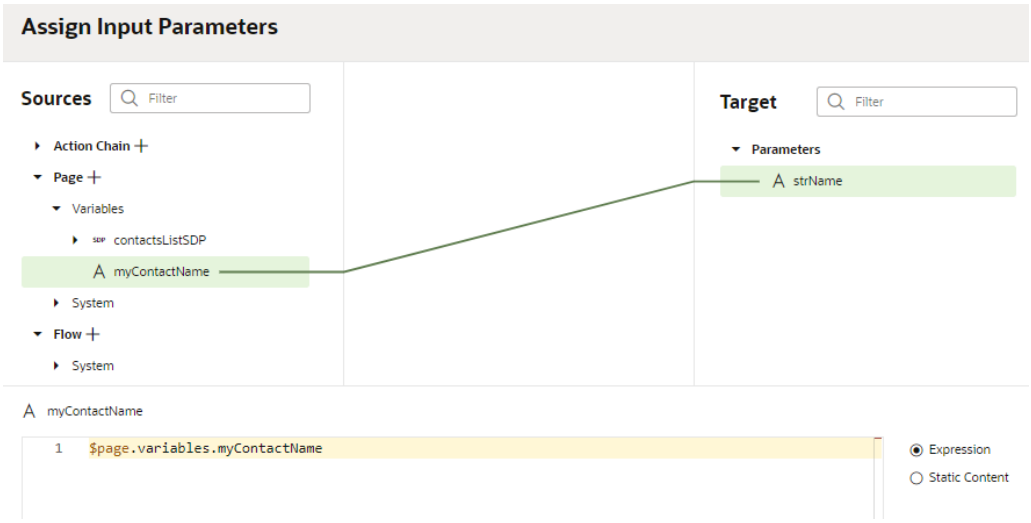
      The pages you can select can be one of the root (shell) pages of the application, another flow or page in the current flow, or a different flow of the parent page. One or more of these options might not be valid targets for your action chain. Here is an example of properties for navigation to a deeply nested page:



b. If the page you select requires input parameters, click the **Assign** link next to Input Parameters to map a page variable to the action's Input Parameter. Click **Save**.

   In the Assign Input Parameters dialog box, you map Sources to Targets by dragging the variable in the Sources pane onto the parameter in the Target pane. If a suitable variable does not exist, use the **+** icon beside the relevant node (Action Chain, Page, and so on) to create a new variable.

   You can click the parameter name to view the expression for the mapped variable.



Add a Navigate Back Action

Add a Navigate Back action to return to the previous page in a browser's history.

To add a Run In Parallel action to an action chain:

1. Open the Actions editor (for example, at the page level).
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Navigate Back** from the Navigation section of the Actions palette into the action chain.

   You can drag the action onto the Add icon ( ) in the action chain; typically this action will be the final action in the chain. The Properties pane opens when you add the action to the chain.
4. Optional: Specify a key/value pair map of parameters to pass to the previous page. If a parameter is not specified, the original value of the input parameter on the destination page is used. If a parameter is specified, it has precedence over fromUrl parameters.

Add an Open URL Action

You add an Open URL action to navigate to an external URL. In a web app, this action opens the specified URL in the current window or in a new window.

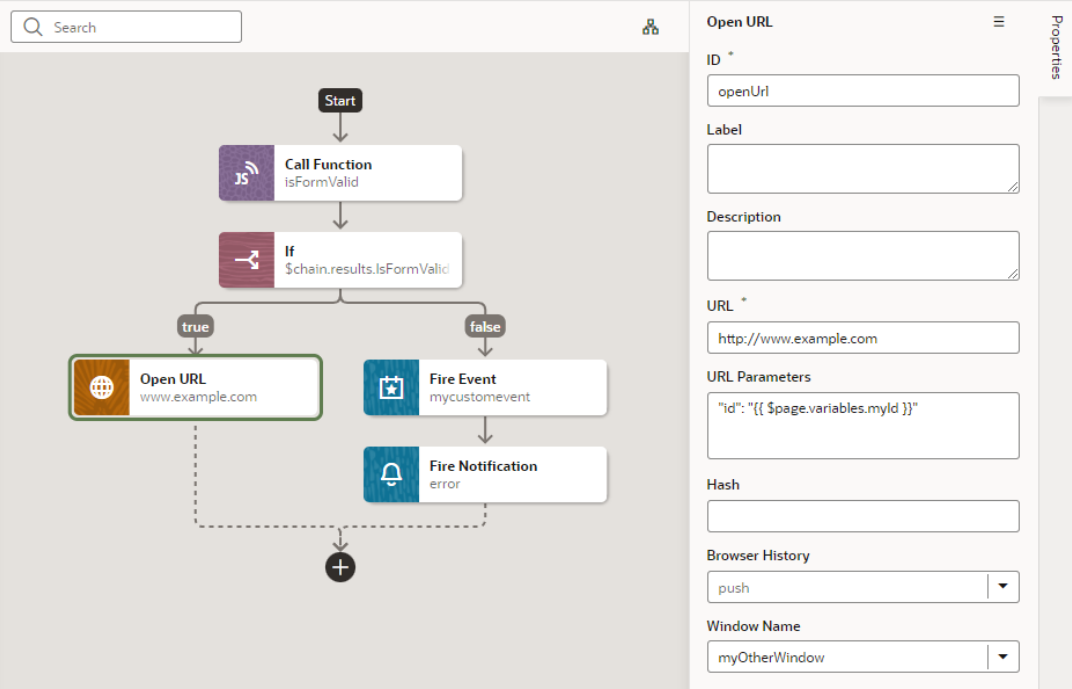
To add an Open URL action to an action chain:

1. Open the Actions editor, for example, at the page level.
2. Create an action chain, or open an existing action chain to add the action in the editor.
3. Drag **Open URL** from the Navigation section of the Actions palette into the action chain.

   You can drag the action onto the Add icon ( ) in the action chain; typically this action will be the final action in the chain. The Properties pane opens when you add the action to the chain.
4. Enter the URL to navigate to.

5. Optional: Specify a key/value pair map of query parameters to pass to the specified URL as URL parameters.
6. Optional: Specify the hash entry to append to the URL.
7. Define a Browser History value, either **replace** or **push** (default), to define the effect on browser history. This value is used only if the resource is used in the same window. If you choose **replace**, the current browser history entry is replaced instead of pushed, meaning that the back button will not go back to it.
8. Specify a name identifying the window as defined in the `window.open()` API. If not defined, the URL opens in the current window. For apps on mobile devices, you have three possible values: `_self` (default), `_blank`, or `_system`. For local file types, this property is ignored.

   Here's an example to open a new window in the browser with the given URL; if you specify a value for the Window Name (as shown here), once on the URL, the browser back button will re-enter the last page and the page input parameters will be remembered.



Custom Actions

In addition to the built-in actions you see in the Actions palette, you can create your own actions and use them in action chains just the way you'd use built-in actions.

Custom actions are created within the context of a specific application, and can't be shared across apps.

Create a Custom Action

To create a custom action, you provide its metadata in a JSON file and its code in a JavaScript file. The metadata contains basic details about the action, any input parameters needed by its implementation method, and optionally, an object for returning values.

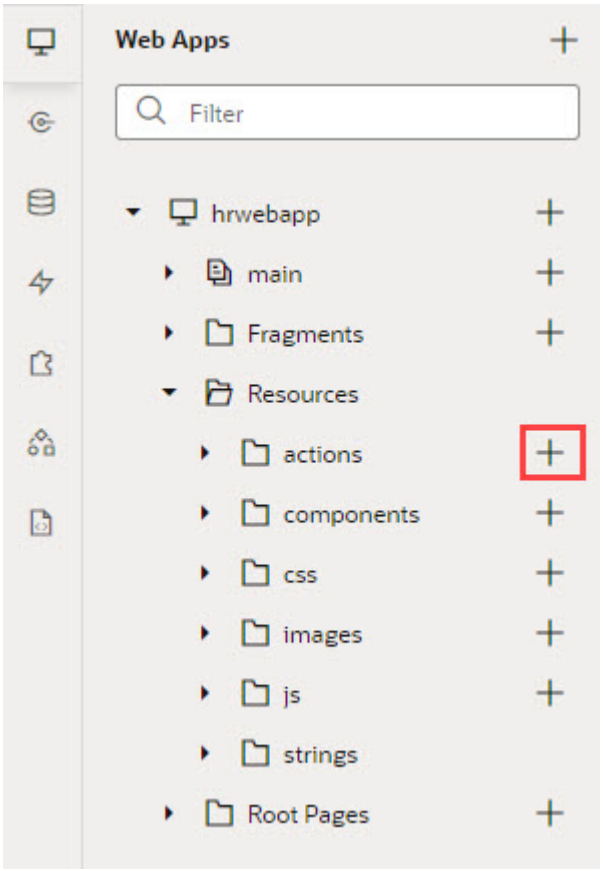
Here's an overview of what's required to create a custom action:

1. Create the Action Files:
   - **action.json**: Contains the metadata for the custom action. Used to define input parameters and to define an object for returning values. This file is also used by the Designer to add the action to the Actions palette, and to display the action's properties in the Properties pane.
   - **action.js**: Contains the code used to implement the custom action.
2. Add the Metadata to `action.json`.
3. Add the code for the custom action to `action.js`.

Create the Action Files

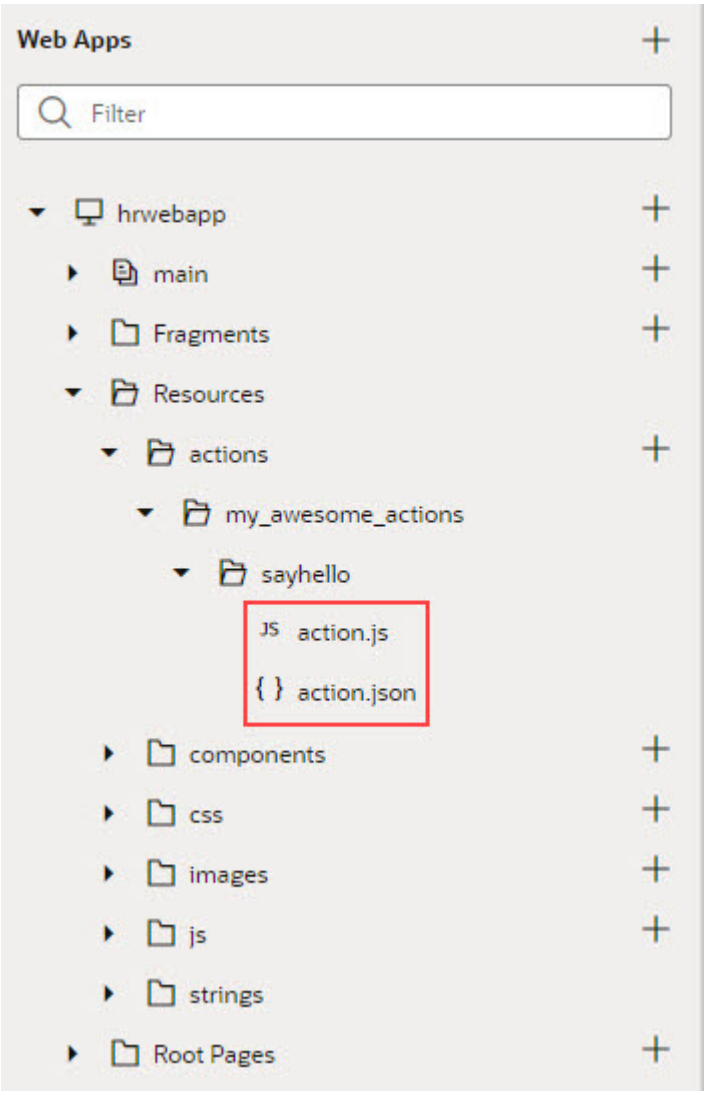
To create the template `action.js` and `action.json` files, for you to start with:

1. Select the **Web Apps** tab, expand the Application node (first node) and the **Resources** node under it, then click the Create Custom Action icon (+) next to the **actions** node:



2. In the **ID** field, enter the name of the action group folder for your new custom action, followed by a forward slash and the name of your new action, as in: `<action-group>/<action-name>`
   For example, you might enter `my_awesome_actions/sayhello`, where `my_awesome_actions` is the name of your group folder, and `sayhello` is the name of your action.

   The two newly created `action.js` and `action.json` templates are stored under the path `resources/actions/<action-group>/<action-name>/`, as shown here:

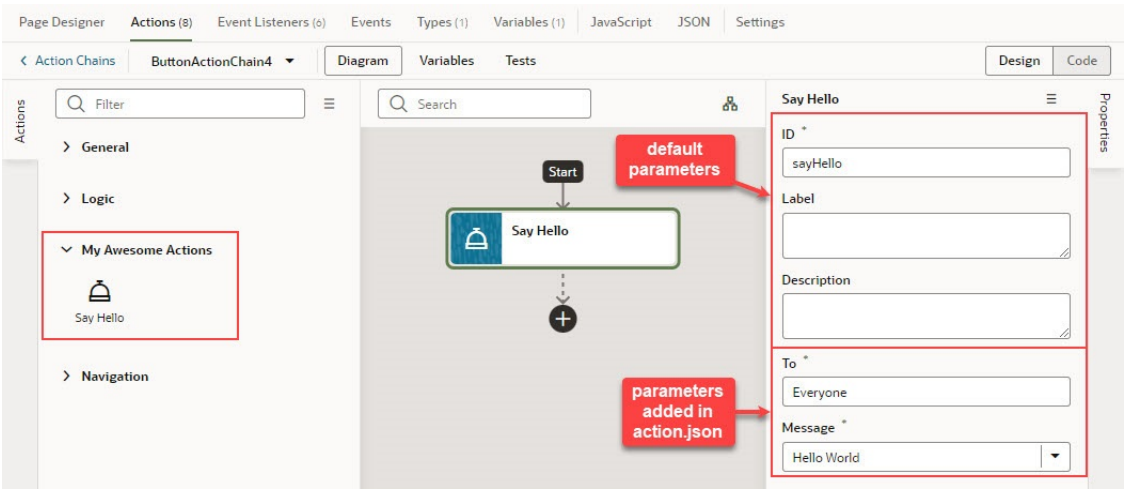


Add the Metadata

You provide the custom action's metadata in the `action.json` file, which includes:

- The action's basic details (ID, display name, icon…)
- An object used to return values from the action's implementation method *(optional)*
- Input parameters needed by the action's implementation method *(optional)*

The Action Chains editor uses the metadata to add the custom action to the Actions palette and to display its parameters in the Properties pane:



When you created the `action.json` file, the default parameters, ID and Description were automatically created for you, but their metadata isn't added to `action.json`. You only add the metadata for the input parameters and the return object that you want to add to the custom action.

Here's an example of the `action.json` file for the sample `sayHello` custom action, with a breakdown of its parts:

- The first set of properties provide the basic details about the custom action.

- The `resultShape` property provides the definition for the object returned by the action's implementation method, which can be used as input for another action when creating an action chain. For instance, the string returned by this action can be an input for an action that writes the string to a log file.
- The `propertyInspector` property defines the action's input parameters.

Define the Custom Action's Properties

Use this table to help you define the properties for your custom action in the `action.json` file.

Property	Required	Description
"id": "",	Yes	Unique ID for custom action.
"category": "My Category",	No	Category to contain custom action in Actions palette. If not specified, action is placed under the default category for custom actions, *Custom*.
"defaultParameters": { },	No	If input parameters are defined and they need default values, use this property to specify default values for them by specifying the input parameter names and their values (name-value pairs). Defaults are assigned when action is first added to an action chain.
"description": "",	No	Brief description of custom action.
"displayName": "",	Yes	Name to display in Actions palette.
"helpDescription": "",	No	Help text to appear for action when user hovers over action's title in Properties pane and clicks the question mark icon.
"iconClass": "",	Yes	Icon to display for action in Actions palette.
"referenceable": "self | extension"	No	Indicates if action is available in extensions; default is `self`, indicating it isn't available in extensions.
"idPrefix": "",	No	Used to auto-generate action IDs for actions when they are added to action chains. When action is added to an action chain, action's ID field in the Properties pane is auto-populated. If specified, ID field is populated using this property's value, otherwise, the action's name is used.
"resultShape": { "someName": "string" },	No	If one or more values are to be returned by the implementation method, use this property to define the object to return them.
"showInDiagram": "on" | "off"	No	If set to `on`, action is available on Actions palette of flow diagram.

Property	Required	Description
"tests": { "requiresMock": "on" | "off" }	No	Setting for action's mock requirements for action chain tests: **on:** Indicates action needs to be mocked. **off:** Indicates Visual Builder provides suggestions for expected action results if `resultShape` parameter is specified in `action.json`. If action has no `resultShape`, suggestions are enabled for the action's input parameters specified in propertyInspector section. Default is `off`.
"propertyInspector": [{}]	No	Metadata for the input parameters needed by implementation method. Input parameters are displayed in Properties pane of Actions editor.

Define Input Parameters for a Custom Action

Use this table to help you define input parameters for your custom action. In the `action.json` file, use the `propertyInspector` property to define the parameters you need:.

Property	Required	Description
"name": "",	Yes	Name of input parameter.
"help": "",	No	Help text to appear for input parameter when user hovers over parameter's title in Properties pane and clicks the question mark icon.
"label": "",	Yes	Label to display for input parameter in Properties pane.
"placeholder": "",	No	Hint text to display for input parameter in Properties pane.
"required": true | false,	No	Indicates if a value is required for input parameter.
"options": [{}],	No	If `component` property for input parameter is set to `comboboxOne` or `comboboxMany`, use this property to specify all of the values to be available for the combobox.
"type": "",	Yes	Parameter's data type, which can be number, string, boolean or object; if the type is not specified, values are stored as strings.
"component: "inputText | textArea | comboboxOne | comboboxMany"	Yes	Indicates if parameter is a text field, text box or a combobox with single or multiple selections.

Add the Code

You provide the code for your custom action using the `action.js` template file that was created for you when you first created the new action.

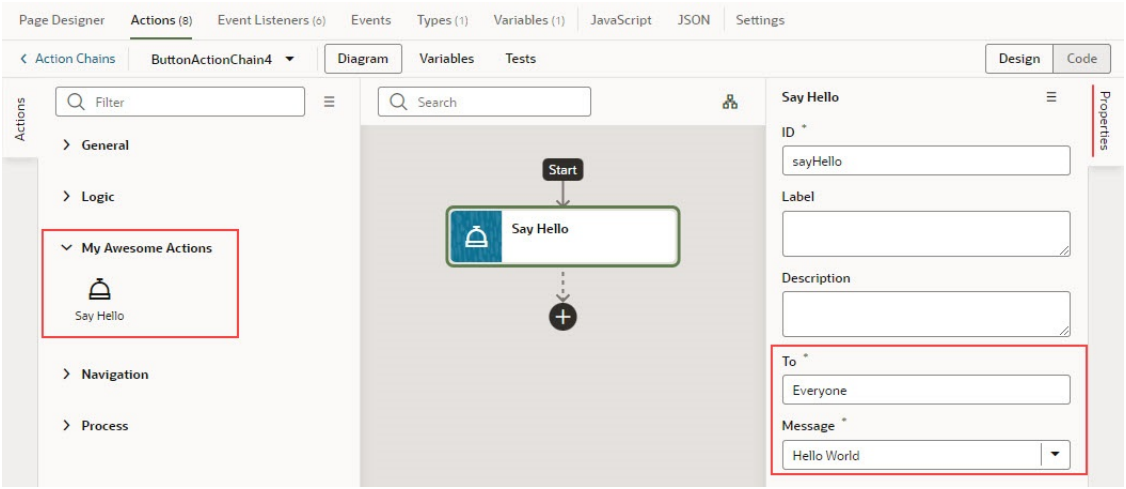
To provide the code for your action, use the `perform()` method, which receives the input parameter, `parameters`. The input parameter contains the values for the action’s input parameters, as entered in the Properties pane.

How Are Input Parameters Passed?

Here's an example of the `perform()` method, in `action.js`, that implements the sample `sayHello` custom action by using the `alert()` method:

```
define(['vb/action/action'], (Action) => {
            'use strict';
            class CustomAction extends Action {
               perform(parameters) {
                  const to = parameters.to;
                  const message = parameters.message;

                  alert(`${message}, ${to}`);

                  if (to && message ) {
                     return Action.createSuccessOutcome({result: to + ", " +
message});
                  }
                  return Action.createFailureOutcome({result: "Something
wrong, unable to SayHello"});
               }
            }
            return CustomAction;
         });
```

The values you enter for the action's input parameters, To and Message, in the Properties pane, are passed to the `perform()` method using the input parameter, `parameters`.



How are Values Returned?

If you need one or more values returned by the method that implements the custom action, you need to define the object that returns them in `action.json`. You do so by using the `resultShape` property.

To return values from the implementation method, you use the `createSuccessOutcome()` method of the `Action` class.

As an example, here's the declaration of the return object for the sample `sayHello` custom action:

```
"resultShape": { "result": "string" },
```

And here's the code line in `action.js` using the `createSuccessOutcome()` method to return a string:

```
 return Action.createSuccessOutcome({result: to + ", " + message});
```

Specify Path to Code

Lastly, you need to tell Visual Builder where the custom action's code file is stored, by adding a `requirejs` property to the `app-flow.json` file. Here are the steps:

1. In the Navigator, on the **Web Apps** tab, select the web app, then select the **JSON** tab to open the `app-flow.json` file.
2. Specify the path to the custom action's code file by adding a `requirejs` property in `app-flow.json` and naming the path, using this format:

```
"<custom-action-ID>": "resources/actions/<action_group>/<action_name>"
```

In this example, `<custom-action-ID>` is the ID for the custom action (case sensitive), as specified in `action.json`.

Here's an example of specifying the path:

```
"requirejs": {
   "paths": {
      "demo/SayHello": "resources/actions/my_awesome_actions/sayhello/action"
    }
},
```

Test Action Chains

Visual Builder can help you test the flow of your action chains by generating suggestions of outcomes to validate in a Tests editor. You can use this editor to apply a test-driven approach to developing action chains.

> **Note:**
>
> All new action chains are now created using JavaScript, which offers significant advantages over JSON. Existing JSON chains can be edited, but no new JSON chains or tests for JSON chains can be created. For a list of benefits to using JavaScript, see the end of Work With JSON Action Chains.

The Tests editor detects what information needs to be provided at runtime to perform actions in an action chain. This information includes values for variables and constants used by the action chain, and actions like Call REST endpoint, the results of which need to be mocked when running a test. Once the necessary context is provided, the editor generates suggestions

for expected outcomes (expectations) that you can add to the test. You can also add your own expectations.

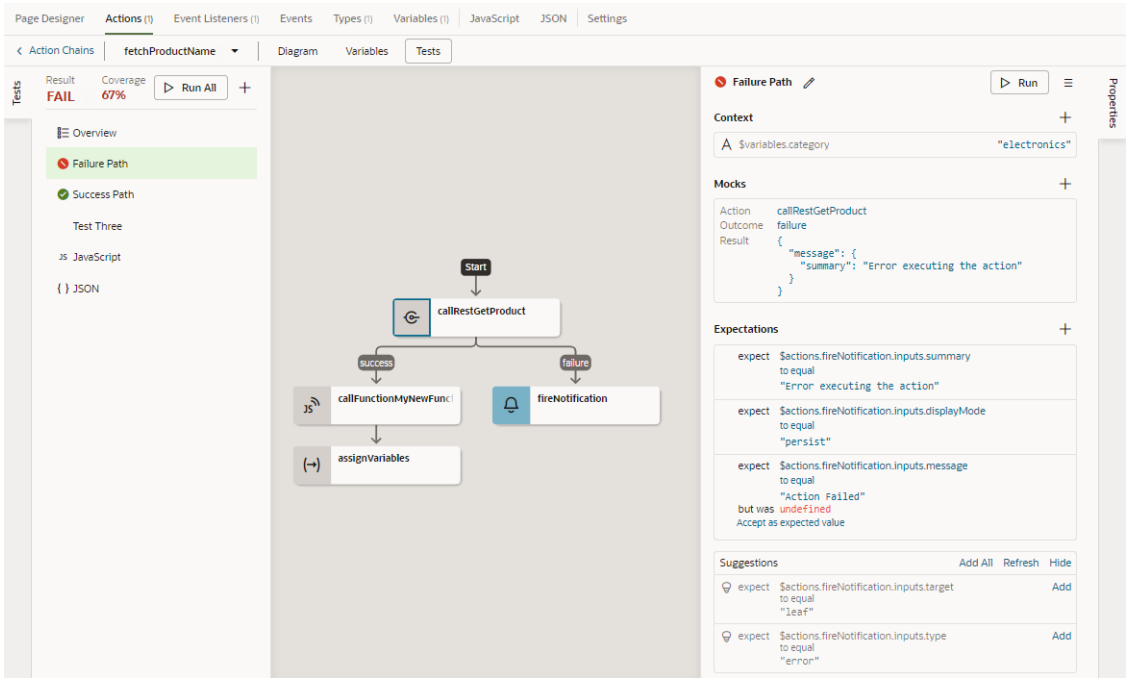
Access this editor from the **Tests** tab in the Actions editor of an individual action chain. The first time you access the Tests editor, click the **+ Test** button to create a test. The test name defaults to `Test 1`; specify an alternative name, if you want.

Once in the editor, you can create one or more tests for the associated action chain. For each test, you define *context*, *mock actions*, and *expectations*.

- A context refers to a variable that the action chain uses. If, for example, you have an action chain that uses a Call Function that takes a `subtotal` variable as input and returns the total after tax, you add a context entry that includes the `subtotal` variable and a sample value for the variable. To add a context, click + in the Properties pane next to Context.
- Mock actions are needed for Call REST endpoint actions and other actions in the action chain. For each mock action, you specify a possible outcome for the action and a result. If, for example, your action chain includes a Call REST endpoint action that fetches product information, you need to specify a mock action that has a success outcome and includes a sample result of product information. To add a mock action, click + in the Properties pane next to Mock. Or, right-click the action for which you need to define a mock action and select **Mock Action** from the context menu.
- Finally, you specify expectations for the test. Visual Builder generates a set of suggested expectations that you can add to the test. You can add one or more of these to the test and edit the expected outcome. To add an expectation, click + in the Properties pane next to Expectations, or click **Get Suggestions**. You can also right-click the action for which you need to define an expectation and select **Add Expectation** from the context menu.

Once you've defined the tests, you can run them individually or all at once using the **Run** or **Run All** button. You can also run these tests using the `vb-test` Grunt task (see Test Action Chains Using the `vb-test` Grunt Task).

The following image shows three tests defined for an action chain that fetches product information:



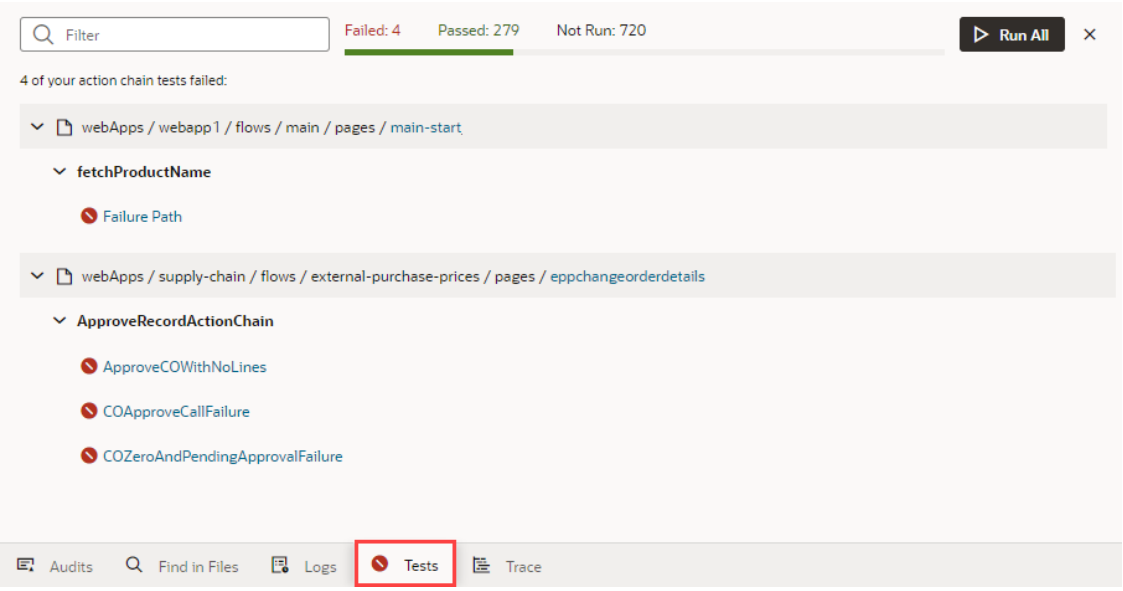
Two of these tests have run (Failure Path and Success Path), one has not (Test Three). The red icon beside the Failure Path test indicates that it failed; the reason for the failure is also marked red in the expectation for the fireNotification message, where the test author set the expected outcome to `Action Failed` but the actual outcome was `undefined`. The green icon beside the Success Path test indicates that the test succeeded. No visual indicator appears beside Test Three because it has not yet been executed.

The percent value for Coverage indicates the level of test coverage for the actions in the action chain. If you create tests that include all actions in the action chain and all expected outcomes for the actions, the percent value for Coverage will be 100%. In the preceding image, the test author has removed some entries from the Expectations list for the Failure Path test, thereby reducing the action chain's Coverage value. You can increase the Coverage value by adding entries that appear under the Suggestions list to the Expectations list.

The source code for your tests is stored in a separate JSON file for easier maintenance, one *`actionchainname`*`-tests.json` file for all the tests in an action chain. To view this file's contents, click **JSON** in the left pane. You can also find this file under the artifact's `chains` folder in the Navigator's Source View tab.

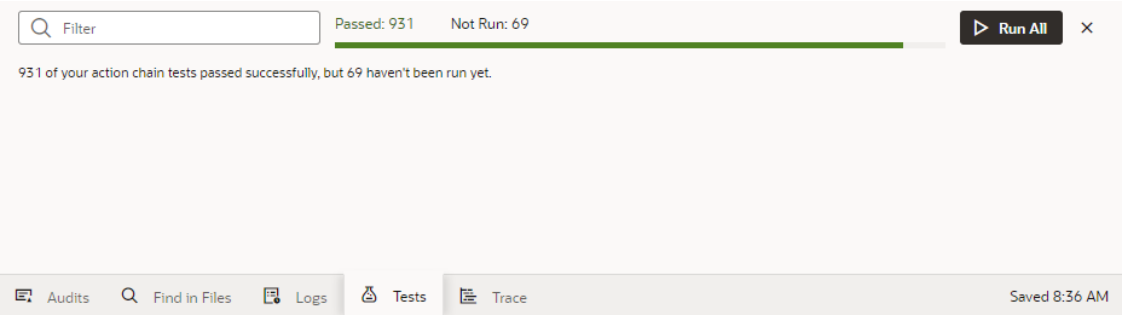
Manage All Tests in a Visual Application

When you define multiple tests for each action chain in an app, it might be easier to manage tests for all apps in a visual application, rather than for each app. You can do this using the **Tests** tab at the bottom of your browser.



This aggregate view helps you get a quick look at the status of all action chain tests in a visual application. When tests fail, you can use this view to quickly access the editor for each failed test and take action as needed.

While you can also run all tests in your application from here, it isn't really required if you've already triggered them. Action chain tests run in the background, even when you're not actively working on your application, and the results are saved. So if you are working on an application, only tests impacted by your changes (for example, if you added a new variable or updated an existing function) are scheduled to run again. You'll likely see something similar to this image until Visual Builder actually runs those tests for you (after 10 seconds of inactivity):



This way, your test results are always available and up-to-date, and you can rely on them to identify and fix breaking-code changes.

Test Action Chains Using the `vb-test` Grunt Task

Visual Builder's `grunt-vb-build` NPM package includes a `vb-test` Grunt task that you can use to run the action chain tests in your visual application on your computer.

To use the `vb-test` Grunt task, you must set up your computer to build the visual application by installing Node.js and its package manage (npm). See Build Your Application Locally. Once you have installed the necessary tools, you need to save the sources for the visual application to your computer. You can get the sources of your visual application in one of the following ways:

- Cloning the Git repository containing the sources
- Exporting the visual application from Visual Builder and extracting it to your local system

To test action chains using the `vb-test` Grunt task:

1. In the command-line interface, navigate to the folder on your local system containing the `package.json` and `Gruntfile.js` files.
2. Enter `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`.
3. Enter the task names in the command-line interface to process your application sources, and then run the suite of tests that you defined in Visual Builder.
   The following example shows how you execute these tasks, along with some of the parameters that they support:

   ```
   # First build application sources. This creates a build/processed
   directory with the built application assets.
   ./node_modules/.bin/grunt vb-process-local

   # Run the suite of action chain tests that you defined using one of the
   following options:
   ## Headless mode:
   grunt vb-test
   ## Test in the Chrome browser and set a timeout value:
   grunt vb-test --karma-browser=Chrome --mocha-timeout=60000
   ```

   The command-line options include the following:

   - `karma-browser`

By default the tests run in headless Chrome, but you can pass Chrome to use the UI (window) mode instead.

Example to run tests in Chrome UI:

```
grunt vb-clean vb-process-local vb-test--karma-browser=Chrome
```

- `karma-debug`

  Runs tests in Chrome UI mode. Suspends execution until you click the **DEBUG** button, at which point you can debug the app tests using Chrome DevTools. The Default value is `false`.

  Example to run tests in Chrome UI and debug mode:

  ```
  grunt vb-clean vb-process-local vb-test --karma-browser=Chrome --karma-debug
  ```

- `karma-log-level`
  Sets the karma logging level. The default level is `INFO`, though you can change this to `DEBUG`, `WARN`, `ERROR`, or `DISABLE`.

  Example to run tests in Chrome UI with increased verbosity:

  ```
  grunt vb-clean vb-process-local vb-test --karma-browser=Chrome --karma-log-level=DEBUG
  ```

- `mocha-timeout`
  Sets the timeout for Mocha tests (in milliseconds).

  Example to run tests in Chrome UI with a timeout for Mocha tests:

  ```
  grunt vb-test --karma-browser=Chrome --mocha-timeout=60000
  ```

For more information on the supported command-line options, see vb-test.

4. Check test results and code coverage reports in the `build/tests/results` directory.

Start an Action Chain

You set up an action chain to be triggered when an event occurs in an artifact. The type of event available depends on the artifact. For example, you can trigger an action chain to start when a lifecycle event such as `vbEnter` is fired to load a page. Or, use the `onValueChanged` variable event when a variable's value changes. You can also use custom events to start an action chain from another action chain.

Start an Action Chain From a Component

When you add a component to a page or layout, you'll need to create a component event and component event listener if you want it to trigger some behavior (for example, to open a URL). The suggested option in the component's Properties pane creates these for you.

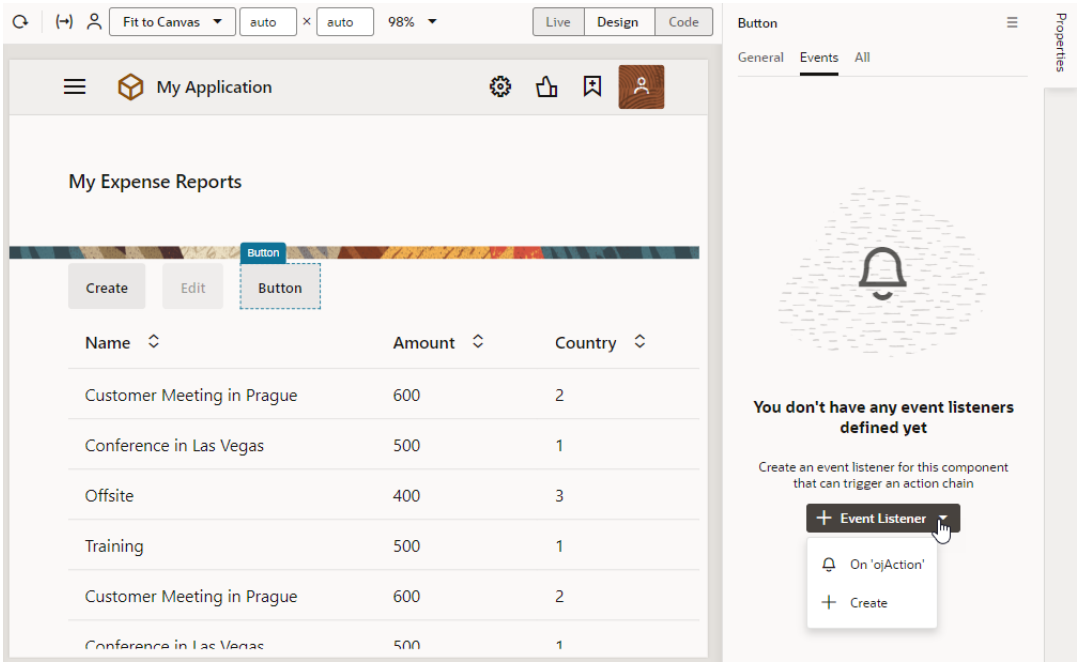
There are various predefined events that you can apply to a component, and the events available are usually determined by the component. For example, the `ojAction` event is triggered when a button is clicked, so you would typically apply it to a button component (you couldn't apply it to a text field component). Each button will have a unique event and an event listener listening for the button's `ojAction` event, and the listener would start an action chain (or multiple action chains) when the event occurs. Each component event will usually have a corresponding component event listener.

> **Note:**
>
> You can add an event to a component only from the component's Properties pane. You can't create one in the Events tab of pages.

To start an action with a component:

1. Select the component in a page or layout.

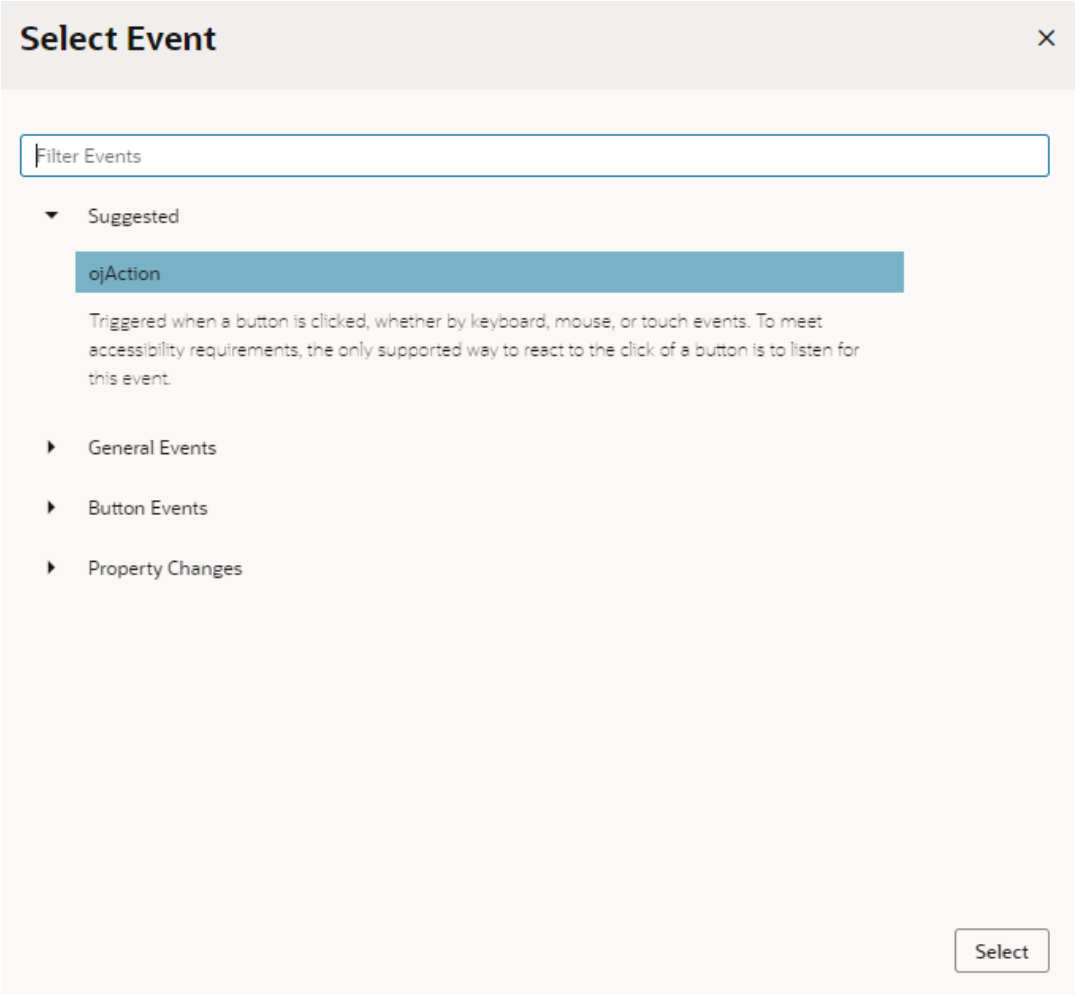
   Typically, you assign events to elements such as buttons, menus, and fields in form components. You can select the component on the canvas, in the Structure view, or in Code view.



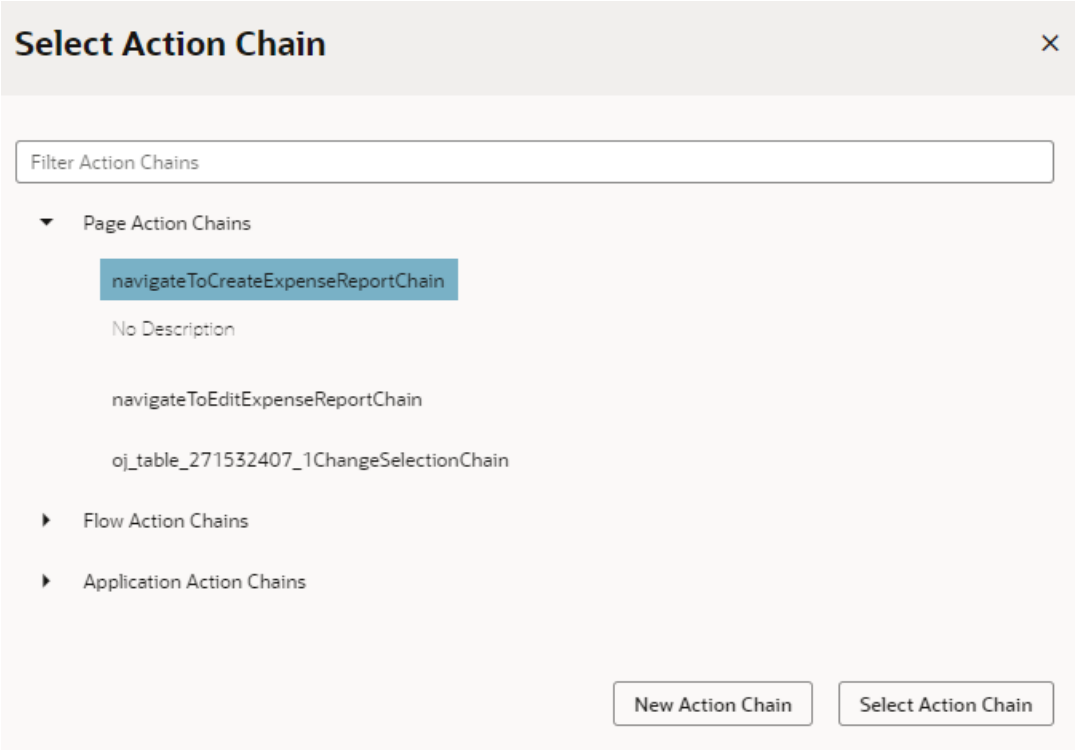
2. In the component's Events tab in the Properties pane, click **+ Event Listener**. You can choose the suggested event as a quick start or you can create a custom event to use a different event.

   When you add the new event using the quick start, an action chain is created for you and the Action Chain editor opens automatically. When you add the new event using the custom option, you'll need to select an event.

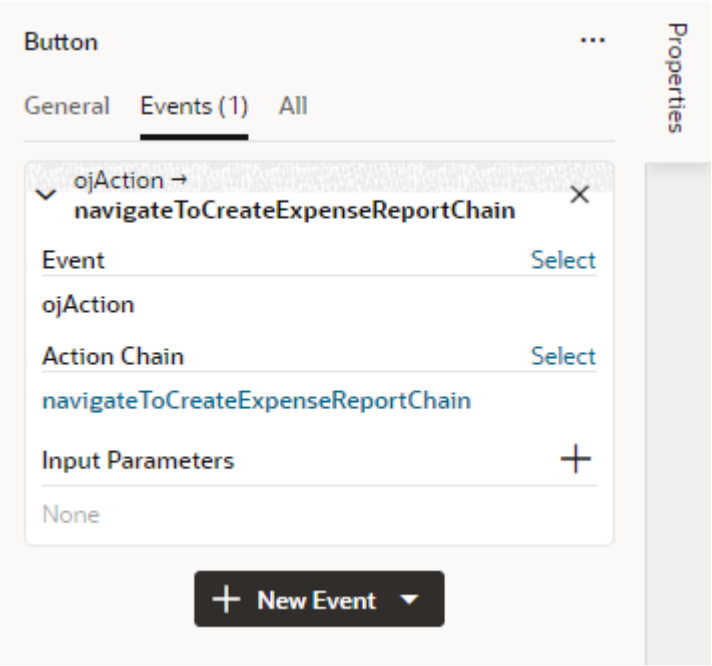
3. For a custom event, select the event you want to use to trigger an action chain. Click **Select**.



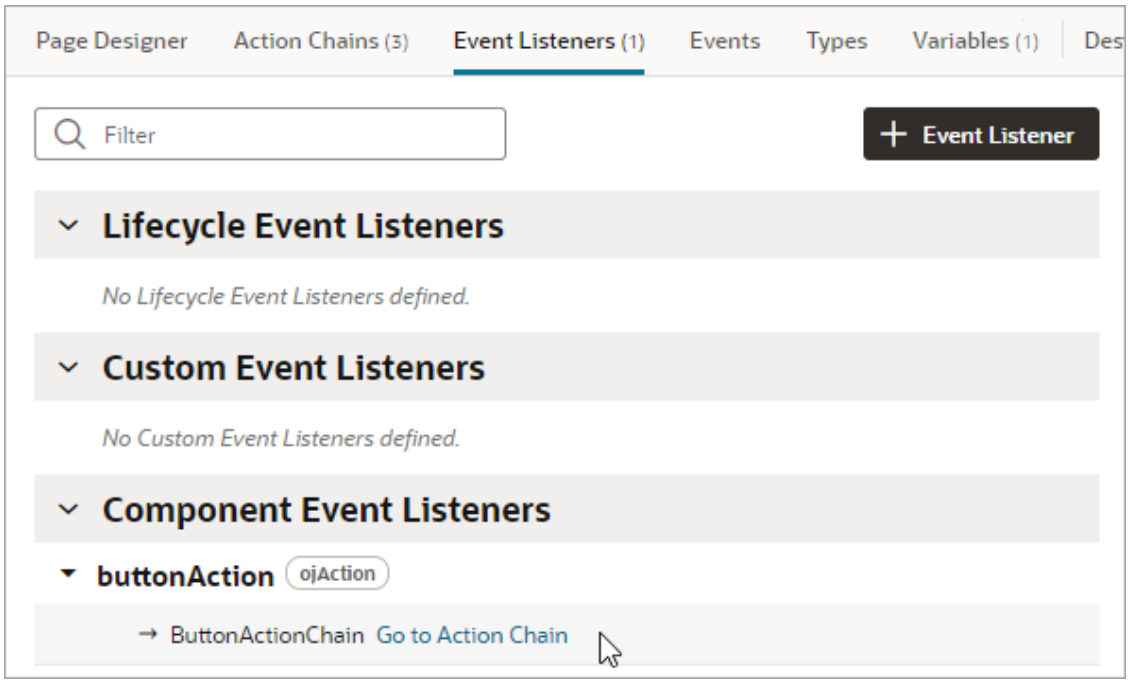
4. Select the action chain you want the event to trigger and click **Select Action Chain**. Alternatively, click **New Action Chain** to create a new action chain.



The Events tab in the Properties pane shows events on the component that Visual Builder responds to by triggering action chains. You can edit the properties, for example, to add input parameters that you want to use in the action chain. Input parameters can provide values from the component and its page to the action chain, which the action chain can then use to determine its behavior. For example, a table selection event could supply details of which row was selected to its action chain.



If you used the quick start option to add an event, a component event listener is created for the new event, and the listener is mapped to the action chain it created for you. If you open the Event Listeners tab, you'll see it listed under Component Event Listeners, along with the action chain that it will trigger.



Start an Action Chain When a Variable Changes

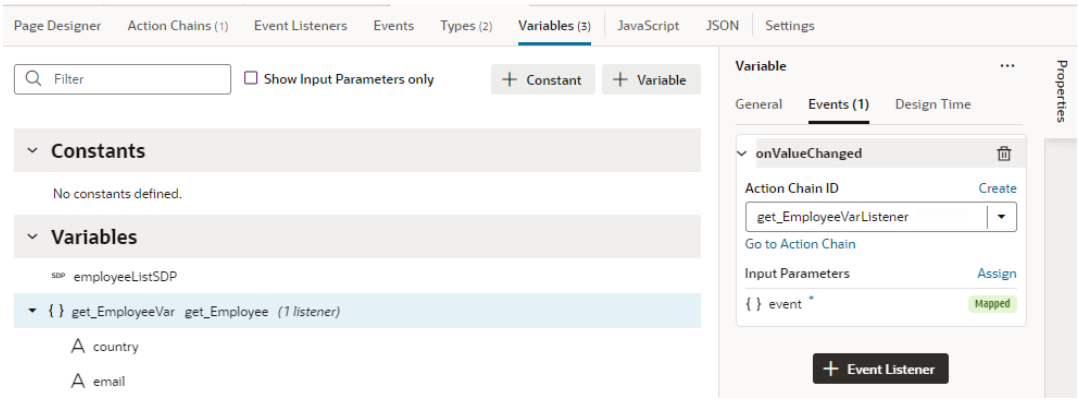
You can start an action chain when the value stored in a variable changes by adding an `onValueChanged` event to the variable.

When you use an `onValueChanged` event to trigger an action chain, the trigger has the payload of the variable's old and new values. For example, let's say you changed the name property of an Employee and then reset the Employee; the framework will send an event that the Employee changed, and as part of the payload indicate that the name has changed.

To start an action chain when the value of a variable changes:

1. Open the **Variables** tab of an artifact.
2. Select the variable in the list, then open the **Events** tab in the Properties pane.
3. Click **+ Event Listener** in the Events tab.
4. Select an action chain from the list. Click **Select**.

   When you add the event to the variable, a variable event listener that listens for the `onValueChanged` event on the variable is automatically created. The variable's Events tab in the Properties pane displays the action chain the event listener will trigger; you can change or remove the action chain, assign input parameters, and add more action chains.



> **Note:**
>
> Variable events and event listeners are not listed in an artifact's Events or Event Listeners tabs.

Start an Action Chain From a Lifecycle Event

Lifecycle events are predefined events that occur during a page's lifecycle. You can start action chains when these events occur by creating event listeners for them. For example, if you want to initialize some component variables when the page opens, you can create an event listener in your artifact that listens for the `vbEnter` event. You could then set the event listener to trigger an action chain that assigns values to the component's variables.

Before you create an event listener to trigger an action chain, it's important to understand a page's lifecycle, so you know where to plug in custom code to augment the page's lifecycle. Each page in your application has a defined lifecycle, which is simply a series of processing steps. These might involve initializing the page, initializing variables and types, rendering components, and so on.

Each stage of the lifecycle has events associated with it. You can "listen" for these events and start action chains whenever they occur to perform something based on your requirements. For example, to load data before a page loads, you can use the `vbEnter` event and start an action chain that calls a GET REST endpoint.

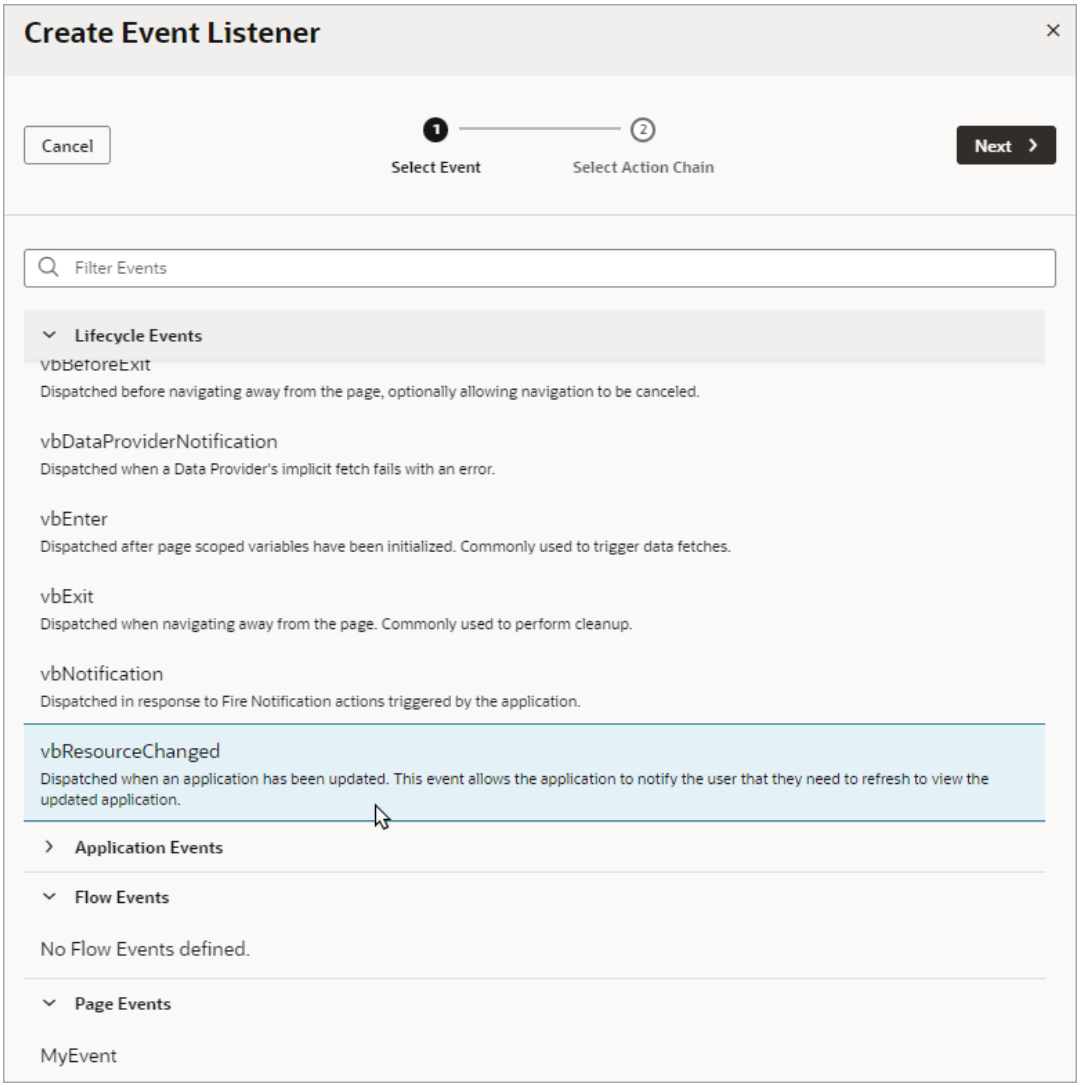
Keep in mind that one or more pages make a flow and each flow has its own lifecycle.

This table describes the lifecycle events you can use to start action chains:

Lifecycle Event	Description
`vbBeforeEnter`	Triggered before navigating to a page. Commonly used when a user does not have permission to access a page and to redirect the user to another page (for example, a login screen). Because this event is dispatched to a page before navigating to it, you can cancel navigation by returning an object with the property `cancelled` set to `true` (`{ cancelled: true }`). For this event, you can use these variable scopes to get data: • `$application`: All application variables can be used in the event's action chain • `$flow`: All parent flow variables can be used in the event's action chain • `$parameters`: All page input parameters from the URL can be used in the event's action chain
`vbEnter`	Triggered after container-scoped variables have been added and initialized with their default values, values from URL parameters, or persisted values, and is dispatched to all flows and pages in the current container hierarchy and the application. Commonly used to fetch data. For this event, you can use these variable scopes to get data: • `$application`: All application variables can be used in the event's action chain • `$flow`: All parent flow variables can be used in the event's action chain • `$page`: All page variables can be used in the event's action chain
`vbBeforeExit`	Triggered on all pages in the hierarchy before navigating away from a page. Commonly used to warn if a page has to be saved before the user leaves it, or to cancel navigation to a page (say, because a user doesn't have permissions to view that page) by returning an object with the property `{ cancelled: true }`.
`vbExit`	Triggered when navigating away from the page and is dispatched to all flows and pages in the current container hierarchy being exited from. Commonly used to perform cleanup before leaving a page, for example, to delete details of a user's session after logout.
`vbAfterNavigate`	Triggered after navigation to the page is complete and is dispatched to all pages and flows in the hierarchy and the application. The event's payload (`$event`) is an object with the following properties: • `currentPage <String>`: Path of the current page • `previousPage <String>`: Path of the previous page • `currentPageParams <Object>`: Current page parameters • `previousPageParams <Object>`: Previous page parameters
`vbNotification`	Triggered when a Fire Notification action is fired by the application.
`vbResourceChanged`	Triggered when an application has been updated. Commonly used to notify the user that they need to refresh to view the updated application.
`vbDataProviderNotification`	Triggered when a Data Provider's implicit fetch fails with an error.

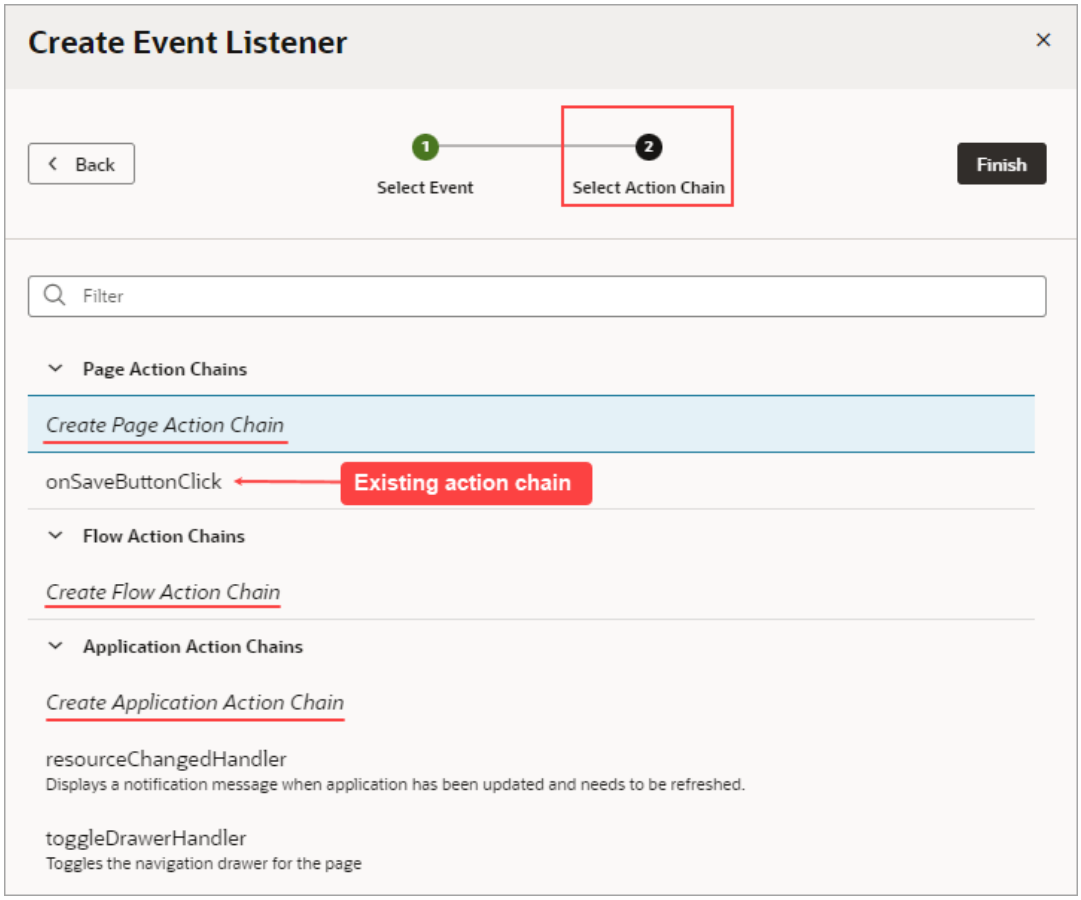
To start an action from a lifecycle event:

1. Open the Event Listeners tab for the page containing the event you want to trigger an action chain for.
2. Click **+ Event Listener**.
3. In the Create Event Listener wizard, expand the Lifecycle Events category and select the event you want to trigger an action chain for. Click **Next**.



4. Select the action chain you want to trigger. You can select any action chain that is scoped for the artifact. 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.

   If you want to create a new action chain now, you can click ⊕ and enter an ID for the new action chain, which you can edit later in the editor. Click **Finish**.



After you create an event listener, you can click **Add Action Chain** for the lifecycle event if you want it to start additional action chains.

Start an Action Chain By Firing a Custom Event

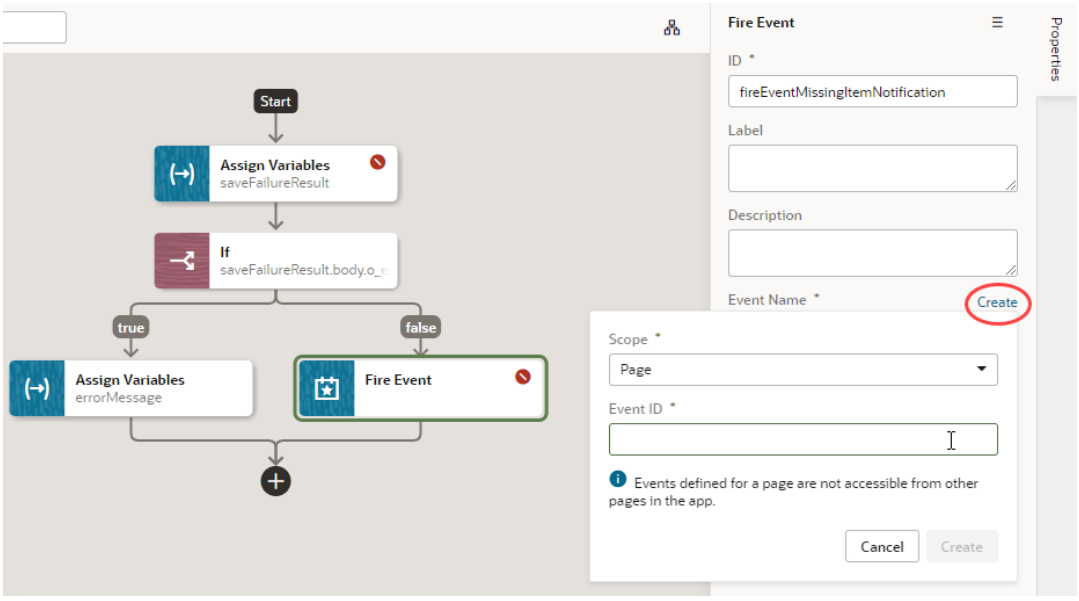
You can start an action from another action chain using a custom event, which is triggered by the Fire Event action in an action chain. Typically, you use a custom event when you want to trigger a notification, like displaying a pop-up window with a message, or perhaps to transform some data. After creating a custom event, you need to create an event listener for it to start the action chain.

Each custom event has a Behavior property, which you can use to set whether action chains run serially or in parallel. The default behavior is "Notify", which allows the action chains to run in parallel. For more about setting an event's Behavior property, see Choose How Custom Events Call Event Listeners.

To start an action chain with a custom event:

1. Open the **Actions** tab for a page, flow, or application.
2. Select the action chain you want to edit. The action chain opens in the Action Chain editor. If you want to create a new action chain, click **+ Action Chain**.
3. In the Action chain editor, drag the **Fire Event** action from the palette and drop it in the action chain where you want the event to occur.
4. In the Fire Event action's Properties pane, specify the Event Name.

If you are using the Fire Event action to trigger a new custom event, click **Create**, enter an Event ID, and specify the event's scope. If you want to trigger a custom event that already exists, you can select it in the drop-down list.



After creating or selecting the event, you can click **Go to Custom Event** in the Properties pane if you want to edit the event's Behavior or Payload properties in the Events editor.

5. Create an event listener for your event:
   a. Open the **Event Listeners** tab and click **+ Event Listener** to open the Create Event Listener wizard.
   b. Select the custom event you added to your action chain. Click **Next**.
   c. Select the action chain you want the event to trigger. Click **Finish**.

> **Tip:**
>
> If you're on the **Events** tab, simply right-click your custom event and click **Create Event Listener** to create an event listener, called `(eventId)Listener`, in the same scope as the event.

12

Work With Events and Event Listeners

Events trigger actions when a user or the browser interacts with your application. For example, when a user clicks a button to navigate to the application's home page, Visual Builder raises an event that triggers the navigate action.

There are many types of events: lifecycle events, variable events, component events, even custom events. Event types are more about how they occur and are all used to execute action chains in your application. The application reacts to events through event listeners, which declaratively specify the action chain to execute when the event occurs.

Define Events in Your Application

An event occurs when something happens in your application. Some examples are when a page loads (lifecycle event), a button is clicked (component event), and when a variable's value changes (variable event). An event's type depends on how it is triggered; for example, a button or a menu would trigger a component event.

How you define events and event listeners depends on the type and scope of the event. This table describes the types of events and how you can define them:

Type of Event	Description	How to Define
Component events	An event associated with a UI component in a page, including those in dynamic components. It's possible to choose which event the component triggers, but available events will depend on the component. For example, an event like `ojAction` is available to a button but not to an input text field.	Define a component's event and event listener in the component's Properties pane in the Page Designer. See Start an Action Chain From a Component.
Variable events	An event specific to a variable that occurs when the value stored in the variable changes. The only available variable event is `onValueChanged`.	Define a variable's event and event listener in the Variables editor. See Start an Action Chain When a Variable Changes.

Type of Event	Description	How to Define
Custom events	A user-defined event to start an action chain. It can be triggered by a Fire Event action in an action chain, or by using an event helper's `fireCustomEvent()` method in a module function (JavaScript function). See: • Module Function Event Helper • Start an Action Chain By Firing a Custom Event	Create a custom event in the Action Chains editor or in the Events editor, then create an event listener for your custom event in the Event Listeners editor. See Start an Action Chain By Firing a Custom Event. **Tip:** You can also create an event listener for a custom event directly from the Events editor. Simply right-click a custom event in the Events editor and click **Create Event Listener** to create an event listener, called *(eventId)*`Listener`, in the same scope as your event.

Type of Event	Description	How to Define
Lifecycle events	Predefined events that are automatically triggered during a page's lifecycle: • `vbBeforeEnter` is triggered before navigating to a page. • `vbEnter` is triggered when all flow or page variables have been initialized. • `vbBeforeExit` is triggered before leaving a page. The `vbBeforeExit` event optionally allows navigation to be canceled (say, when a page has unsaved changes) by returning an object with the property `cancelled` set to `true`. When using the browser (back or forward button), the event's payload is an object containing default parameter values. See the vbBeforeExit description in the *Oracle Visual Builder Page Model Reference*. • `vbExit` is triggered before leaving a flow or page. • `vbAfterNavigate` is triggered when navigation to the page is complete. You can associate action chains with these events to augment a page or flow's default lifecycle. For example, if you want to initialize some component variables when a page opens, you can create an event listener in your artifact that listens for the `vbEnter` event, then set the event listener to trigger an action chain that assigns values to the component's variables.	Create an event listener for a lifecycle event in the Event Listeners editor. See Start an Action Chain From a Lifecycle Event.

Create Event Listeners for Events

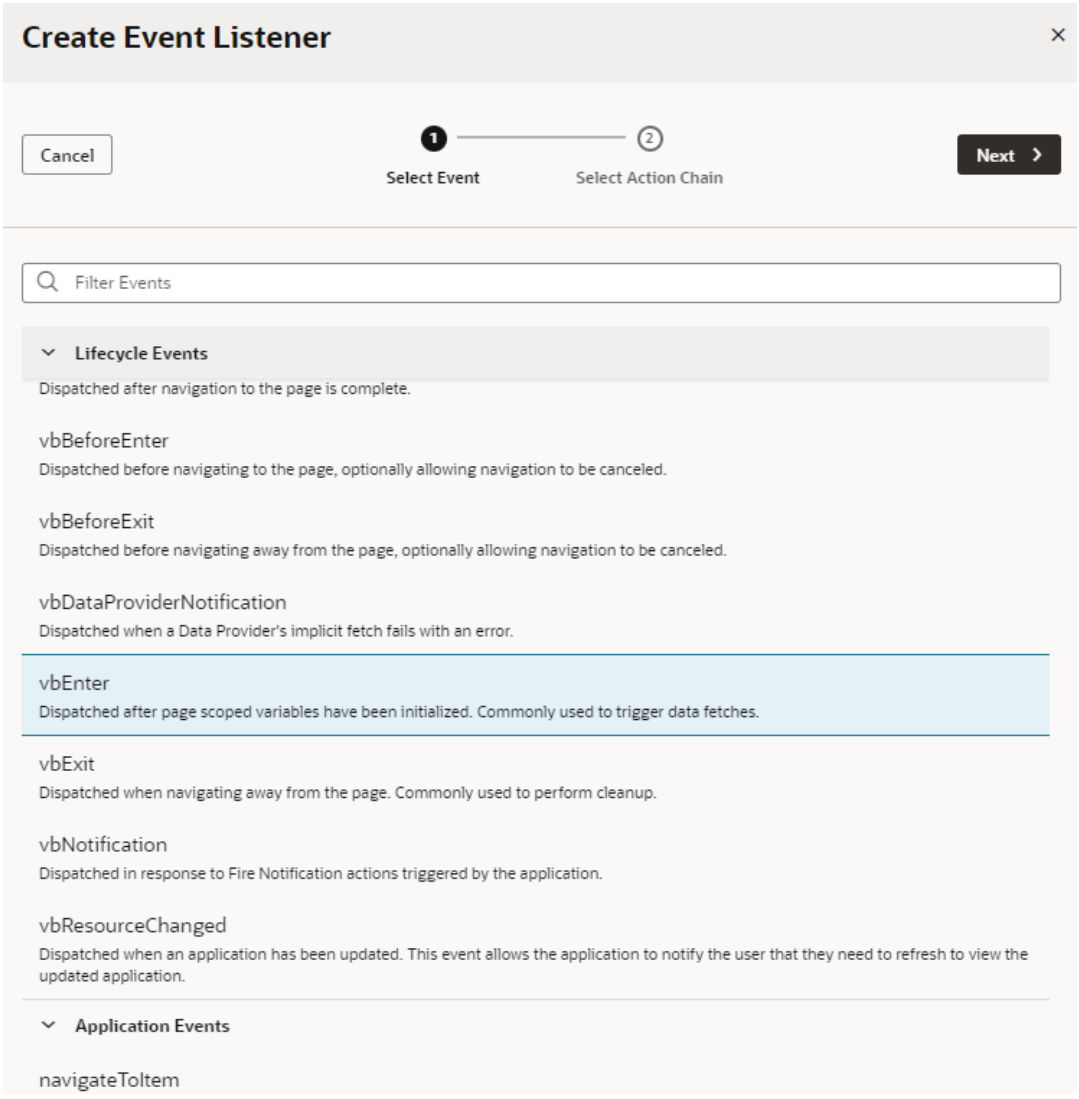
When an event (such as a button click) occurs in a page, it can start one or more action chains if an event listener is "listening" for it. To create an event listener, you select the event it should listen for and the action chain you want it to trigger. You can create event listeners for custom events as well as for predefined lifecyle events.

An action chain can be started by multiple event listeners, so you might have a `SaveData` action chain that can be started by two different event listeners listening for two different events.

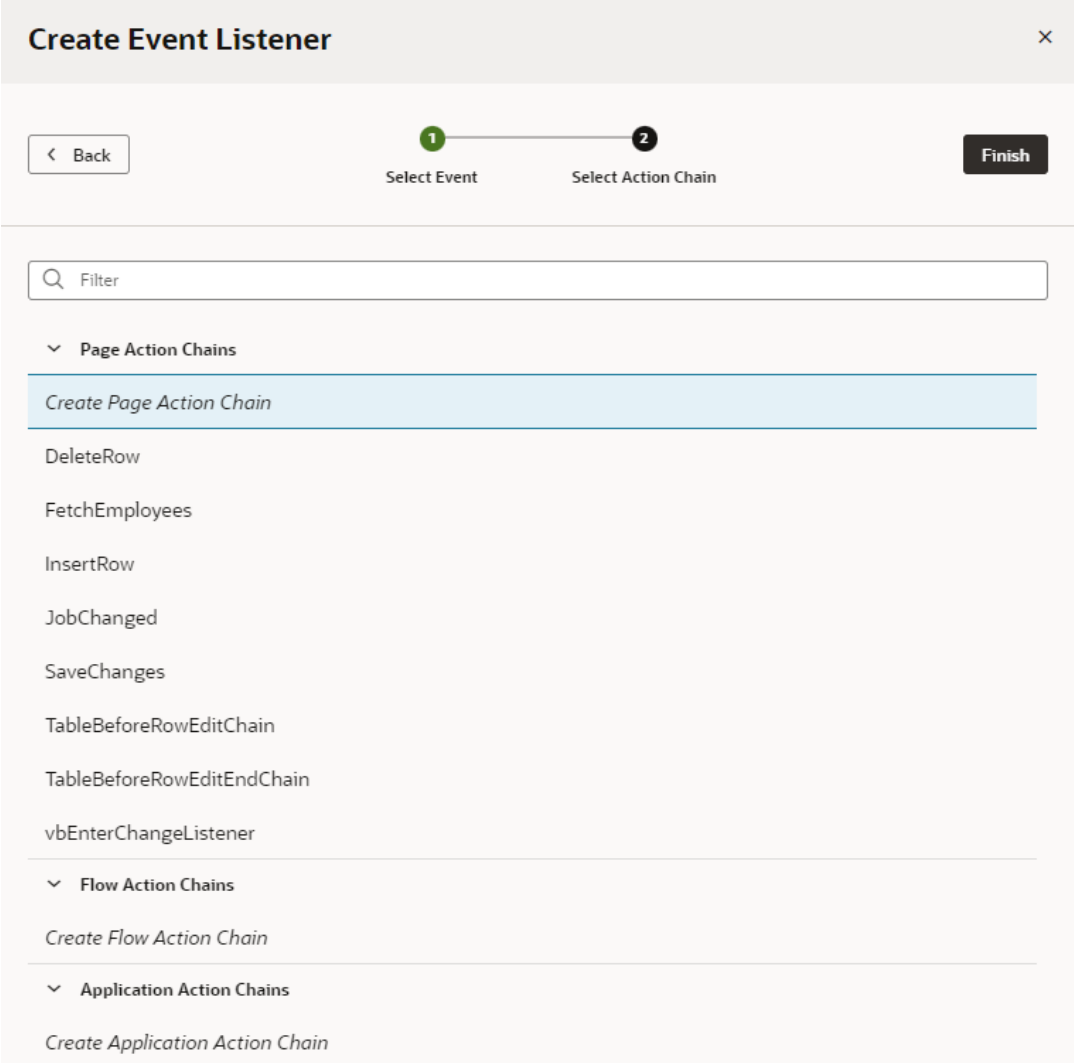
To create a listener for an event and associate it with an action chain:

> **Tip:**
>
> If you're working with an existing custom event on the Events tab, you can create event listeners directly from there. Simply right-click the custom event and click **Create Event Listener** to create an event listener, called `(eventId)Listener`, in the same scope as the event. Note that this option isn't available for custom events that fire to containers.

1. Open an artifact's **Event Listeners** tab, then click **+ Event Listener**.
2. In the Create Event Listener wizard, select the event you want to trigger an action chain. Depending on where you're creating the listener, your list might include page events, flow events, and application events, in addition to lifecycle events.



3. Click **Next**.
4. Select the action chain you want to trigger, or create a new action chain which you can edit later.

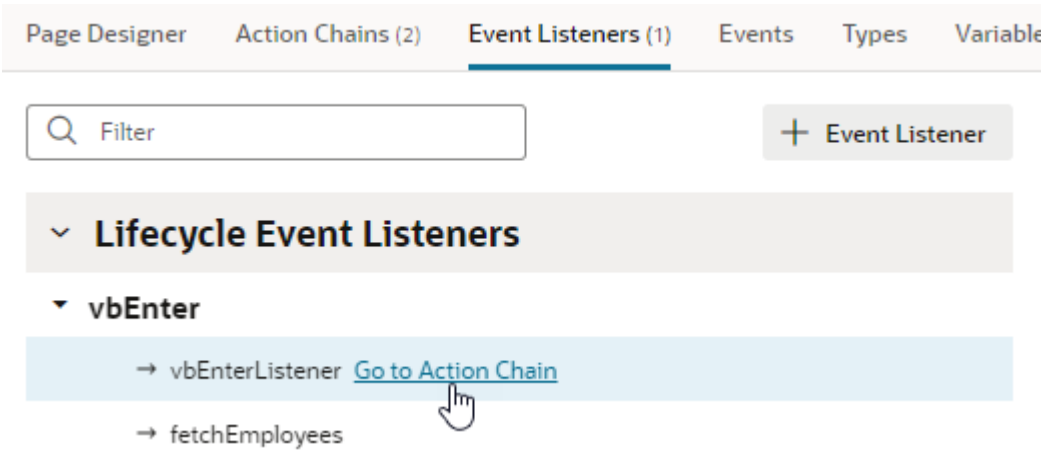


While an event listener can trigger multiple action chains, you can add only one action chain at a time in the wizard.

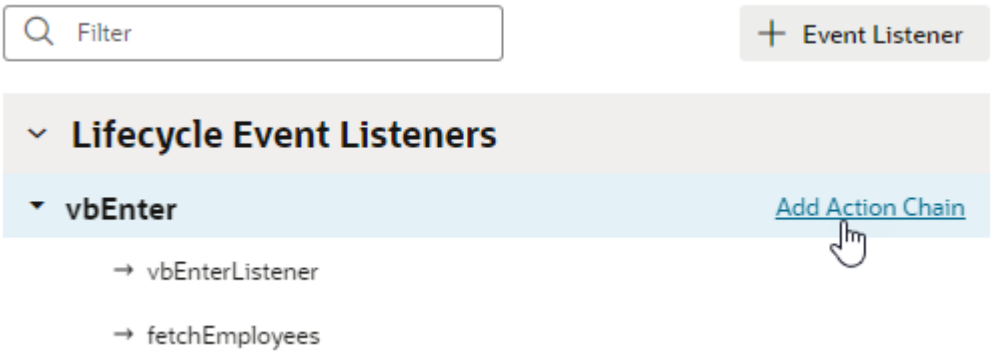
5. Click **Finish**.

If you chose to create a new action chain, look for the associated event listener, typically prefixed with the event you chose. For example, if you selected **vbEnter** as the event, a new event listener called **vbEnterListener** is created under Event Listeners.

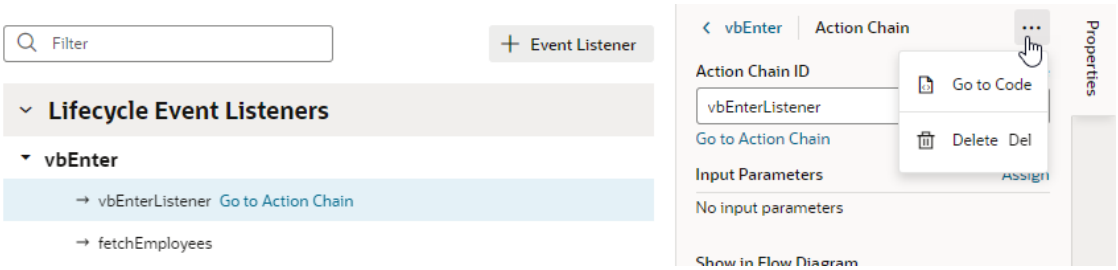
To edit an action chain associated with an event listener, move your cursor over the action chain (or select it), then click **Go to Action Chain** to open the action chain in the Action Chains editor.



You can add additional action chains to an event listener any time you want. Simply move your cursor over the event name (or select it), then click **Add Action Chain** to relaunch the Create Event Listener wizard.



If you want to delete an event listener, or remove an action chain triggered by an event listener, select it and click **Delete** from the Actions menu in the Properties pane. Deleting an action chain in the Event Listeners tab means it will no longer be triggered by the listener, but it won't delete the actual action chain.



> **Note:**
>
> A page-level or component event listener includes a **Show in Flow Diagram** property to surface the listener in the Flow Diagram, allowing you to create action chains that bind to an existing event listener. See Bind an Action Chain in the Flow Diagram to an Existing Event Listener.

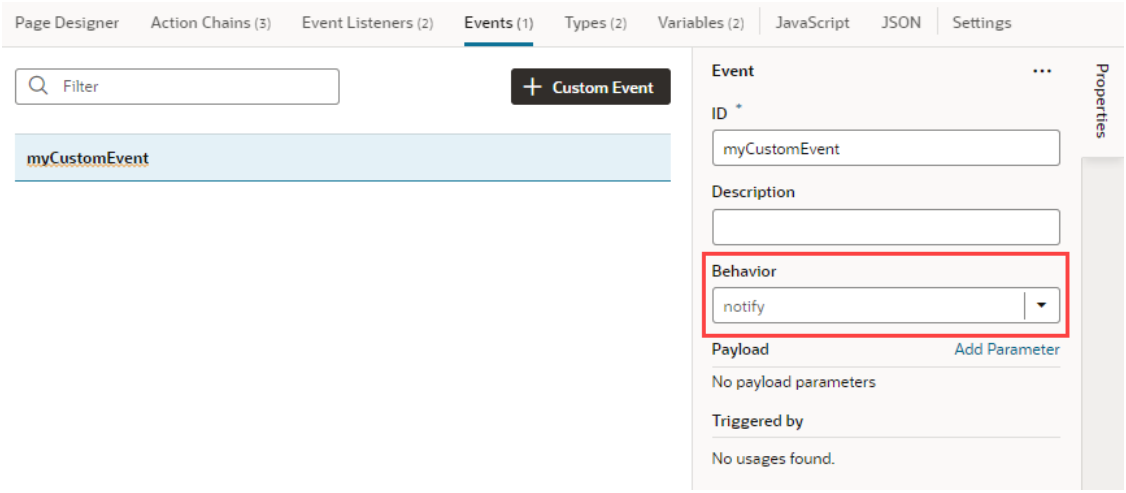
Choose How Custom Events Call Event Listeners

Each custom event has a behavior type that defines how the event listeners will be called in relation to each other, whether the result for the listener is available, and what form the result would take.

The behavior type does not define the order in which listeners are called, but whether the listener is called serially or in parallel, that is, whether the action that raised the event waits for a listener resolution, and what the "result" of the listener invocation looks like. So in this case, "serially" means:

- For a single event listener (in a container), all the event listener chains are called sequentially, in a declared order. This means that a listener action chain is not called until the previous action chain has finished (and resolved, it returns a Promise)
- The event listeners for the next container's listeners are not called until the listener action chains for any previous container's event listeners have finished (and resolved, it returns a Promise)

You can choose the behavior type of a custom event in the Properties pane of the Events editor:



A custom event will have one of the following behavior types:

Behavior Type	Description
notify	**Parallel**: The event is triggered but the application does not wait for the extension to process it. Chain results are **not** available to the action (or helper) that fired the event (because the listeners are called without waiting). This is the **default** behavior.

Behavior Type	Description
notifyAndWait	**Serial**: Each action chain listener must complete (and resolve any returned Promise, if any), before another event listener action chain is called. Chain results are **not** available to the action (or helper) that fired the event.
checkForCancel	**Serial**: Each action chain listener must complete (and resolve any returned Promise, if any), before another event listener action chain is called. If any of the listener's chains returns a "success" with a payload of `{ "stopPropagation": true }`, the application will stop calling event listeners. Chain results are **not** available to the Action (or helper) that fired the event.
transform (deprecated)	Use **transformPayload** instead. If your existing event listener is set to **transform**, it is recommended that you switch to **transformPayload**.
transformPayload	**Serial**: Each action chain listener must complete (and resolve any returned Promise, if any), before another event listener action chain is called. Chain results are available to the action, and the action can modify the chain's results before passing it back to another action following it.

Raise Fragment or Layout Events that Emit to the Parent Container

Layouts and fragments defined in your application are typically unaware of their parent container's context. This means that events defined within a layout or fragment are "listenable" only within the layout or fragment's scope. To make these events listenable on the parent container (say, a page or another container like a different outer fragment), you'll need to fire custom events that the parent can handle.

Consider this example: Say a fragment defines a form with a Save button. Any time a user updates the form's data and clicks the button, an `on-click` event triggers a REST call action that saves the updates. To make the update available on the page that consumes the fragments, you'll need a Fire Event action that emits its payload to the fragment container in your action chain. This makes it possible for the page to listen for this custom event, bind an event listener to the same event, and process the payload further if needed.

When a new custom event is fired from the page, keep in mind that the custom event (unlike a page event) "bubbles" up the container 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:

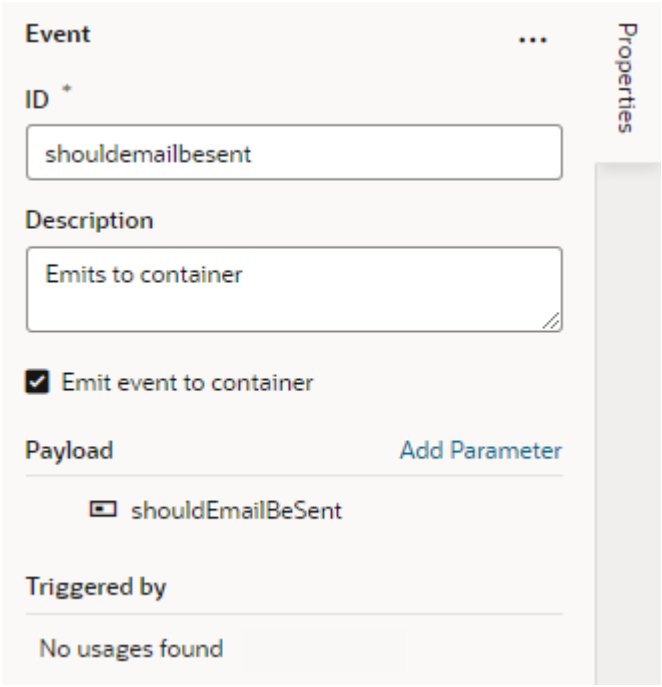
- The page from where the event is fired
- The flow containing the page
- The page containing the flow
- Recursively up the container, ending with the application.

To make a layout or fragment event listenable on the parent container:

1. Create a custom event that emits its payload to the parent container.
    a. In your layout or fragment's **Events** tab, click **+ Custom Event**.
    b. Enter an event ID (say, `shouldemailbesent`), then select the option to emit the event's payload to its parent component. For a layout event, select **Emit event to page**; for a fragment event, select **Emit event to container**. Click **Create**.

c. If necessary, select the event in the Events editor, then click **Add Parameter** next to Payload in the Properties pane to specify the payload that will be passed to the parent container.

d. Enter the payload parameter name, select its type, and click **Create**. In our fragment example, this might be a `shouldEmailBeSent` payload parameter of type boolean:



> **Note:**
>
> When an event is set to emit its payload to its parent container, its `propagationBehavior` property is set to `container` in the fragment or layout model. The default is `self`, indicating that the event can only be handled by event listeners defined in the layout or fragment.

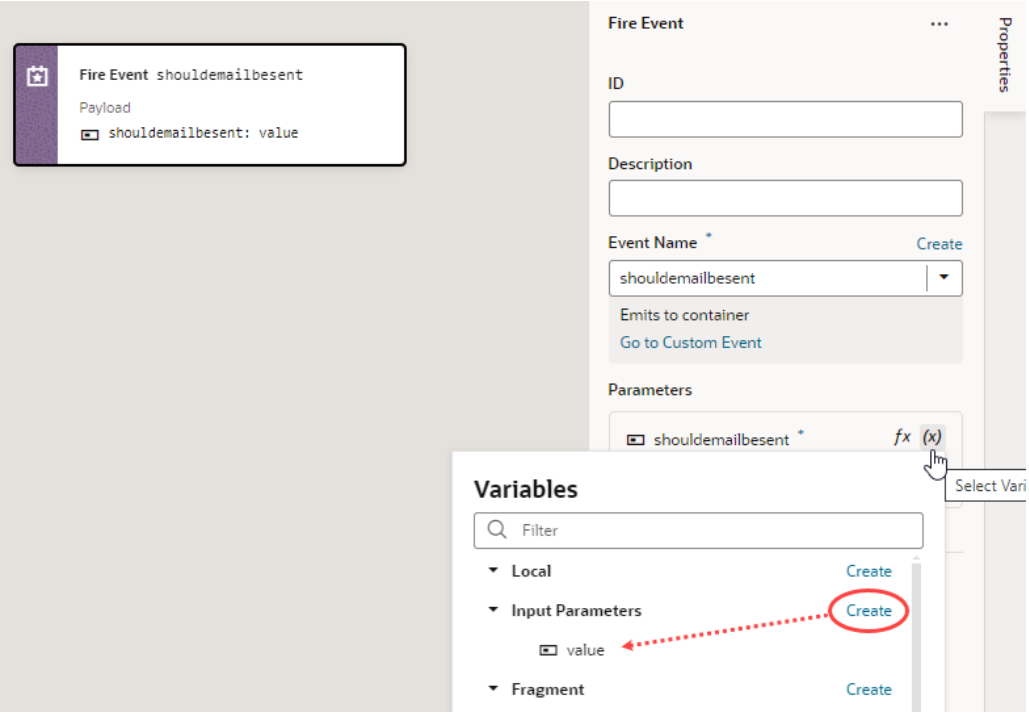
2. Create an action chain with the Fire Event action that will be triggered when the event occurs.

   a. Switch to the layout or fragment's **Action Chains** tab, click **+ Action Chain**, enter a ID, and click **Create** to create a new action chain. You can also select an existing action chain.

   b. In your action chain, drag and drop a Fire Event action.

   c. In the Fire Event's Properties pane, select the event to be fired (for example, `shouldemailbesent`).

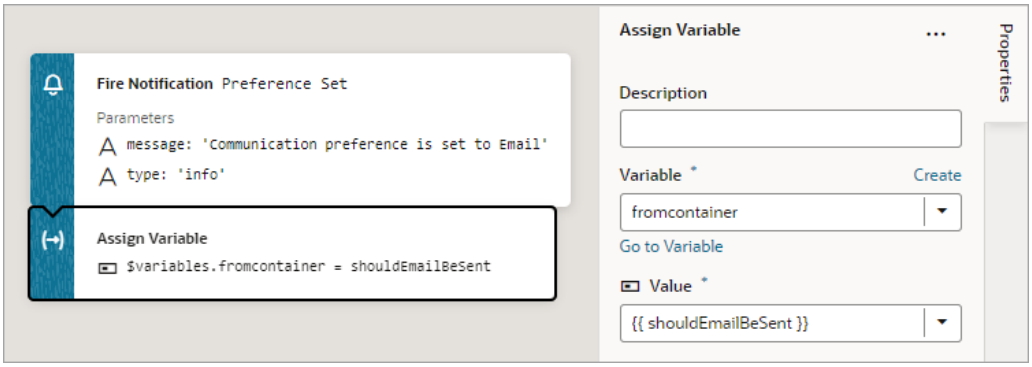
   d. Under Parameters, click ***(x)*** next to `shouldemailbesent` to open the variables picker, then create a string-type `value` under Input Parameters.



3. On the page that uses your layout or fragment, configure the parent container to handle the custom event.

   a. In the Page Designer, select the component (for example, a fragment) to open its Properties pane, click the **Events** tab, then click **+ Event Listener** and select the suggested custom event (for example, `shouldemailbesent`).

   b. Define the action chain that must be triggered when the event occurs. For example, you might want a notification to appear on the page when the user toggles the Switch in a fragment. To do this, you add a Fire Notification action, followed by an Assign Variable action to assign the action chain's value to a page-level variable.



13

Work With Application Resources

As you develop an application, you can import and export static resource files for use in your application's pages.

Resources are typically files that you import to support or add functionality to pages in your application. For example, when you want to use an image in a page, you can import the image as a resource into an `images` folder, then use an Image component on a page to reference the imported image.

By default, each web (and mobile) application contains a `resources` folder to store custom components, images, translation files, and other resources that can be used in the application. Here are the folders created by default for the following types of resource files:

Folder	Description
`actions`	Location for custom actions that you might define in your application. Right-click the folder to either create or import a custom action.
`components`	Location for custom web components that are installed to your application (by importing them in the Page Designer). You can also create a custom component here by clicking **Create Component**.

> **Note:**
>
> Importing custom components to the `resources/components` folder or creating them there makes them a part of your application. Because these components are not cached, you're likely to run into performance issues when they are downloaded each time you reload the Page Designer for preview, or at runtime when you publish an update to your app. As a best practice then, it helps to publish your components to a CDN (Content Delivery Network) or an external location that your browser can cache requests from. This is useful especially when you have multiple apps that use the same components. Talk to your administrator for site-specific information on how to publish these components externally.

`css`	Location for the `app.css` stylesheet that is linked from your application's pages. This stylesheet is empty by default. You can edit the stylesheet to add custom styling to page elements. See Add a Custom Style to a Component. The `app.css` file is not used to control the styling of Oracle JET components.
`images`	Default location for any custom images that you might want to add to your application. Images in the application's `resources` directory can be used in any page in the application. This folder contains two favicon files by default (`favicon-dark.ico` and `favicon-light.ico`), used by web and Progressive Web Apps. The favicon used depends on the mode the app uses (light or dark mode). Overwrite the default-provided favicons if you want to use your own favicons. 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 in the `resources` folder of the flow that contains the page.

Folder	Description
`js`	Location for external JavaScript files that you want to use in your application's pages.
`strings`	Default location for your translation bundles. See About Translation Resources. Flow and page artifacts can also contain a `strings` folder that stores translation bundles.

To work with resources at the visual application level, see Export and Import Application Resources.

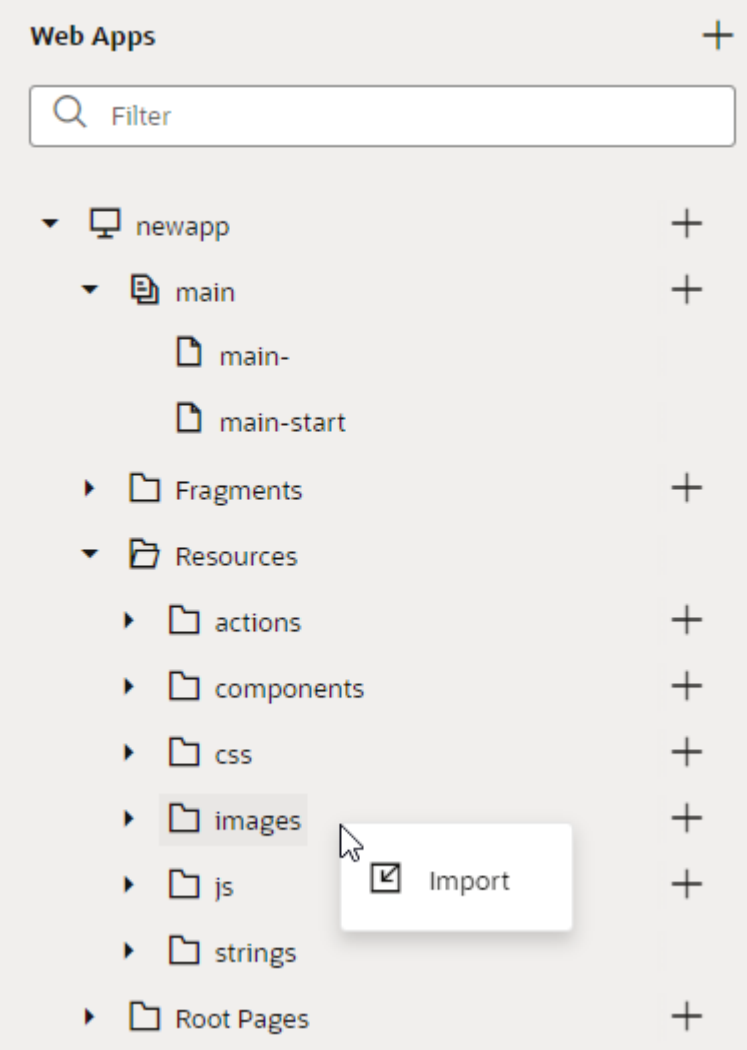
Import Resources

To add resources that you want to use in your application's pages, you import individual files or ZIP archives by using the Import command available when you right-click the resources folder in the Navigator or any artifact, or by dragging the file directly from your local file system onto a folder in the Navigator.

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. The import location is also important to make sure you're importing the resource file where you can access it. For example, it's possible to 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, so you won't be able to apply the image you've imported to a UI component.

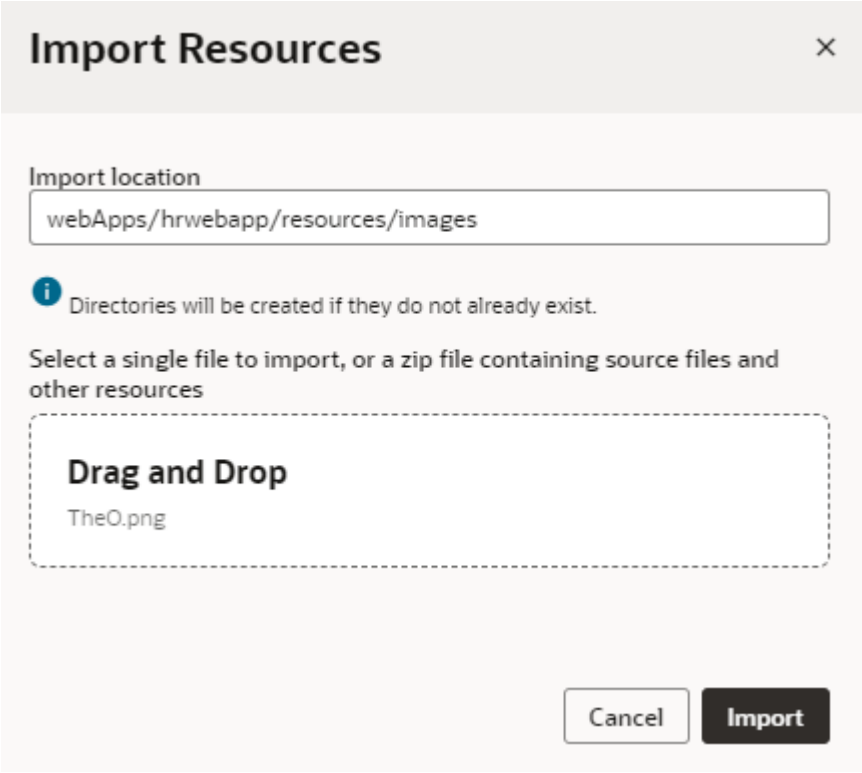
To import resources for use in an application:

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 select **Import**.



Alternatively, drag a file from your local file system onto the folder or artifact in the tree view 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, edit the path in the Import location field to create new folders.

4. Click **Import**.

   A confirmation appears and your resource file is added at the location you specified.

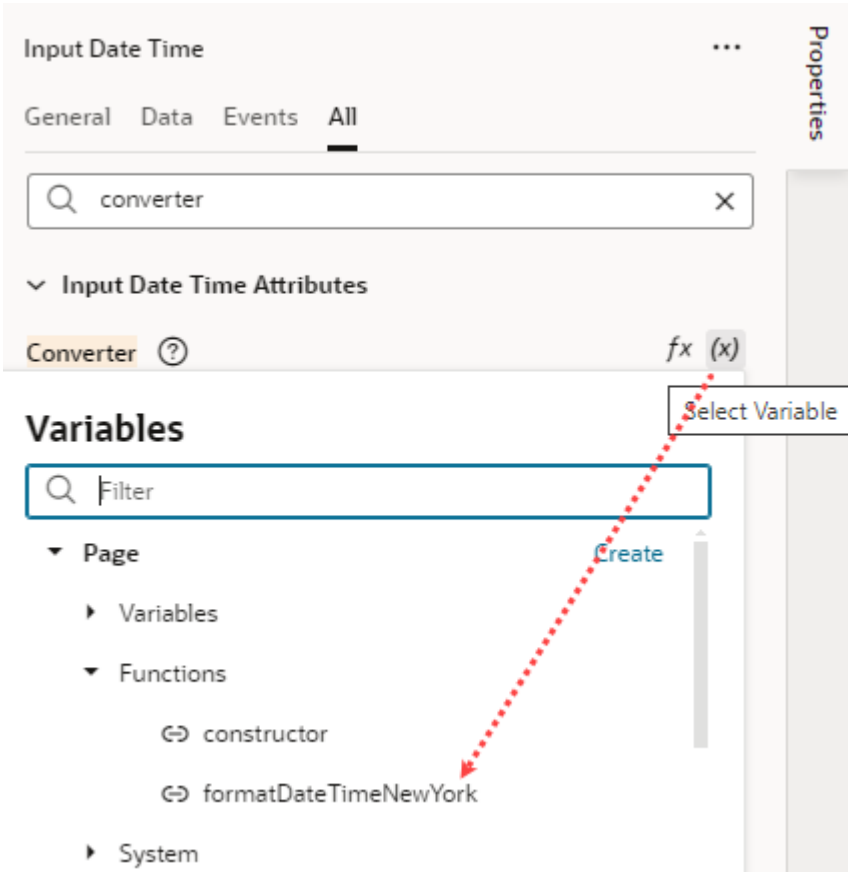
Manage Custom Component, CSS, and Module Imports

You can import resources such as custom CSS files, modules, and components to create "declarative" references to imported resources.

The **Imports** tab in the Settings editor enables you to manage resources imported for an application, flow, or page artifact. You can manage custom components, CSS files, as well as modules containing code that you want to call in your application. Let's consider some sample scenarios of when you'd want to use this tab:

- When your artifact includes components that are deprecated or no longer used, these component definitions stay intact in the artifact's metadata, but might be flagged by audits as a deprecated or unused component dependency. While you can resolve this issue by manually editing the JSON editor, you can use the Imports tab to manage these imports without potentially introducing errors.
- When you want to use custom CSS files for specific pages, you'll usually need to add an import statement to the page's HTML manually. By declaratively adding custom CSS files on the Imports tab, you can easily apply these imported CSS files to any page or pages in a flow without having to manually add an import to the HTML pages.
- When you want to use JavaScript modules at the application, flow, or page level to create custom functions within the module (say, the `IntlConverterUtils` utility function that lets you format a date field as an ISO string), referencing the module from the Imports tab makes it available for you to call in your application without having to add code to your JSON or JavaScript files.

You can call these functions in an action chain using the Call Function action and in a component's property, by selecting the function in the Expression editor or Variables picker in the properties pane, as shown here:



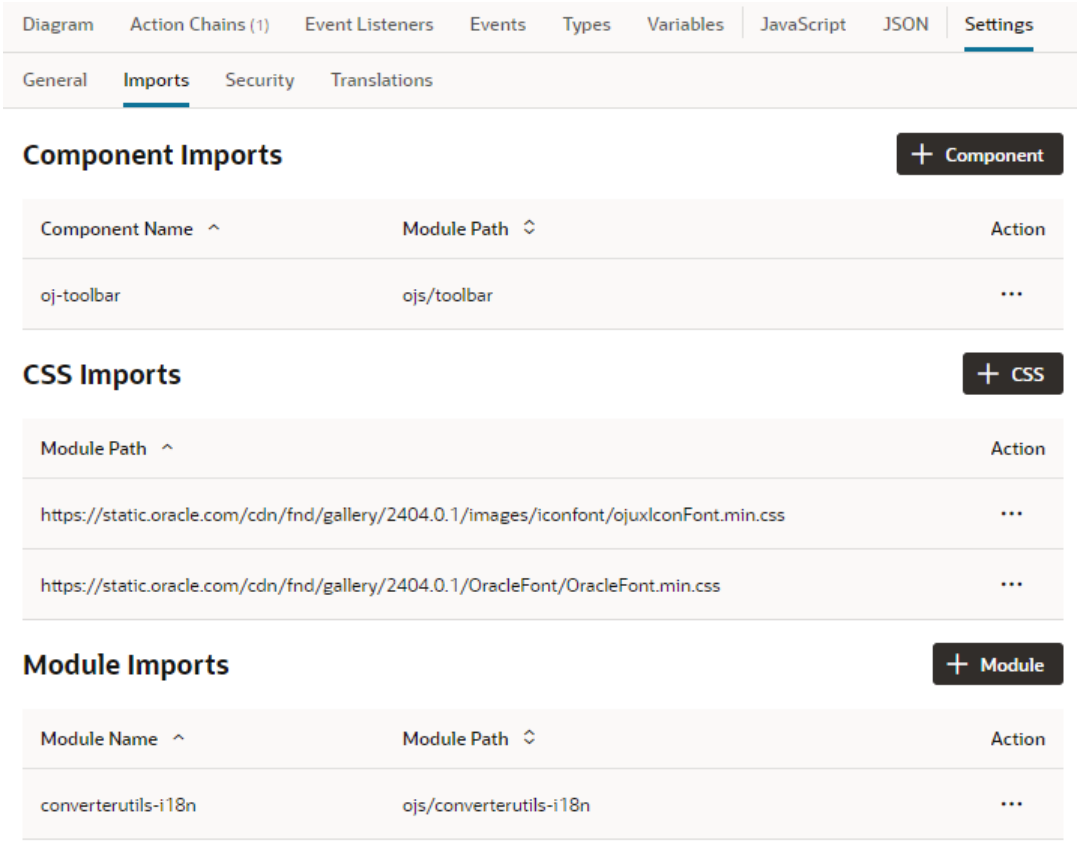
To manage imports for an application, flow, or page artifact:

1. Open the **Imports** tab in the Settings editor of an application, flow, or page artifact.
2. Import components, CSS, and custom modules:
   - To import custom components to your application, flow, or page artifact, click **+ Component**, then enter the component name and path to the component module. To manage an existing component, click its Actions menu and select **Edit** or **Delete**.
   - To reference CSS files in your application, flow, or page artifact, click **+ CSS**, then create a reference to an existing file, an external file, or a new file:
     - To create a reference to an existing CSS file in your resources folder, click **Existing**, then select the file from the drop-down list. (For information on how to add CSS files to your application's resources, see Work With Application Resources.)
     - To create a reference to an external CSS file (say, a font or an icon in an external resource that you'd like to use), click **External**, then specify the path to the file.
     - To create a reference to a new CSS file, click **New** and specify the name and path to the new file (which will be created for you).

     To manage an existing CSS file, click its Actions menu and select **Edit** or **Delete**.

- To reference custom modules that contain code you want to call in your application, flow, or page artifact, click **+ Module**, then enter the module name and path to the module.
  To manage an existing module, click its Actions menu and select **Edit** or **Delete**.

Here's an example of imports at the flow level:



3. Click **Create** or **Create & New** to repeat the action.

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 Properties pane 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 Properties pane.

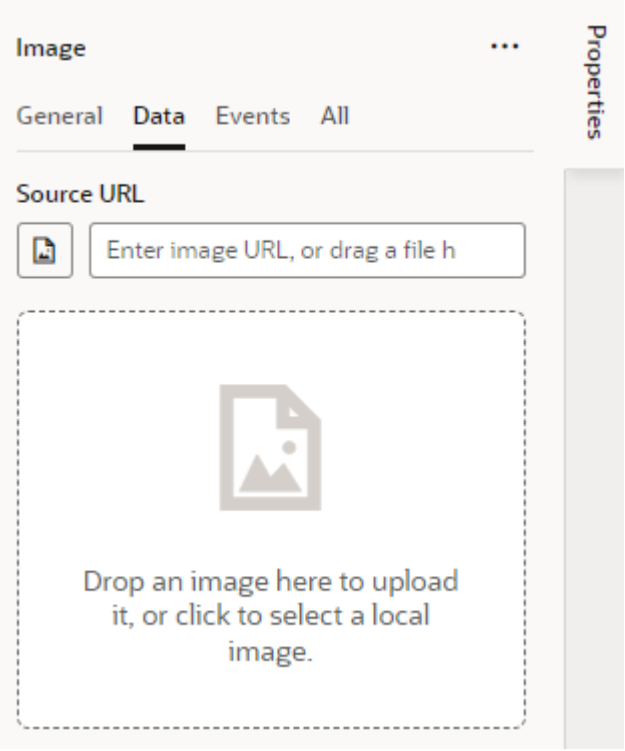
To import images into the Image Gallery:

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 Properties pane and click the Image Gallery icon ( ).

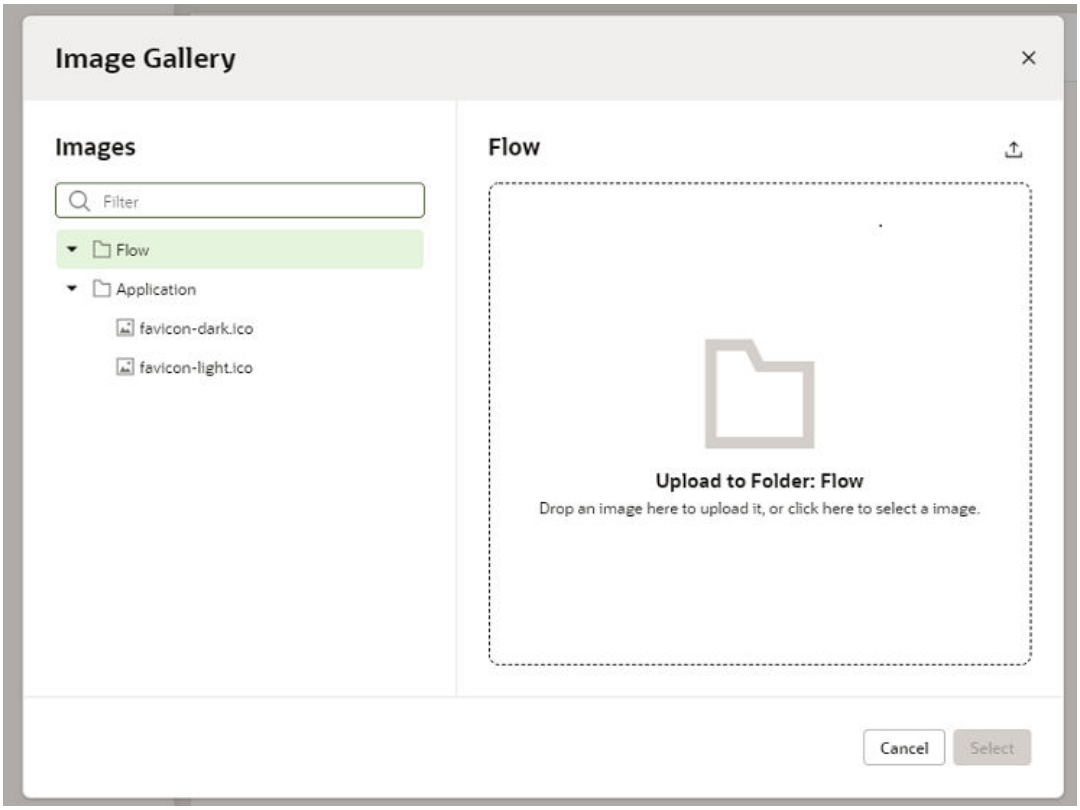
   When an Image component is selected on the canvas, the Data tab in the Properties pane displays a Source URL field for the path to the stored image. The field is empty when no image has been defined.



   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.

3. Select **Application** in the Images panel of the Image Gallery.

   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. In the Application section, two favicon images also appear that Visual Builder uses for web and Progressive Web Apps. Overwrite these to use your own favicon images. 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.

14

Work With Code

Most application development in Visual Builder is visual and declarative. Sometimes though you might want to update your application's source code or add custom code to make your application richer—and code editors in the Designer let you do just that.

The HTML, JavaScript, JSON, and Code view editors give you direct access to the code created by visual tools when you develop your application. For example, when designing a page, you can choose to directly edit the source code of the HTML, JavaScript, and JSON files used to describe a page's layout and behavior. Here's a look at the different code editors and what each is used for:

Tab	Description
Code view in Page Designer	Displays a page's HTML. See Add a Component Using Code Completion.
HTML	Displays an application's `index.html` file.
JavaScript	Displays JavaScript functions at the application, flow, or page level, as well as at the layout and fragment level. See Work With JavaScript.
JSON	Displays the JSON file that describes the artifact's metadata (including variables and action chains) at the application, flow, or page level, as well as at the layout and fragment level. See Work With JSON.

Visual Builder's code editors are based on Monaco, a JavaScript library bundled from the Visual Studio Code source, which provides a variety of code-editing features, including tooltips and hints, parameter information, and code completion. See how you can trigger code insights in each editor.

Work With JavaScript

Define your own JavaScript functions to extend an application's functionality for your business needs. For example, you can add a custom JavaScript function to validate whether required fields in a form have values, or to calculate the result of an add or multiply operation.

JavaScript functions are defined within the scope of a `module`, which can be at the application, flow, or page level. An `AppModule` contains one or more flows, each with its own `FlowModule`, and each flow can have one or more pages, each with its own `PageModule`. Within a page, there can be several UI events, each of which is typically associated with an action chain.

If your function will only be called in a page (say, to load some data when the page loads), you can define it within the `PageModule`. If you want multiple pages to call a function (say, to load libraries for customizing navigation elements or custom web components), you'll need to define the function within the `FlowModule` or the `AppModule`. If you're working with a fragment or layout, you'd use `FragmentModule` or `LayoutModule`. Functions defined here are available only within the scope of the layout or fragment.

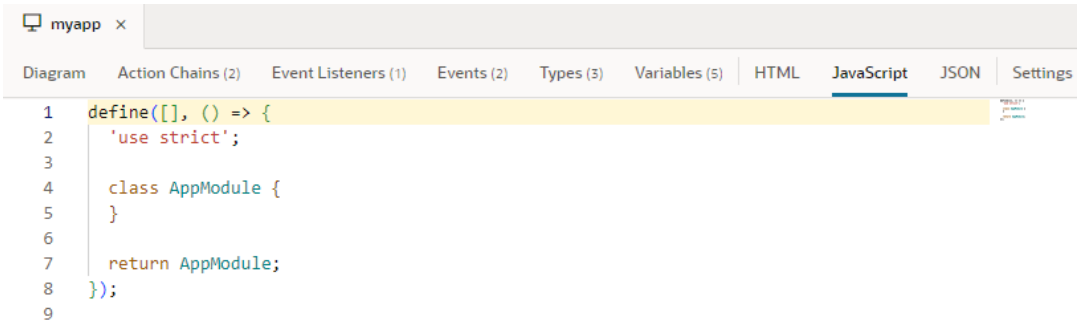
You can also import and reference third-party JavaScript libraries whose functions, objects, and variables you want to use in your custom code. VB Studio also supports RequireJS, a JavaScript file and module loader that simplifies the task of managing library references.

Add a Custom JavaScript Function

To add a custom JavaScript function, you define the function within the Module class provided in the JavaScript editor for your page, flow, or application. You can also do this for layouts and fragments.

1. Open the artifact for which you want to add a JavaScript function, then select the **JavaScript** tab.

   For example, to use a JavaScript function in multiple pages, you can define the function at the app level or at the flow level for those pages. To define the function at the application level, select the application node, then click the **JavaScript** tab:



   JavaScript functions are defined in different files based on the artifact's scope. An application uses the `app-flow.js` file, a flow uses *`flow-name`*`-flow.js`, and a page uses *`page-name`*`-page.js`. Additionally, a layout uses `layout.js` and a fragment uses *`fragment-name`*`-fragment.js`.

2. Define your JavaScript function within the Module class (`AppModule`, `FlowModule`, or `PageModule`) provided in the JavaScript editor. If you're working with a fragment or layout, you'd use `FragmentModule` or `LayoutModule`.

   To define an app-level function, for example, define your function within the `AppModule` class:

```
define([], () => {
  'use strict';

  class AppModule {
  // write your function here
  }

  return AppModule;
});
```

   Here's an example of an `AppModule` function that takes a string, appends some text to it, then returns that string:

```
define([], () => {
  'use strict';

  class AppModule {
   // Code for a custom AppModule method
```

```
    myAppModuleMethod(s) {
      return s + " has visited my AppModule method";
    }
  }

  return AppModule;
});
```

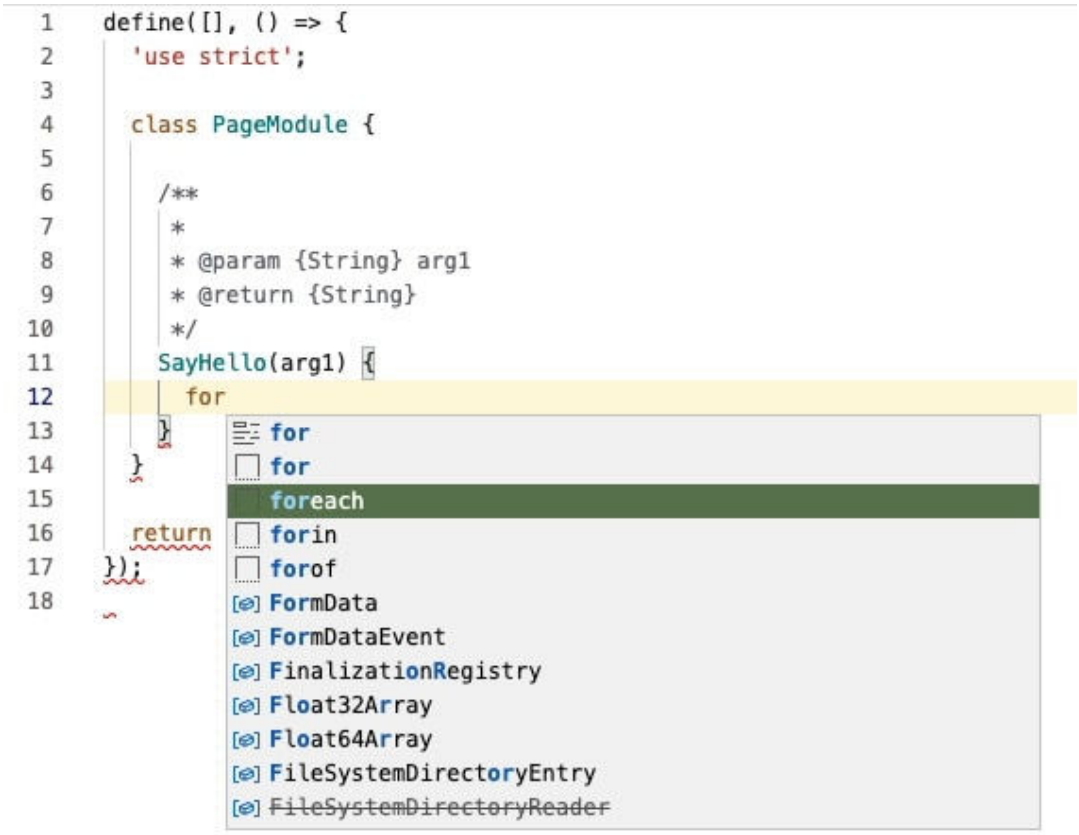
If any function within the class needs to access the application context, make sure you create a constructor for the class and include the context as an input parameter:

```
constructor(context){}
```

Here's another example where two functions have been created in the `PageModule` class: a constructor and a module function. When the page is opened, the corresponding instance of the `PageModule` class (shown below) is created for the page. Also, the instance's constructor is automatically called and the application context passed to the constructor:

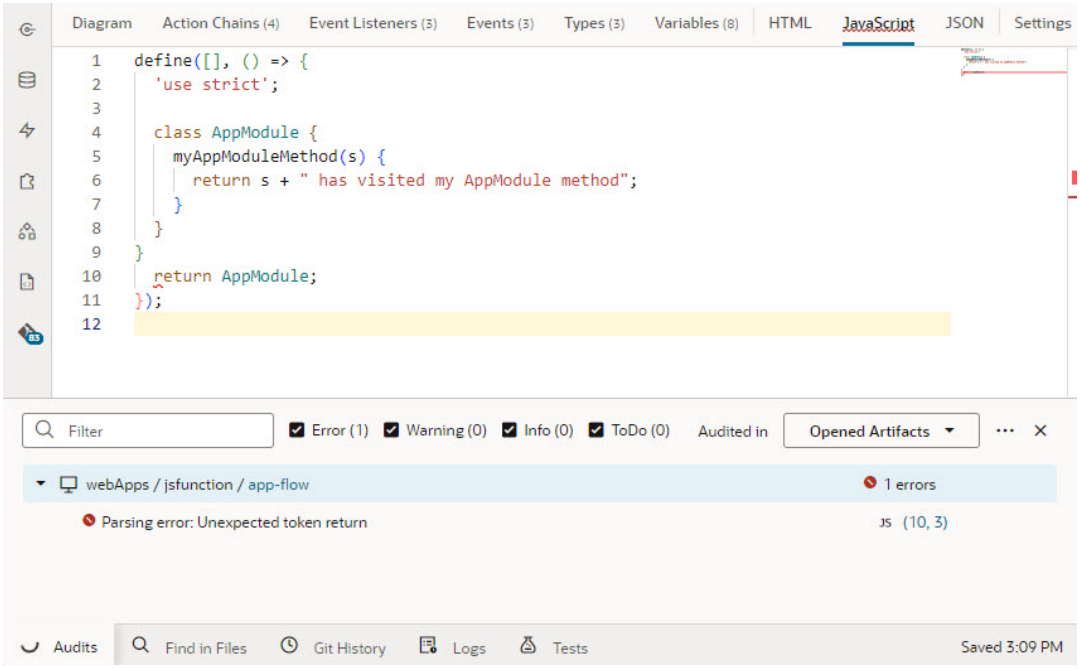
```
define([], () => {
  'use strict';

  class PageModule {

    constructor(context) {
      this.eventHelper = context.getEventHelper();
    }

    fireSomeCustomPageEvent() {
      this.eventHelper.fireNotificationEvent(
           {summary: 'Summary here', message: 'Message here,'});
    }
  }

  return PageModule;
});
```

As you write your code in the JavaScript editor, take advantage of suggestions that provide code-completion capabilities. This includes code snippets for common JavaScript structures, such as "for" and "while" loops and conditional statements. For example, typing `for` in the editor will show you various "for" loop structures:



Selecting a structure will let you easily switch the variables in the structure.

3. Watch for syntax errors in your code. Lines with syntax issues are flagged in the right margin, which you can then resolve from the Audits pane. A light bulb icon in the left margin indicates a hint that you can use to correct invalid code.



For help with JavaScript syntax, see https://developer.mozilla.org/en-US/docs/Web/JavaScript#reference. These additional resources can be helpful as well:

- https://www.w3schools.com/js/js_es6.asp
- https://www.javascripttutorial.net/es6/

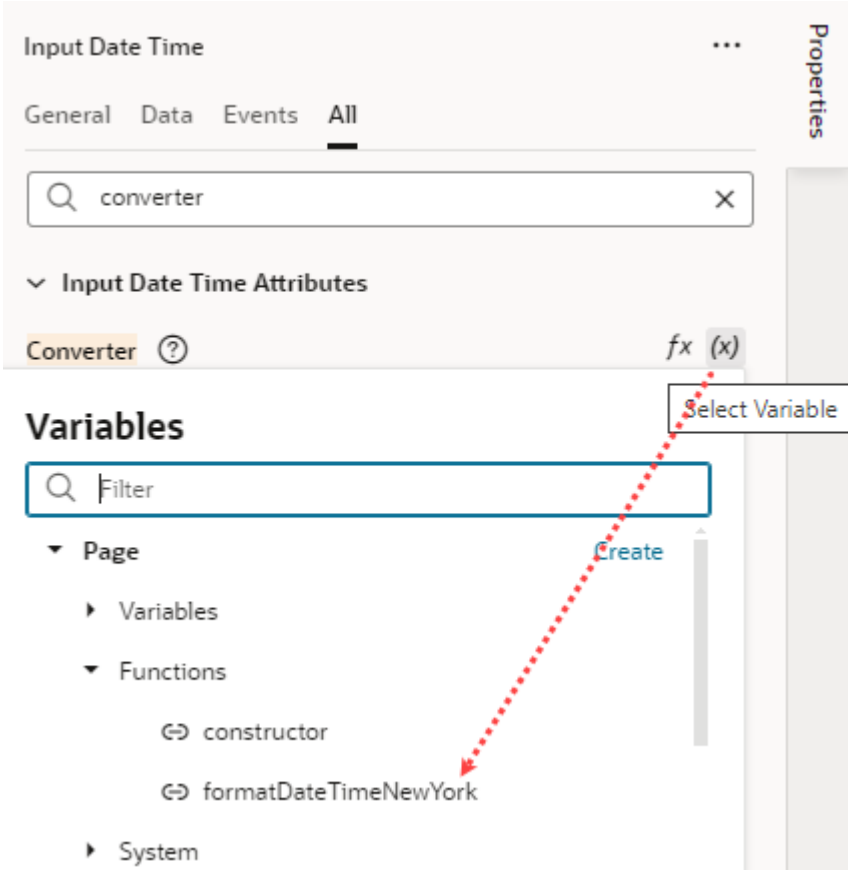
After you've defined your custom JavaScript functions, you can call them in action chains as well as UI components:

- In an action chain, select the function in the Call Function action.

  Let's say your UI requires you to enter some text in an input component, then click a button to call a function that passes that text, modifies it, and binds it to an output component. The input component is bound to an InputVariable and the output component is bound to an OutputVariable. You now build an action chain that's triggered when the button is clicked, where a Call Function action calls the custom function and maps its input parameter to the InputVariable, followed by an Assign Variable action that assigns the output of the function to the OutputVariable:

- In a component's Properties pane, select the function in the Expression editor or Variables picker of a property.

  Let's say your UI fetches data from a REST service and you want to convert some values shown in an Input Date Time component. By adding a converter function that uses the IntlDateTimeConverter API, you can select your custom function in the Variables picker to convert the value to a suitable date-time format:



To write efficient expressions that handle situations where a referenced field might not be available or the field's value could be null, see How Do I Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null?

Use Variables with a JavaScript Module

You can't directly get or set variables from within your JavaScript modules. However, you can use the Call Function action to access your JS module. This action takes an array of parameters which can include variables and can return a result that you can assign to a variable.

This approach ensures that the variable has a consistent state from the beginning to the end of your action chain's execution.

To "get" a value, pass the variable in as a parameter to the module function that you are calling using a Call Function action in the action chain.

To "set" a variable based on the return value from that Call Function action, use an Assign Variable action to copy the result of the function into the desired variable in whatever scope.

Work With Third-Party JavaScript Libraries

It’s possible to reference third-party JavaScript libraries in Visual Builder when you want to use functions, objects, and variables within that library in your custom code.

To reference external JavaScript libraries in your code, you need to import the JavaScript library to your application's resources, then add custom code to reference the file to be loaded in the module.

1. Import the JavaScript library as an archive to your application.

   a. In the Web Apps pane, go to the **Resources** folder, right-click **js**, and click **Import**.

      If you're using Source view, go to `webApps/<webapp>/resources/js` and click **Import**.
      You can choose to import your files directly to the `resources` folder, but it's best practice to keep all your JavaScript files in the `resources/js` folder.

   b. Select the archive you want to import and click **Import**.

2. Define your custom code and reference the file to load into the module. To do that, you use a `define` statement, which provides the path to the file and the alias with which you refer to the imported library in code.

   Here's an example of the `define` statement used for the `gl-matrix-min.js` library, a collection of vectors, matrices, and associated linear algebra operations, which has been imported to the application's `resources/js` folder:

```
define(['resources/js/gl-matrix-min'], (glmatrix) => {
  'use strict';

  class AppModule {
    createVec3(form) {
      let myVec3 = glmatrix.vec3.create();
      glmatrix.vec3.set(myVec3, 0,0, 2.0);
      return myVec3;
    }
  }

  return AppModule;
});
```

   Note how the file is referenced simply by adding `resources/js` to the name of the JS file (you don't need the `.js` extension). Note also the alias `glmatrix`, which is used to name your import. This alias is the name you'll use to reference the objects and functions within the library.

> **Tip:**
>
> It's also possible to import a JavaScript library to your app's resources, then use **Imports** in the Settings editor to create a reference to the imported resource that you can call in your application without adding code to your JavaScript file. See Manage Custom Component, CSS, and Module Imports.

Use RequireJS to Reference Third-Party JavaScript Libraries

If you want to use third-party JavaScript libraries in your application, you can import the library and add a `requirejs` statement to your application's definition.

1. Open the `app-flow.json` file for your application.
   - In the Web Apps pane, select your application node, then click the **JSON** tab, or
   - In the Source view, locate the file for your application under `webapps`.

2. Add a `requirejs` statement to the application's definition. For example, if you've added `gl-matrix-min.js` to your application's resources under …/applications/*<your-app-id>*/resources/js/, add:

```
"requirejs": {
  "paths": {
    "gl-matrix": "resources/js/gl-matrix-min"
  }
}
```

   You can also use an expression as the value. For example, instead of `resources/js/gl-matrix-min`, enter:

```
"requirejs": {
  "paths": {
    "gl-matrix": "{{ 'resources/js/' + $initParams.resourceFolder }}"
  }
}
```

   Either way, make sure the `requirejs` entry is a sibling of the `id` or `description` entries. If a `requirejs` section already exists, simply add your entry under `paths`.

3. To load and use your library in a module, use the `define` statement to make your library a dependency for your module. In your JS file, enter, for example:

```
define(['gl-matrix'], (glmatrix) => {
          'use strict';
 ...
```

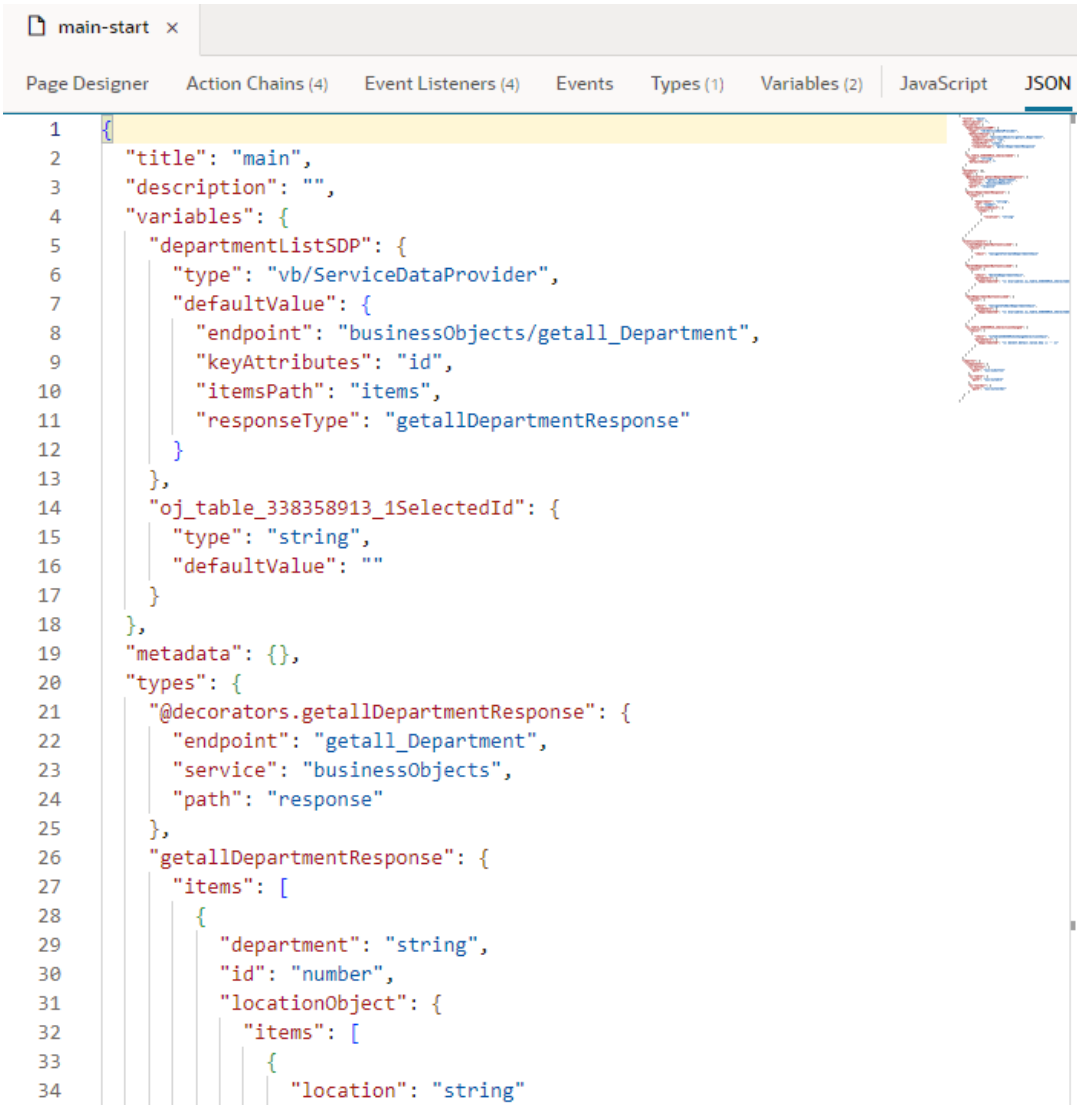
Work With JSON

When you're building an application, everything that you do in the visual editors—creating and modifying variables, types, action chains, and so on—is saved as JSON metadata. The JSON editor displays this metadata, allowing you to modify it manually if needed.

Each application, flow, and page has its own JSON file to store metadata, as does each layout and fragment. By default, an application uses `app-flow.json`, a flow uses *`flow-name`*`-flow.json`, and a page uses *`page-name`*`-page.json`. A layout uses `layout.json` and a fragment uses *`fragment-name`*`-fragment.json`.

To work with an artifact's JSON metadata:

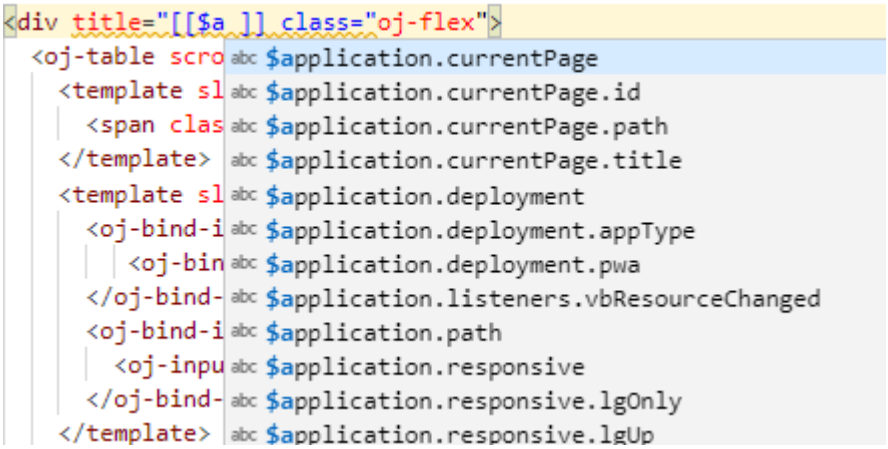
1. Select the artifact, then click the **JSON** tab. For example, here's a view of the page-level JSON editor:



2. Update the metadata as required.

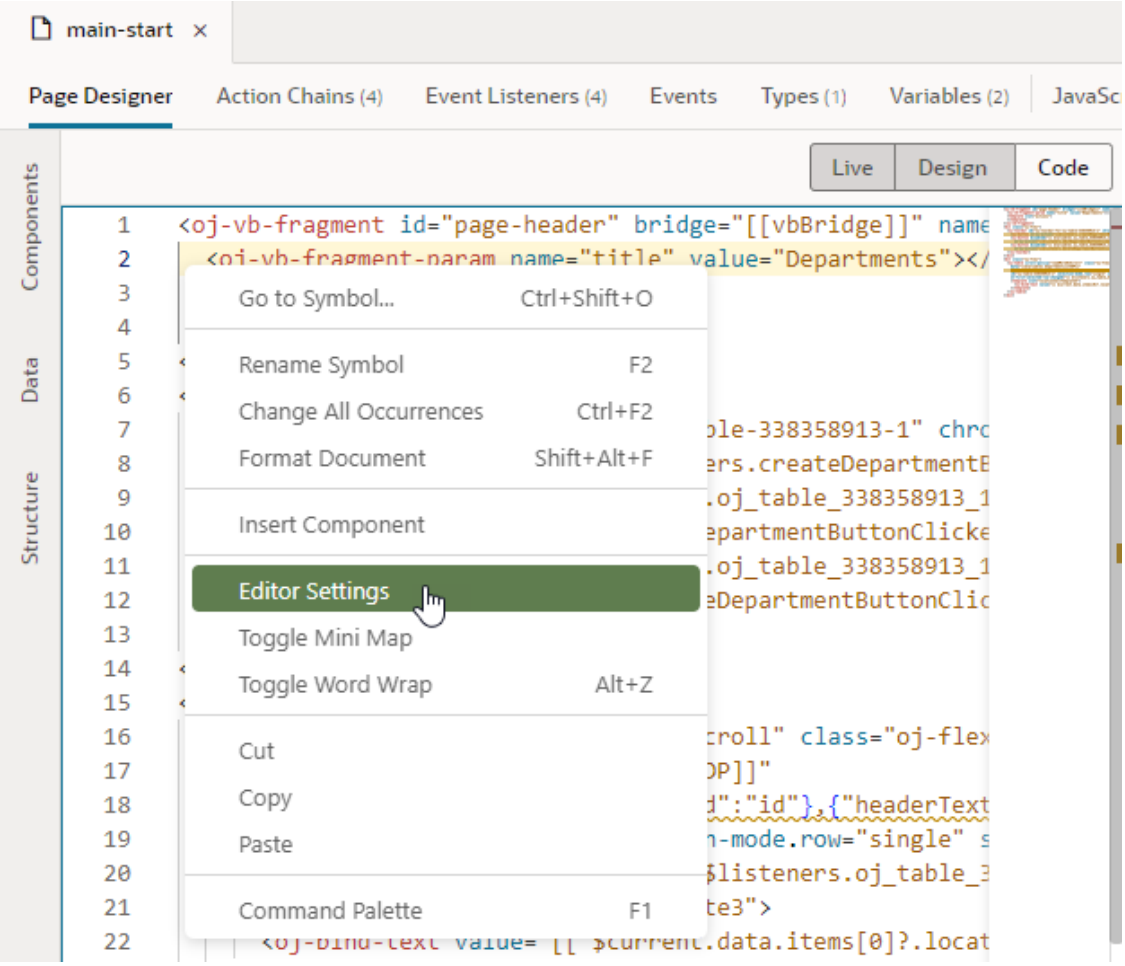
Trigger Code Insight

When working with code editors, you can invoke insights either by pressing Ctrl+Space or by entering a trigger character such as the dot (.) in the JavaScript editor. Here's how you can use insights in the different editors:

Editor	Use this to trigger insight:
HTML	• Enter the left-angle bracket (<) to trigger HTML tag insight. • Press Ctrl+Space to trigger HTML attribute name insight. Insight automatically kicks in after you enter two characters from the attribute name. • Press Ctrl+Space to trigger attribute value insight inside an HTML attribute value, double brackets ([[ ]]), or double braces ({{ }}). If the HTML is well formed, insight kicks in after you enter two characters of the expression. For example, entering `<div title="[[ $a ]]">` triggers attribute value insight for expressions after `$a`, as shown here:  • Enter any character to trigger insight in CSS files.
JavaScript	Enter any character to trigger insight. URL selector and standard imports for a module are not supported.
JSON	Enter any character to trigger insight based on the file's associated JSON schema.

Manage Code Editor Settings

To customize a code editor to your liking or to enforce consistent code formatting styles for everyone who works on an application, use the **Editor Settings** option in a code editor's context menu. Doing this brings up the `settings.json` file, which you can use to control how a code editor functions:



Use `settings.json` to control tab width, font size, and more. By default, only a handful of settings show, but you can include several more properties as listed here.

Setting	Description	Default
`editor.acceptSuggestionOnEnter`	Whether insight suggestions should be accepted on pressing the Enter key, in addition to the Tab key: • `on`: Accept a suggestion with Enter as well as Tab • `off`: Accept a suggestion only with Tab • `smart`: Accept a suggestion with Enter only when the change is textual	`on`
`editor.accessibilityPageSize`	Number of lines read out by a screen reader	None
`editor.accessibilitySupport`	Whether the editor should be optimized for use with screen readers: • `on`: Keep editor optimized for usage with a screen reader • `off`: Do not optimize editor for usage with a screen reader • `auto`: Optimize editor only when a screen reader is detected	`on`

Setting	Description	Default
editor.autoIndent	Control automatic indentation while typing: • none: Do not automatically insert indentation • advanced: Keep the current line's indentation, honor language-defined brackets, and invoke special onEnterRules defined by languages • full: Keep the current line's indentation, honor language-defined brackets, invoke special onEnterRules defined by languages, and honor indentationRules defined by languages • brackets: Keep the current line's indentation and honor language-defined brackets • keep: Keep the current line's indentation	advanced
editor.cursorBlinking	Control cursor blinking: • blink, smooth, phase, or expand: Provide various degrees of the blinking animation • solid: No blinking	blink
editor.cursorStyle	Control the appearance of the cursor: • line: A line at the current position • line-thin: A thinner version of line at the current position • block: A solid block that covers the current character • block-outline: A block that outlines the current character • underline: An underline at the current position • underline-thin: A thinner version of underline at the current position	line
editor.cursorWidth	When cursorStyle is set to line, controls the width of the line	2
editor.cursorSurroundingLines	Minimum number of lines visible above and below the cursor, starting with 0	0
editor.cursorSurroundingLinesStyle	Whether cursorSurroundingLines should be enforced: • default: Enforce cursorSurroundingLines only when cursor is changed using the mouse • all: Enforce cursorSurroundingLines always	default
editor.dragAndDrop	Enable or disable drag and drop of a selection: true or false	false
editor.emptySelectionClipboard	Whether copying without selection should copy the current line: true or false	true
editor.folding	Enable or disable code folding: true or false. The folding margin disappears when folding is disabled.	true
editor.fontFamily	Font family to use in the editor	monospace
editor.fontSize	Control text font size in pixels, starting with 10. A value under 10 may be difficult to read.	14
editor.fontWeight	Weight of the font used in the editor: normal, bold, or numbers between 1 and 1000	normal
editor.formatOnPaste	Whether pasted content should be automatically formatted: true or false	false

Setting	Description	Default
`editor.formatOnType`	Whether a line should be automatically formatted while typing: `true` or `false`	`false`
`editor.insertSpaces`	Insert spaces (instead of tabs) when the Tab key is used for indentation	`true`
`editor.letterSpacing`	Control spacing between letters, in pixels	None
`editor.lineHeight`	Control height of a line	None
`editor.matchBrackets`	Whether matching brackets should be highlighted when the cursor is at a brace: `always`, `never`, or `near`	`always`
`editor.mouseWheelScrollSensitivity`	Numbers of lines to scroll when the mouse wheel is used	`1`
`editor.mouseWheelZoom`	Whether pressing the Control key and the mouse wheel should change font size: `true` or `false`	`false`
`editor.multiCursorModifier`	Modifier to be used with a mouse click to create multiple cursors: • `alt`: Maps to the Alt key on Windows and to the Option key on Mac • `ctrlCmd`: Maps to the Control key on Windows and the Command key on Mac	`alt`
`editor.occurrencesHighlight`	Whether to track cursor and highlight other occurrences of the current word or variable: `true` or `false`	`true`
`editor.renderLineHighlight`	Controls how the current line is highlighted: • `all`: Highlight the current line as well as the gutter • `line`: Only highlight the current line • `gutter`: Only highlight the current line's gutter • `none`: Do not highlight the current line	`all`
`editor.renderWhitespace`	Control how the editor should render whitespace characters: • `none`: Do not render whitespace characters • `boundary`: Render whitespace characters except for single spaces between words • `selection`: Render whitespace characters only on selected text • `trailing`: Render only trailing whitespace characters • `all`: Render all whitespace characters	`selection`
`editor.selectOnLineNumbers`	Whether the line should be selected if the line number is clicked: `true` or `false`	`true`
`editor.showFoldingControls`	Control when folding controls show: • `always`: Always show the folding controls • `mouseover`: Show the folding controls only when the mouse is over the gutter	`mouseover`
`editor.showUnused`	Whether unused variables should be faded out: `true` or `false`	None
`editor.suggestFontSize`	Font size for insight suggestions	None
`editor.suggestLineHeight`	Line height for insight suggestions	None
`editor.suggestOnTriggerCharacters`	Whether insight should be triggered by special characters: `true` or `false`	`true`

Setting	Description	Default
`editor.suggestSelection`	Controls how suggestion history works: • `first`: Always select the first suggestion • `recentlyUsed`: Select recent suggestions • `recentlyUsedByPrefix`: Select suggestions based on previous prefixes that have completed those suggestions	None
`editor.tabCompletion`	Enable or disable completion by pressing the Tab key: • `on`: Insert the best matching suggestion when pressing Tab • `off`: Disable Tab completion • `onlySnippets`: Tab complete snippets when their prefixes match	None
`editor.tabSize`	Number of spaces a tab is equal to, starting with 1	`2`
`editor.theme`	Changes the editor's color theme: `redwood`, `vs`, `vs-dark`, or `hc-black`	`redwood`
`editor.wordWrap`	Controls word wrap in the editor: • `on`: Wrap lines at the viewport width • `off`: Do not wrap lines • `wordWrapColumn`: Wrap lines at `wordWrapColumn` • `bounded`: Wrap lines at the minimum of viewport and `wordWrapColumn`	`off`
`editor.wordWrapColumn`	Number of columns to use when `wordWrap` is set to `wordWrapColumn`, starting with 20	None
`editor.wrappingIndent`	Controls how a wrapped line is rendered: • `none`: No indentation. Wrapped lines begin at column 1 • `same`: Wrapped lines use the same indentation as the parent • `indent`: Wrapped lines get +1 indentation toward the parent • `deepIndent`: Wrapped lines get +2 indentation toward the parent	`same`
`editor.minimap.enabled`	Show or hide the code minimap.	`true`
`editor.minimap.size`	Control the size of the minimap: • `proportional`: The minimap has the same size as the editor contents (and might scroll) • `fill`: The minimap will stretch or shrink as necessary to fill the height of the editor (no scrolling) • `fit`: The minimap will shrink as necessary to never be larger than the editor (no scrolling)	`fit`
`editor.minimap.side`	Where to render the minimap: `right` or `left`	`right`
`editor.minimap.renderCharacters`	Render characters on a line as opposed to color blocks: `true` or `false`	`true`
`editor.minimap.scale`	Scale for rendering the minimap, starting with 1	`1`

15

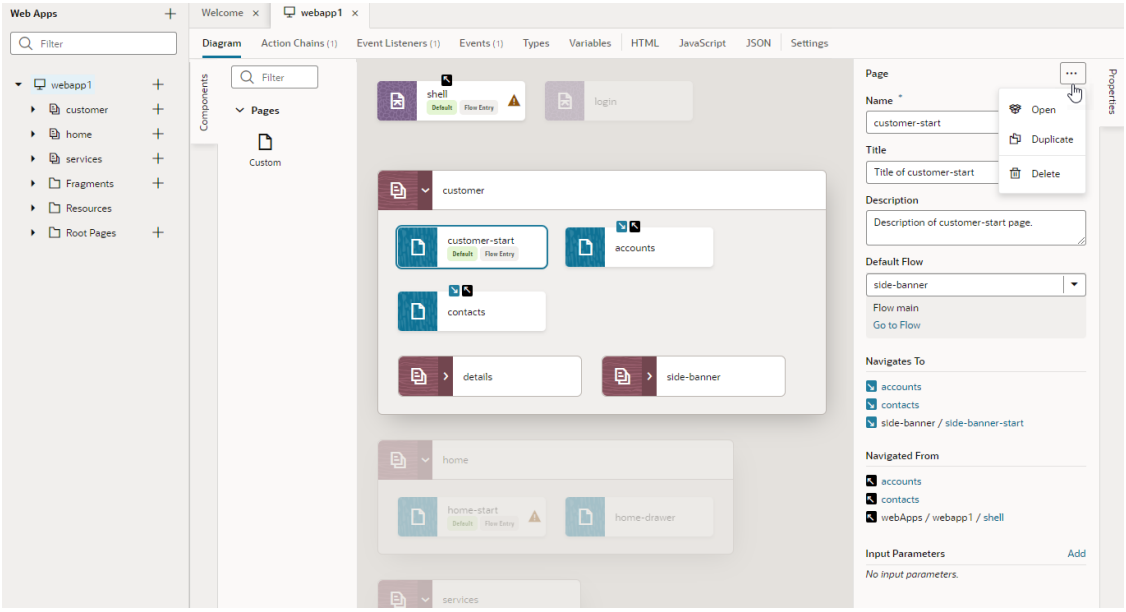
Work With the Diagram View

As your application takes shape in the Designer, you can use the Diagram view for a visual representation of your application's structure.

The Diagram view, shown only for application and flow artifacts, displays an application's root pages, flows, and pages within flows. It's a handy tool that lets you view default pages, navigation flows, even audit status at a glance. You can also use this view to update your application in context; for example, you can change the default root page or the default flow and see how your updates change your application's structure.

When you open an application artifact, the Diagram view displays the application's root page as well as a hierarchical view of the artifact's flows and sub-flows. When you open a flow artifact, the Diagram view displays the pages contained in the flow as well as their navigational relationships. A Properties pane displays by default, showing additional information about the selected artifact. There's also a Components palette that you can use to add pages (and actions for a flow artifact).

Here's an example of what you might see when you open the Diagram tab at the application level:



You can expand or collapse a flow to show or hide its pages (and optionally, sub-flows). Click a page to view its navigational relationships in the diagram as well as in the Properties pane. For example, clicking the `customer-start` page shows navigation icons ( ) on the `accounts` and `contacts` page tiles, indicating that you can navigate from `customer-start` to those pages and back. When navigation is one way, meaning you can go from one page to another but not navigate back, you'll only see the icon, as shown on the `side-banner-start` page. You'll see similar navigation details in the Properties pane under Navigates to and Navigated From when the page is selected. Notice how flows or pages that don't have any relationship with the selected page fade into the background.

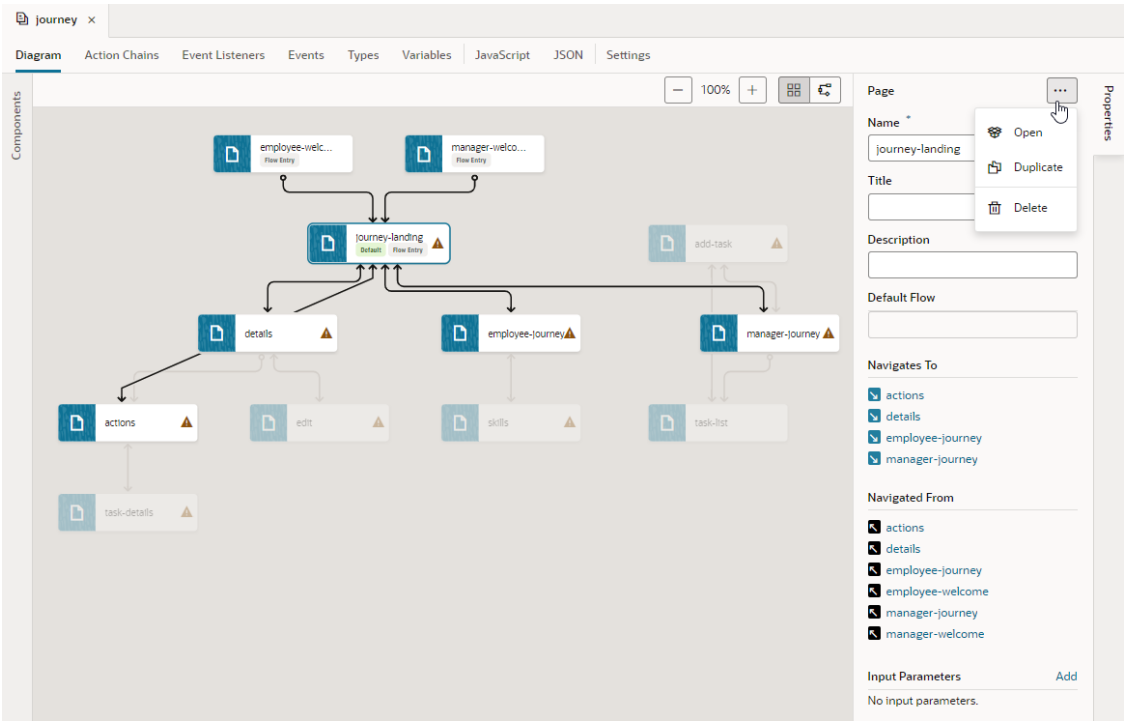
It's possible to make changes to your application from the Properties pane. You can change page titles and descriptions as well as update the app's default root page, the default flow, and the default page in a flow. You can even open, duplicate, and delete selected items. Alternatively, you can double-click an artifact (a root page, a flow, or a page) to open up the artifact's editor and make changes as required.

The Diagram view also flags pages with audit issues (or). These issues also show at the flow level, a useful indicator when the flow is collapsed that audit issues exist in the flow's pages.

View a Flow's Navigation in Diagram View

When you open a flow artifact in the Diagram tab, the Flow Diagram view ( ) displays all pages within the flow and their navigational relationships. You can use this high-level view to focus on principal navigation between pages in the flow.

Here's an example of what you might see in the Flow Diagram view:

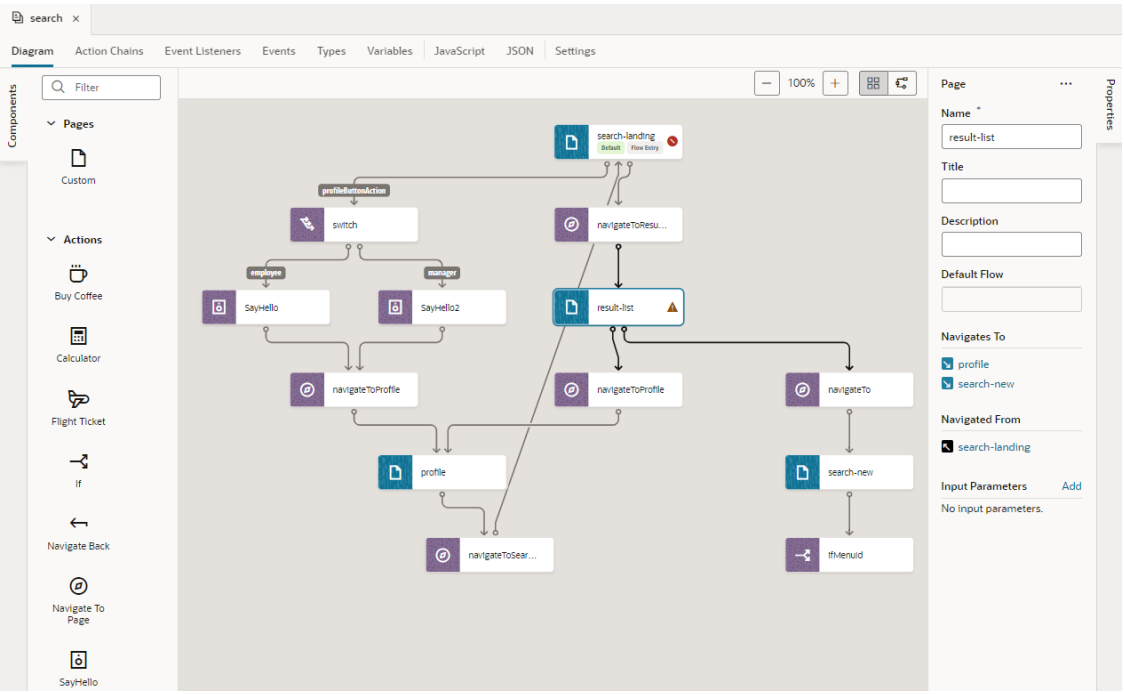


Notice how the default page (`journey-landing`) has a Default badge, indicating it as the flow's default page. You can click a page tile to highlight all navigation links. Links flow in the direction of the navigation from source to target page; pages that navigate to each other have arrows at both ends. Take note of how unrelated pages fade into the background to give you a better idea of how the flow is constructed. Navigational details also show in a selected page's Properties pane; you can also add input parameters, duplicate and delete a page as well as open it in the Page Designer.

Add Pages and Action Chains to a Flow in Diagram View

Use the Flow Diagram view to add pages and actions chains to a flow while keeping the entire flow in context. Creating a flow in the Flow Diagram, instead of the page editors, is convenient when you want to build workflows without needing to code. It can also help you visually navigate complex flows, even reuse sub-flows.

You can build a flow by adding pages and creating page-level action chains, just by dragging items from the Components palette and dropping them onto a tile in the diagram. Here's an example of a `search` workflow that shows all pages and their corresponding actions created via the Flow Diagram:



You can click a page or action tile—both are distinctly color-coded for easier identification—to highlight all the connecting links. Take note of how links flow in the direction of the navigation from source to target page; pages that navigate to each other will have arrows at both ends. You can use a selected tile's Properties pane to view additional information and do some other functions.

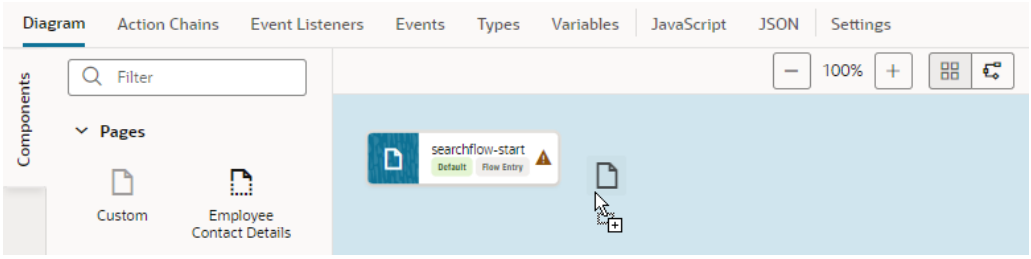
> **Note:**
>
> Opening a flow that was built using the page editors only shows navigation by default in the Flow Diagram. But if you were to build your flow from scratch in the Flow Diagram, all pages and associated action chains will also show. To change this setting, see Show or Hide an Action Chain in the Flow Diagram.

You can also duplicate your workflow by clicking **Duplicate** in the flow's right-click menu in the app's tree view. Duplicating a flow will copy all its content, including pages, chains, and sub-flows, and can serve as a starting point for a new workflow.

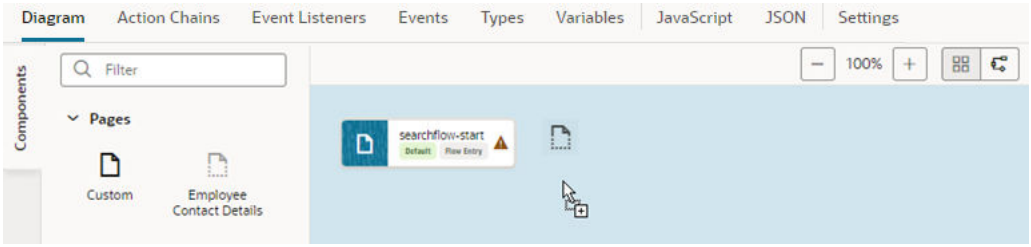
Add a Page in the Flow Diagram

To add a page in a Flow Diagram, you drag and drop a page from the Components palette onto the diagram. Adding a page to a flow is similar in the Flow Diagram view ( ) as well as the Grid view ( ).

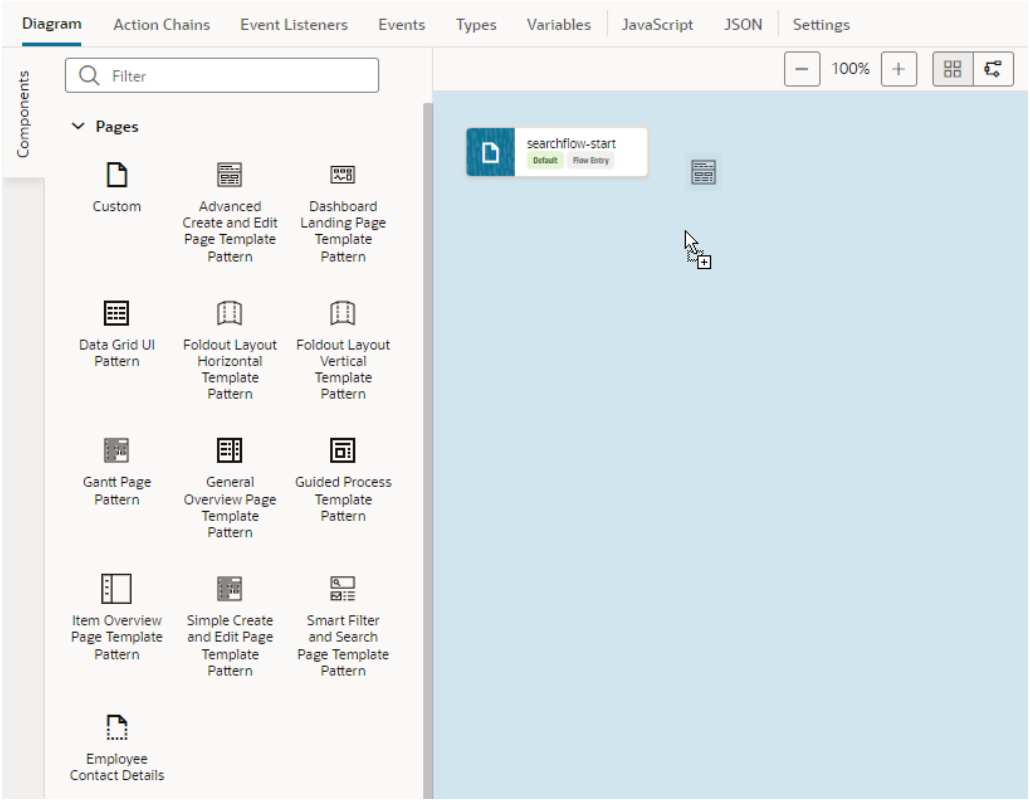
1. Open your application in the Navigator, then click the Create Flow icon ( ) next to the application node to create a new flow.
2. In the Diagram view (Flow or Grid), you can create an empty page, a page with a pattern, or one with an existing fragment. To create a page with a fragment, the fragment must be tagged with the `page` metadata tag in its **Used For** setting (either from its Properties pane or Settings editor). Without the `page` tag, the fragment won't surface in the Components palette.
   - To create a page without any content, drag **Custom** under Pages in the Components palette and drop it onto the diagram.



   - To create a page containing a specific fragment, drag the fragment under Pages in the Components palette and drop it onto the diagram.



   - To create a page containing a page pattern, drag the pattern under Pages in the Components palette and drop it onto the diagram.



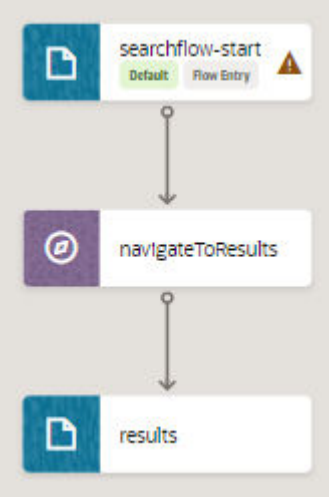
When your application includes `page` fragments as well as patterns, both will be available to you in the Components palette.

3. In the Create Page dialog, give the page a name, then click **Create**.

   A new page tile appears in the diagram (with its properties displayed in the Properties pane). The newly created page's icon in the diagram will match the fragment or pattern icon used to create the page.

Tip:

Want to quickly create a page that automatically navigates to an existing page? You can, but only in the default Flow Diagram view. Simply drag a **Custom** item from the Components palette and drop it directly onto an existing page, enter a name for the new page when prompted, and click **Create**. A new page is created and a `navigateToPage` action chain that navigates from the existing page to the new page is added to the existing page, as shown here:



After you've created a page, select the page tile to view and update its properties in the Properties pane. You can manage the page using the options in the Properties pane's Menu (***):

- To open a page in the Page Designer, where you design it as needed, click **Open**. You can also double-click the page tile to open it in the Page Designer.
- To duplicate a page, click **Duplicate**. Duplicating a page copies all the page's action chains.
- To delete a page, click **Delete**.

Create an Action Chain in the Flow Diagram

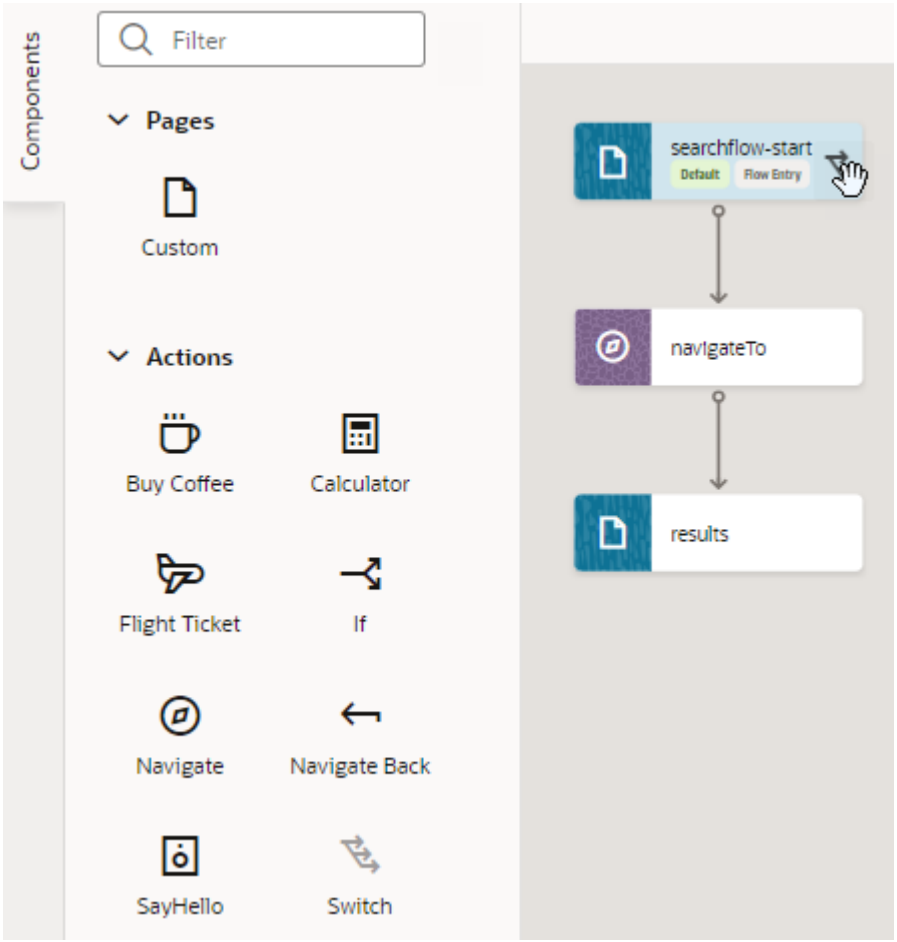
To create an action chain for a page in the Flow Diagram, you drag and drop an action from the Components palette onto a page. You can add built-in actions (such as Navigate, If, and Switch) as well as custom actions to create a page-level action chain.

Note:

If you want to use custom actions in a Flow Diagram, the custom action's `showInDiagram` property must be enabled to surface the action in the Flow Diagram's Actions palette. See Define the Custom Action's Properties.

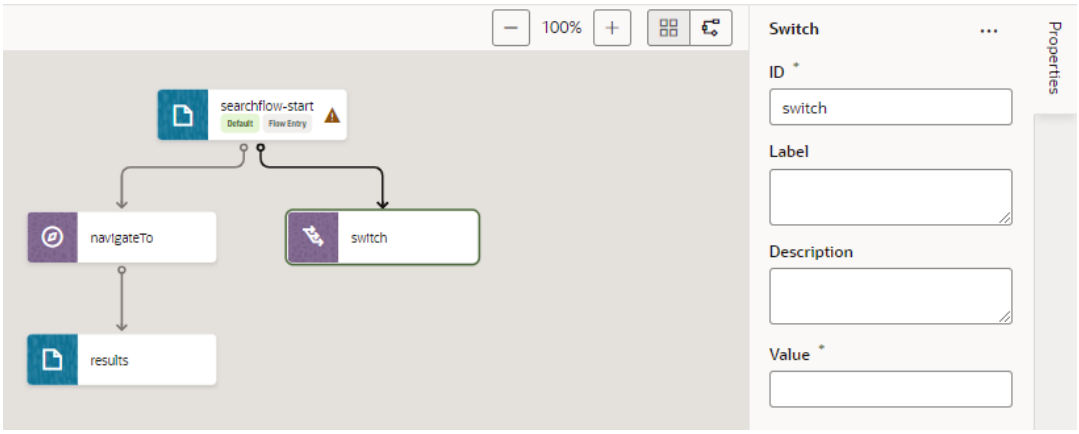
To create an action chain for a page in the Flow Diagram:

1. Select an application's flow to open it the Flow Diagram view ( ).
2. Drag and drop an action (built-in or custom) under Actions in the Components palette and drop it onto a page in the diagram.



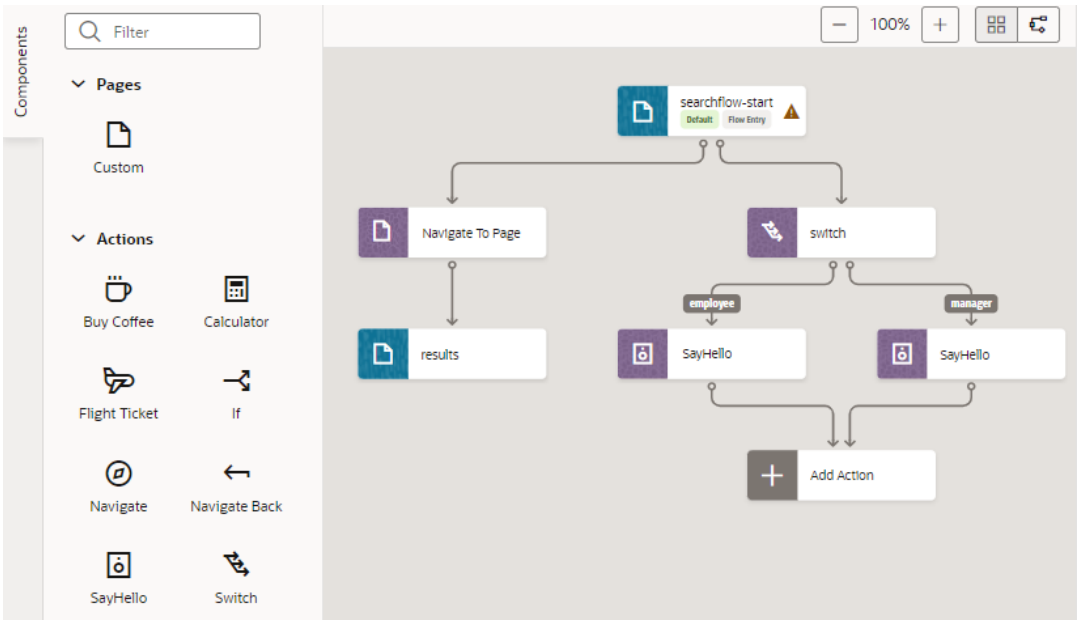
3. When prompted to select an event listener, select **Bind an event listener later**. If you want to bind an action chain to an existing event listener, you'll need to first surface it in the Flow Diagram. See Bind an Action Chain in the Flow Diagram to an Existing Event Listener.

   A new action chain is created with your action as the root. Use the action's Properties pane to suitably configure your action's properties. The properties that display are typical for built-in actions. For example, here's what you'll see for a Switch action:

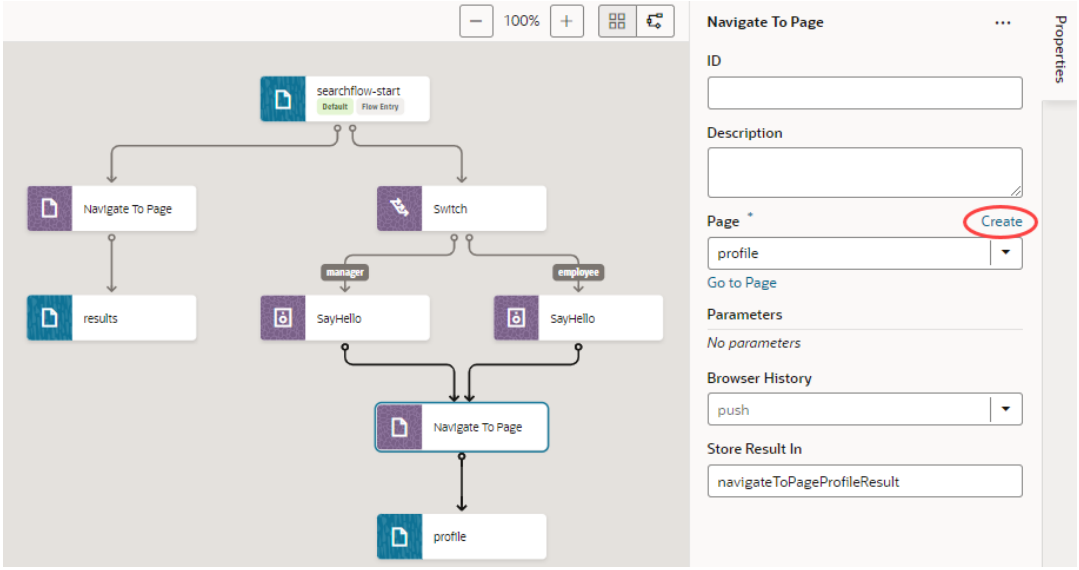


4. If your action involves additional steps, drag and drop additional actions as needed.

   Say you want to call a custom Say Hello action in the Switch action to display an employee-specific message, you would drag the Say Hello action onto the switch action to add multiple cases:



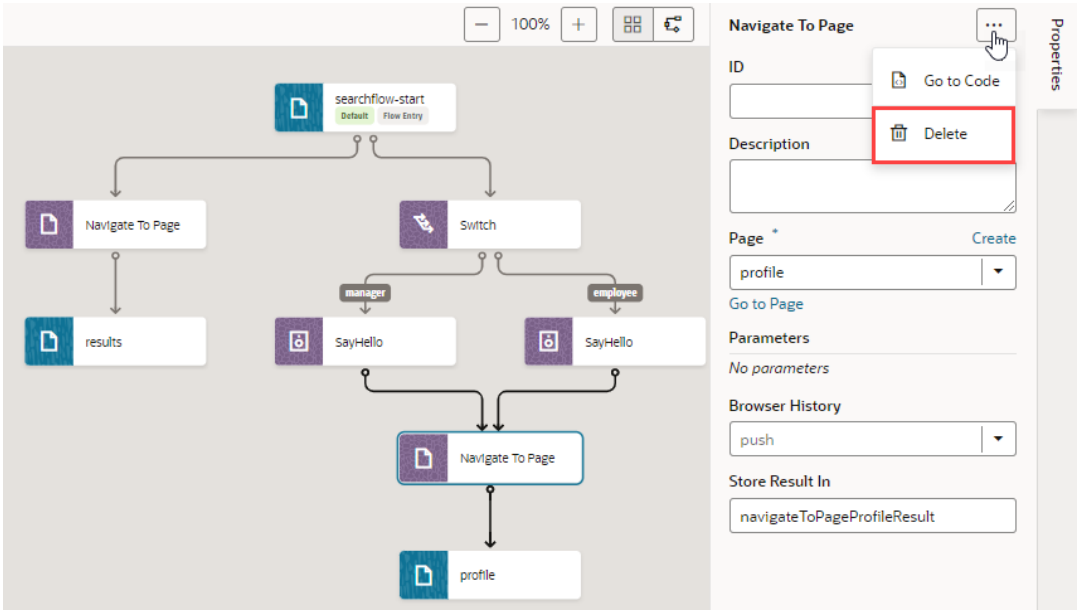
   The cases you add for the Switch action show as labels, as do outcomes of decision nodes for an If action. The first action you drop onto an If action is considered the `true` outcome and the second is considered the `false` outcome. When the If action has both true and false outcomes or a Switch action has more than one outcome, a placeholder `Add Action` node appears (as shown in the image above), so you can specify the action after the branches join (`Navigate To Page` as shown here):



5. To add more actions to the action chain, drag and drop an action onto an existing action in the diagram.

   You can drag an action over any other action node anywhere in the action chain, as long as the node is highlighted in green to indicate that more actions are allowed. (You likely won't extend a chain with a Navigate action as the action navigates you away from the page and subsequent actions won't take effect.)

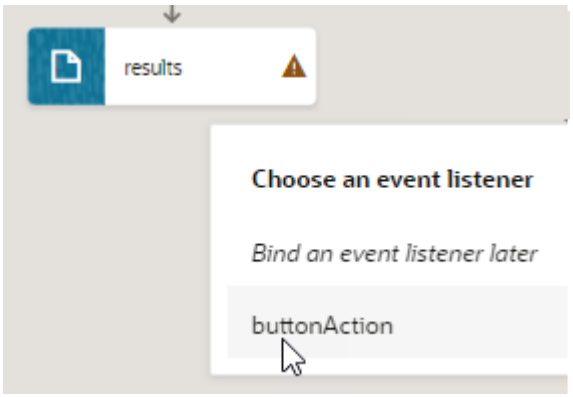
   If you want to delete an action, click Menu ••• in the action's properties and click **Delete**.



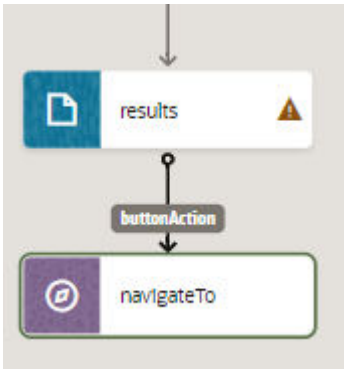
Bind an Action Chain in the Flow Diagram to an Existing Event Listener

To associate a new action chain in the Flow Diagram with an existing event listener, you'll need to surface the event listener in the Flow Diagram. Event listeners call action chains in response to component or lifecycle events.

1. Enable the event listener you want to use with an action chain to surface in the Flow Diagram.
   a. Select the page that contains your event listener, then click the **Event Listeners** tab.
   b. Select an existing event listener, or create a new one. See Create Event Listeners for Events.
   c. In the event listener's Properties pane, select **On** under **Show in Flow Diagram**. This option shows only for page-level event listeners.
2. Bind an action chain in the Flow Diagram to the event listener.
   a. Select the flow containing the page-level event listener and open it the Flow Diagram view ( ).
   b. Drag and drop an action from the Components palette onto the page containing the event listener.
   c. When prompted, select the event listener:



The new action chain that's created will be added to the event listener. The link label also shows the listener's name, as shown here:

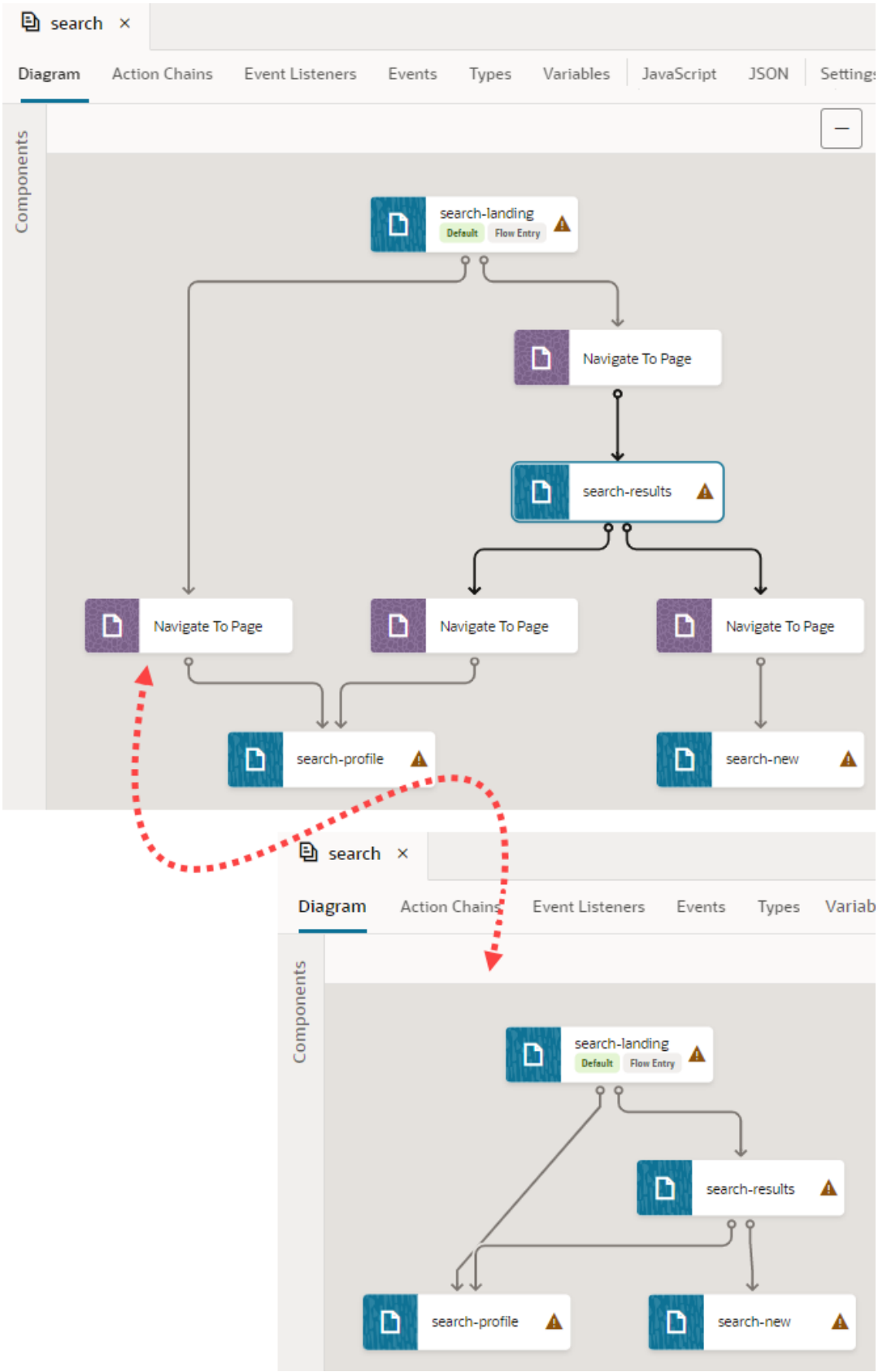


Show or Hide an Action Chain in the Flow Diagram

By default, action chains created via a Flow Diagram show all action nodes and the event listeners that they are bound to. If you want a simpler view, you can change this setting so that a flow shows only its pages and their navigational relationships.

To show or hide an action chain in the Flow Diagram:

1. Select an application's flow to open it the Flow Diagram view ( ).
2. Double-click the page containing the action chain you want to show or hide.
3. When the page opens in the Page Designer, switch to the **Action Chains** tab and select an action chain to open it in the Action Chains editor.
4. In the action chain's Properties pane, look for **Show in Flow Diagram**:
   - Select **Navigation Only** to show only the navigational relationships for the page associated with the action chain.
   - Select **Full** to show all action chain details, including actions and associated event listeners.
5. Return to the Flow Diagram. Here's an example showing the two views:


search
Diagram
Action Chains
Event Listeners
Events
Types
Variables
JavaScript
JSON
Components
search-landing
Default
Flow Entry
Navigate To Page
search-results
Navigate To Page
Navigate To Page
Navigate To Page
search-profile
search-new
search
Diagram
Action Chains
Event Listeners
Events
Types
Components
search-landing
Default
Flow Entry
search-results
search-profile
search-new

On the left is a flow's Full view, showing action chains (and their event listeners) configured for a set of pages. On the right is the Navigation Only view for the same set of pages, where only pages and their navigation show.

16

Work With Fragments

As you design a web application, some pages might quickly become large and unwieldy. One way to simplify the process of building and maintaining complex pages is to use fragments.

Large, complex pages broken down into several smaller fragments are easier to maintain. For example, when a page uses a foldout layout with multiple panels or includes multiple tabs, you might find it easier to keep each panel or tab's content in a fragment. This way, you modularize your app's logic and can maintain each panel or tab separately.

This might sound similar to what you'd accomplish with flows, but flows and fragments are fundamentally different. Flows group pages by business function and allow navigation between pages within or across flows. Fragments, on the other hand, break up a page into separate sections for easier organization and code management, and can even serve as entire page templates. Unlike flows, they can be used in multiple pages, even multiple times in the same page—which brings us to reusability, the most compelling reason to use fragments.

Because a fragment encapsulates parts of a page in its own HTML, JSON, and JavaScript files, it can be shared across pages, flows, even other fragments in your application. For example, suppose different sections of several pages use the same form, you can create fragments containing the form, then reuse those fragments in several other pages.

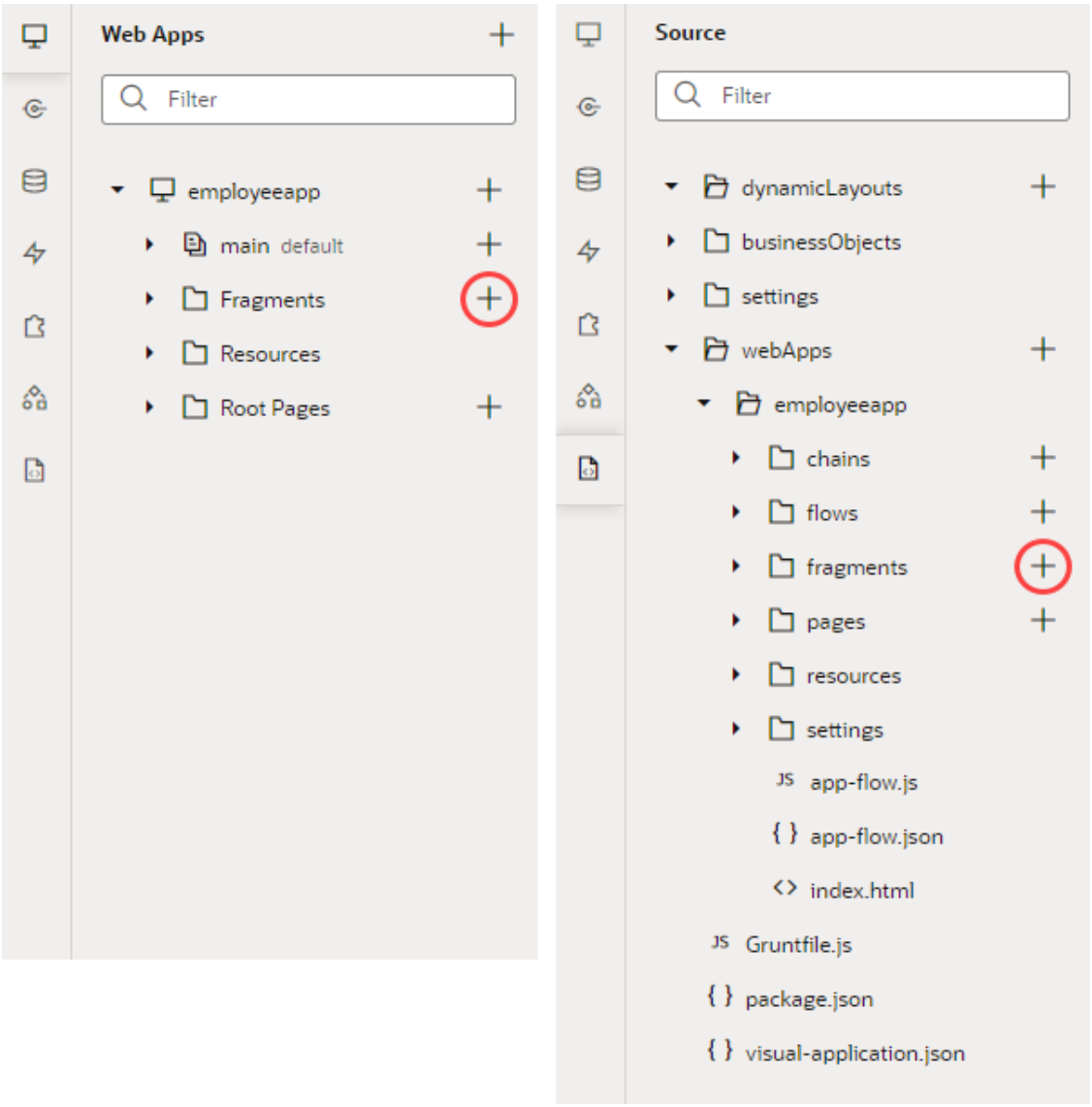
Besides the benefits of reuse, fragments provide performance gains. Typically, all resources used in a page (components, modules, and so on) load when the page renders. But sometimes you don't need all components and the related model, especially those triggered by UI events or hidden behind other UI components, to load right away. For example, you don't need components in a panel's edit version to show until the user clicks the edit icon. If you define the edit panel in a fragment, you could delay rendering until you actually need to show the fragment to improve the time it takes for the page to render initially.

Here's a video that provides a high-level look at fragments and walks you through a sample: Fragments in Visual Builder.

Create and Add a Fragment to a Page

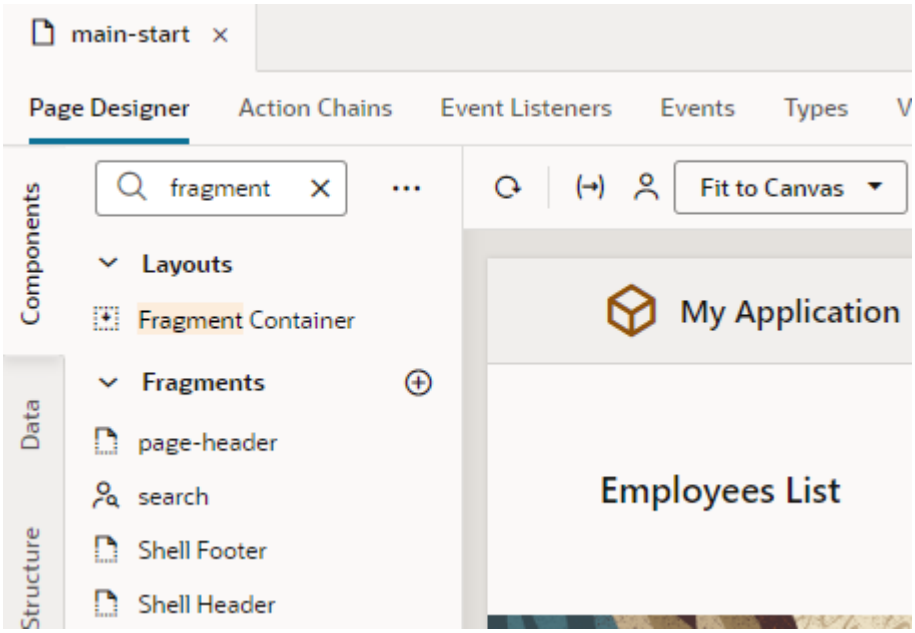
Create one or more fragments to define sections of a page. Say, your page has separate tabs for employees and managers; you can create two fragments, one for either tab's content. Deciding how many fragments to create depends on your application, the degree to which you wish to reuse portions of a page between multiple pages, and the extent to which you want to simplify complex pages.

There are many ways to create fragments: you can create one using the Create Fragment icon (+) next to the Fragments node in your application's Web Apps view or in Source view as shown here:



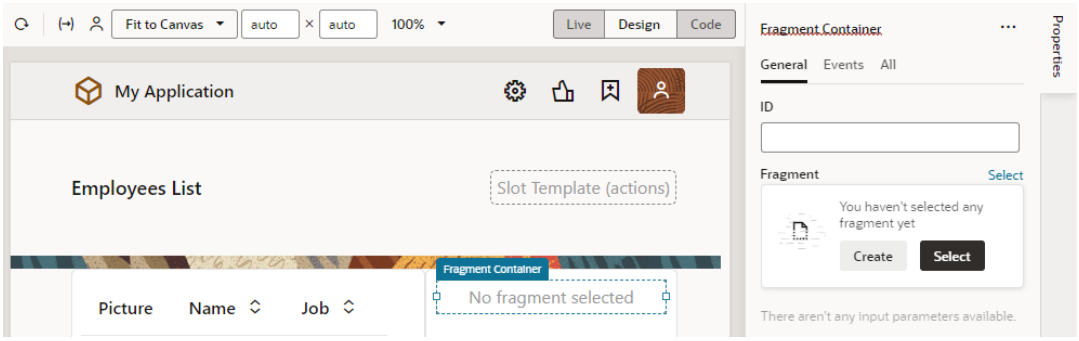
Alternatively, create a fragment when designing a page by clicking ⊕ next to **Fragments** in the Components palette. You can also start with a fragment container on a page and add a fragment to it—which is what we'll do here:

1. Open your web application in the Navigator, then open the page where you want to use fragments. Fragments are most commonly embedded in pages, other fragments, form and field templates, dynamic containers, and list item and foldout panel components.
2. Drag a Fragment Container from the Components palette onto the canvas and drop it where you want a fragment to display. It's easiest to filter for the components you want to use.

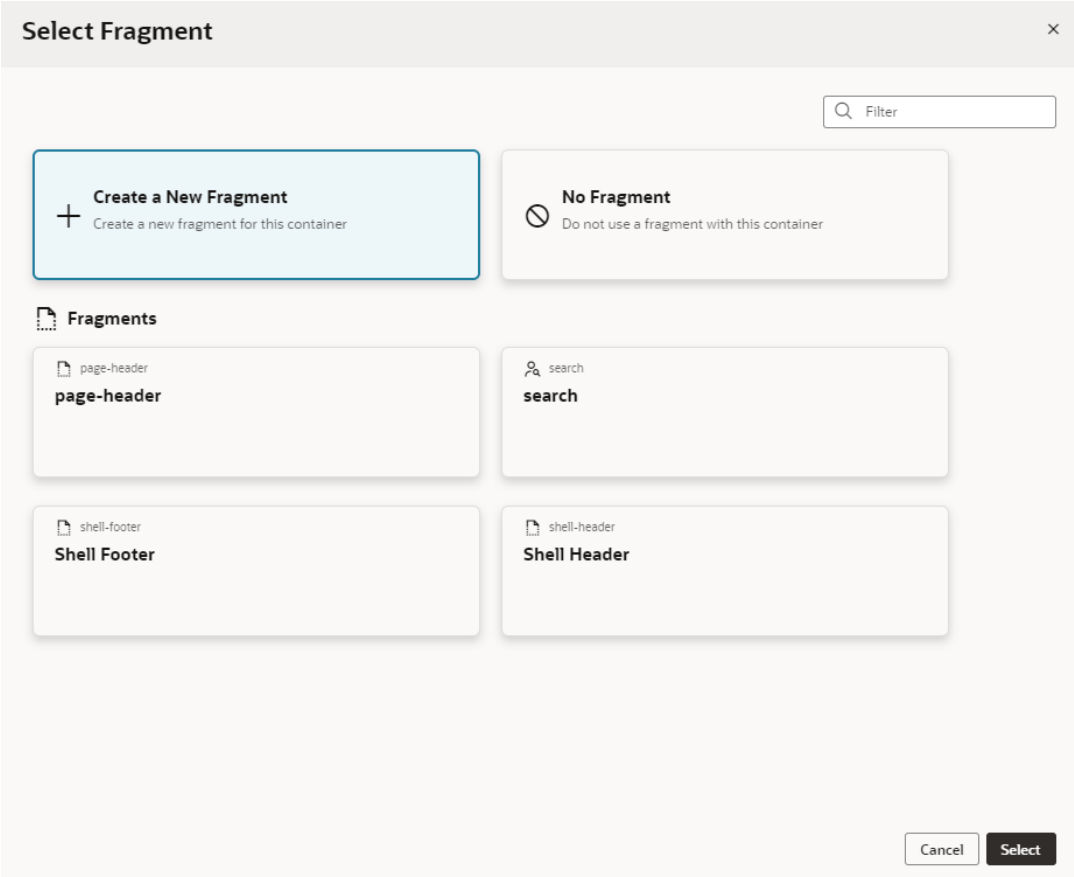


> **Tip:**
>
> Notice how the `search` fragment shows up in addition to the Fragment Container when you enter `fragment` in the Filter field? That's because by default all existing fragments—including those such as `Shell Header` and `Shell Footer` created automatically by web app templates—become available for use in your application's pages. You can simply drag and drop these fragments onto the canvas if you wanted to use them in a page. If you added a Fragment Container to the page (as we've done here), you can select these fragments from the Fragment Container's properties.

3. In the General tab of the container's Properties pane, click **Select** to select an existing fragment or **Create** to create a new one. For demonstration purposes, we'll create a new fragment.

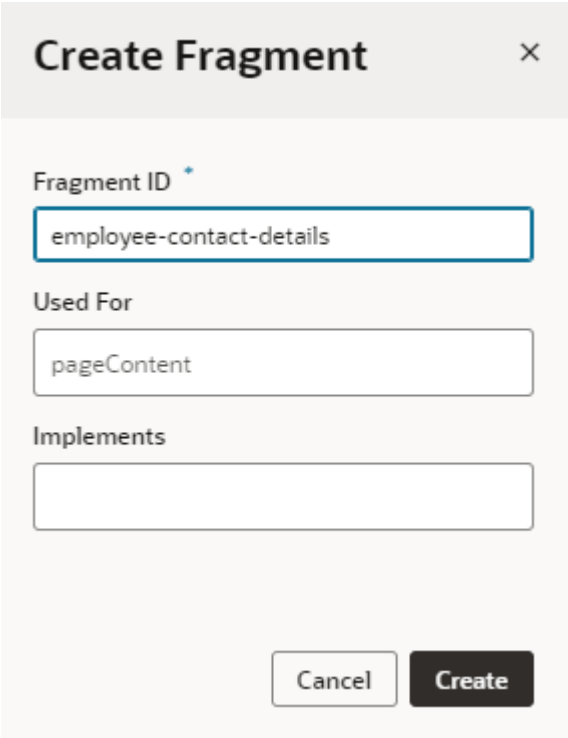


If you click **Select** and find that the available fragments don't meet your needs, you can create a new fragment even from the Select Fragment dialog, as shown here:



Click **Select**.

4. In the Create Fragment dialog, enter a name for the fragment in the **Fragment ID** field.



a. In the **Used For** field, select where you want the fragment to surface as you drop it on pages or page components.

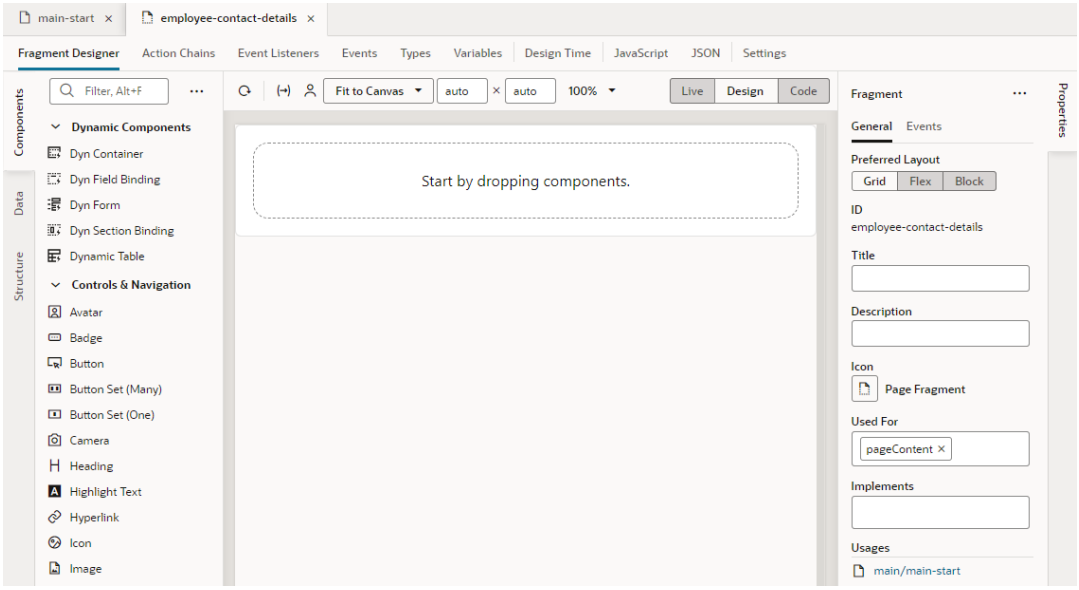
   By default, every fragment is tagged as `pageContent`, surfacing it under Fragments in the Components palette as generic content available for use in any page. But using the appropriate metadata tag displays the fragment only where it is best used. For example, a fragment tagged as `formTemplate` would be available to you only when you're looking to drop a fragment in a dynamic form template.

   You can always change where the fragment surfaces by editing the **Used For** setting in the fragment's Properties pane or its Settings editor.

b. In the **Implements** field, select tags that suggest the fragment as preferred content for particular components. If you're not sure, leave it blank. You can add it later in the fragment's Properties pane or its Settings editor.

c. Click **Create**.

A new empty fragment opens in the Fragment Designer:



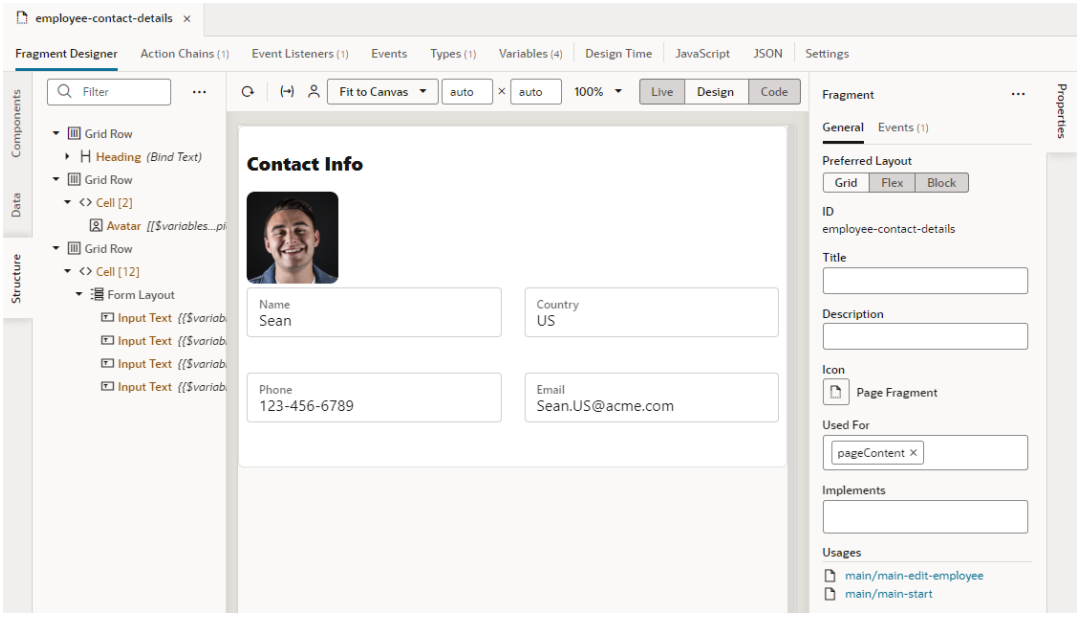
> **Tip:**
>
> To view where a fragment is consumed, look under **Usages** in the fragment's Properties pane. In our example here, the `employee-contact-details` fragment is being used by the `main-start` page in the `main` flow. Clicking the `main/main-start` link will open the page in the Page Designer.

5. Now design your fragment in the Fragment Designer.

   The Fragment Designer is similar to the Page Designer, except that it builds a fragment instead of a page. You can add standard as well as dynamic components (including dynamic containers) to your fragment, then use other editor tabs to bind components to events, action chains, variables, and functions, much like what you'd do when developing a page. You can also define types, including those from code that can be associated to an InstanceFactory variable.

Keep in mind though that a fragment is a self-contained piece of UI that's unaware of its parent container's context. So what you see on the canvas as well as in the Structure and Code views are the contents of this particular fragment. Other editor tabs allow you to edit artifacts within the scope of this fragment. So a fragment cannot call actions on its parent container, but it can fire custom events that the parent can handle. You can't also navigate to a fragment, only to a page.

Here's an example of a fragment set up to show an employee's contact information:



(For steps on how to design this sample fragment and wire up the necessary parameters, see Sample Scenario: Create a Fragment and Pass Values.)

6. Optional: Return to your page to create more fragments and add them to the page. You can add as many fragments as you need to a page, even add fragments to other fragments.

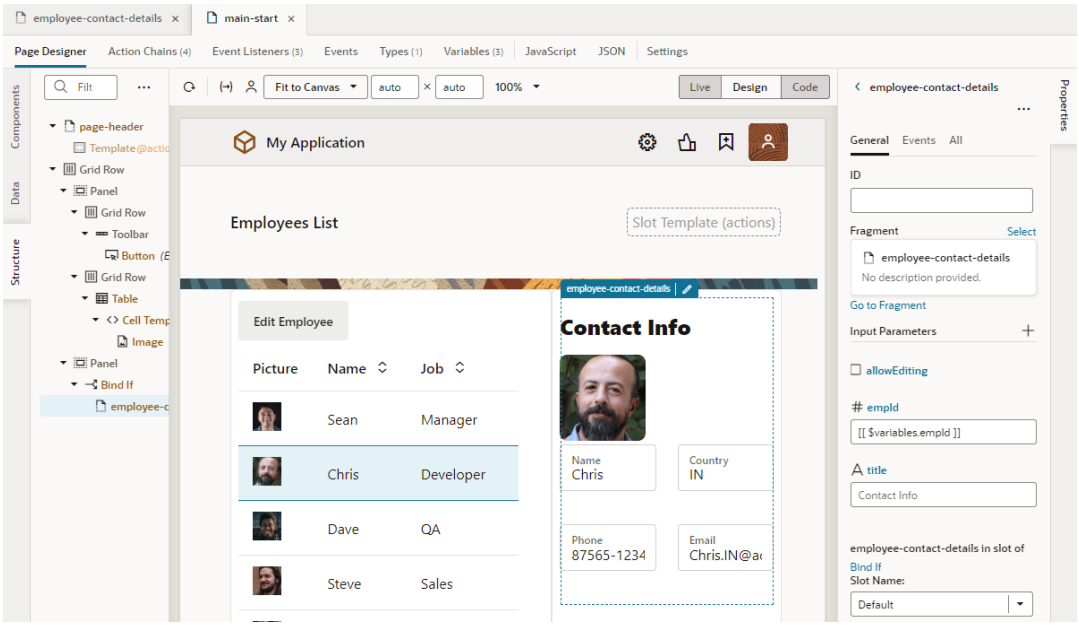
   a. Click ⊕ next to **Fragments** in the Components palette and create a fragment.

   b. Drag and drop it onto the canvas to add it to the page.

   Alternatively, repeat steps 2 to 5 to create and add fragments starting with a fragment container on a page.

> **Tip:**
>
> It's also possible to create a page starting with the contents of an existing fragment, essentially using the fragment as a page template.

After a fragment is added to a page, its content will automatically render on the page that consumes it, as shown here (this includes any changes you make on the fragment as well):



You can then use the fragment's Properties pane to view fragment variables that have been enabled as input parameters, even create them on the fragment container. You can choose to override the parameter's default value with an alternate value for the page, or use a page variable to provide initial values for a fragment's input parameter. (Hover over each input parameter to view its type and description, if one was provided.)

You can also configure the fragment's container to react to events raised by the fragment.

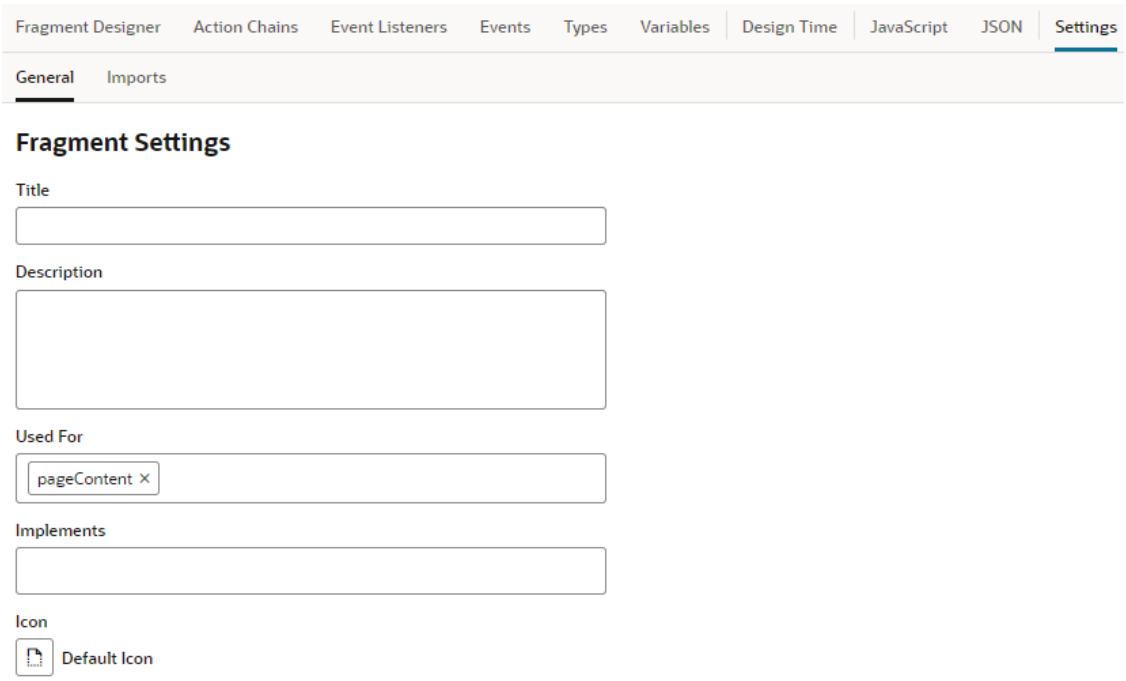
Once a fragment is added to a page, you can change it if you want. You can replace it with another fragment, even remove it completely from the page. To do this:

1. Open the page that uses the fragment you want to change, select the fragment on the page, then in the fragment's Properties pane, click **Select** next to Fragment.
2. Make your choice in the Select Fragment dialog:
   - To remove an existing fragment on the page, select **No Fragment**.
   - To replace the existing fragment with another, select the fragment you want to use under Fragments.
3. Click **Select**.

Manage Fragment Settings

Every fragment includes a Settings editor, which you use to specify some general settings as well as imported resources such as custom components, CSS files, and modules.

To configure settings for a fragment, open the fragment, then click **Settings** to open the Settings editor:



You can also find these settings on the fragment's Properties pane in the Fragment Designer. Here's how you can use the different settings:

Setting	Description
General tab	Contains general fragment settings:
Title	Fragment title that replaces the ID wherever the fragment appears. For example, if you created a fragment with the ID `myfragment`, then set `My Fragment` as the title, the `My Fragment` title will show instead of the ID wherever the fragment displays (in the Components palette, Structure view, and on the canvas).
Description	Fragment description that displays as a tooltip when you hover your cursor over the fragment's help icon in the Components palette.

Setting	Description
Used For	Tags that best describe where the fragment should be used. The value you select from the drop-down list will be used to surface the fragment in the right context and filter it out where it isn't suitable. For example, if you choose `formTemplate`, the fragment will be available only when the user is looking to use a fragment in a dynamic form template. If you choose `page`, the fragment will be available in the Flow Diagram's Components palette as well as in the Create Page dialog whenever you create a page. You can choose more than one tag to surface the fragment in multiple locations. By default, all fragments are tagged as `pageContent`, meaning they become available only in the Page Designer's Components palette (under the Fragments category). **Note:** If you tag a fragment as `formTemplate` or `fieldTemplate`, you can indicate how fragment metadata, specifically data-binding expressions in fragment input parameters, must be generated depending on where the fragment is embedded. See Set the Binding Type for Variables in Dynamic Components.
Implements	Tags that suggest the fragment as preferred content for particular components. For example, if you choose `FoldoutPanelElement`, users working with a foldout layout will see this fragment suggested as content that can be added as a foldout section.
Icon	Default icon associated with the fragment that will display wherever this fragment is used (for example, in the Components palette). Click to open the Icon Gallery, then make your selection.
Imports tab	Contains settings to manage resources such as custom CSS files, modules, and components imported at the fragment level, allowing you to create declarative references in the fragment to those resources. See Manage Custom Component, CSS, and Module Imports.

Reuse a Fragment

Because a fragment is essentially a reusable piece of UI, you can use it in a page as well as in multiple pages, even other fragments.

Say you define a fragment to show an employee's contact information. You can pull in the fragment in multiple pages, where ever you want an employee's contact details to show. For example, you can use the fragment in a page that displays an employee's contact details as well as in another page where the contact details can be edited.

You can also use a fragment multiple times in the same page, typically when you provide different sets of input parameters to the same fragment. It's also possible to create pages starting with the contents of an existing fragment, essentially using the fragment as a page template.

Because fragments are defined at the application level, they can be used in any page, in any flow within the application.

- To use a fragment in multiple pages:
  1. Open the page you want to add the fragment to.
  2. In the Page Designer, drag and drop a Fragment Container from the Components palette onto the canvas. Then in the General tab of the container's Properties pane, click **Select** to select an existing fragment. If the available fragments don't meet your needs, you can create a new fragment from the Select Fragment dialog.

     > **Tip:**
     >
     > If your fragment already exists, simply locate it in the Components palette (you can enter `frag` to filter components or scroll down to the Fragments category), then drag and drop it directly onto the canvas.

  3. Repeat the steps to add the same fragment to another page.
- To use a fragment multiple times on the same page:
  1. Open the page you want to add the fragment to.
  2. In the Page Designer, drag and drop the fragment from the Components palette onto the canvas.
  3. To add the fragment to another area of the page, drag it from the Components palette and drop it where you want it to display.
- To use a fragment within a fragment:
  1. Open the fragment where you want to use another fragment.
  2. In the Fragment Designer, drag and drop a fragment from the Components palette onto the canvas.

     Alternatively, drag and drop a Fragment Container from the Components palette onto the canvas. Then in the General tab of the container's Properties pane, click **Select** to select an existing fragment or **Create** to create a new one.

     When you add a fragment to a fragment, both pieces will display on the page consuming the initial fragment. For example, if you added emergency contacts as a separate fragment within the employee's contact information, the emergency details will display on every page that pulls in the contact information fragment.

Pass Data Between a Fragment and Its Parent Container

Passing data between a fragment and its parent container (say, a page or another container like a different outer fragment) involves enabling fragment variables as required or optional input parameters to the container. It's also possible to enable the variable to "write back" directly to the container.

Passing Data From a Page (or Outer Container) to a Fragment

When you define fragment variables and enable them as required or optional input parameters, a page or any container that consumes the fragment can provide values for the input parameters. A page, for example, can define or associate its variable to a fragment's input parameter. This way, a page variable's value is used as the initial value for the input parameter enabled in the fragment. See Enable Page Variables to Provide Initial Values for a Fragment's Input Parameters .

When the same page variable's value changes "mid-cycle", the updated value is automatically reapplied on the fragment input parameter and an `onValueChanged` event triggered on the fragment variable.

Passing Data From a Fragment to a Page (or Outer Container)

There are two ways to do this, and the option you choose really depends on your business use case:

- When a fragment variable is enabled as an input parameter, you can additionally choose to write it back to the container. This option allows changes to the fragment variable to be automatically written back to the page variable that was used as the input parameter.
- While automatic writeback to a page variable is convenient and powerful, there may be cases where multiple changes to the fragment variables occur (say, the variable's state needs to be gathered and raised via a custom event). For such scenarios, you can define custom events that "emit" from the fragment to the fragment container. A custom event that emits to the container can provide information to the page (or container) that references it. It's important to remember that there are two types of custom events that can be defined in fragments (the one discussed here is the second type in this table):

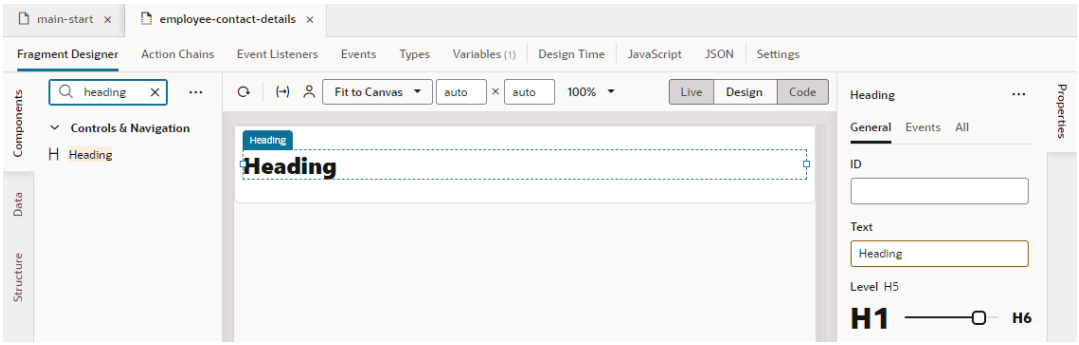
Event Type	Description
Events that can be handled by listeners defined in the fragment	These events are similar to other custom events in Visual Builder and are used to communicate consolidated state changes made in the fragment to the outer container. For example, a fragment defined as a multi-step survey may raise a regular custom event for itself to take note (say, on a Next button), but after the survey is completed, it may communicate the completion state (along with the entries) to the outer container.
Events that "emit" to the container	These events are used to propagate values to a fragment's parent container. See Create Custom Events that Emit to a Fragment's Parent Container.

Enable Fragment Variables as Input Parameters

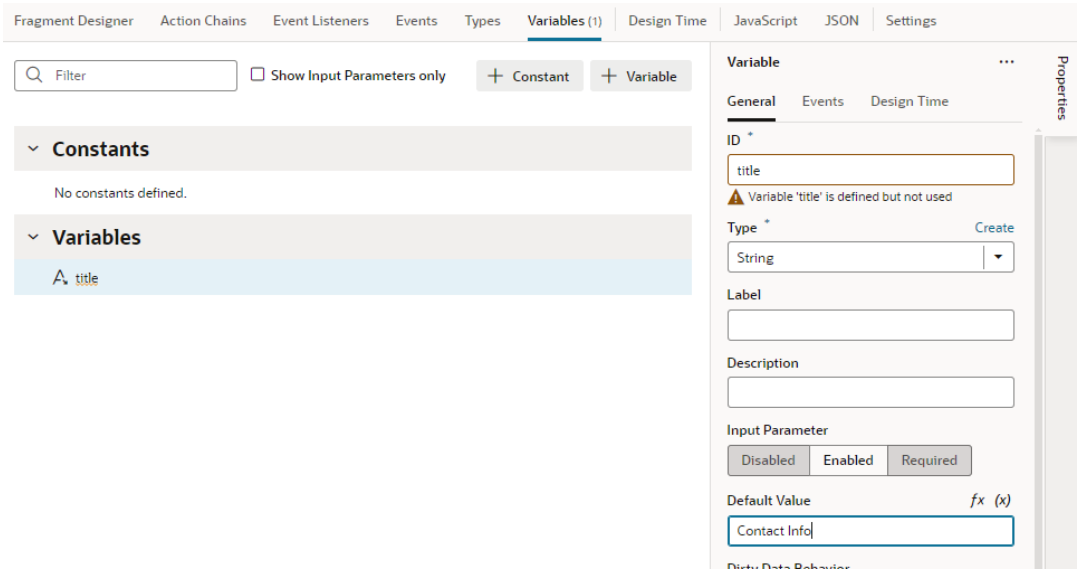
To pass values to a fragment from a page that consumes the fragment, you first define input parameters on the fragment that are either required or optional. For example, you might define a default placeholder title in a fragment variable and enable it as a parameter for a page. In this case, you'll have the option of overriding the default value with an alternate value on a particular page.

To enable a fragment variable as an input parameter:

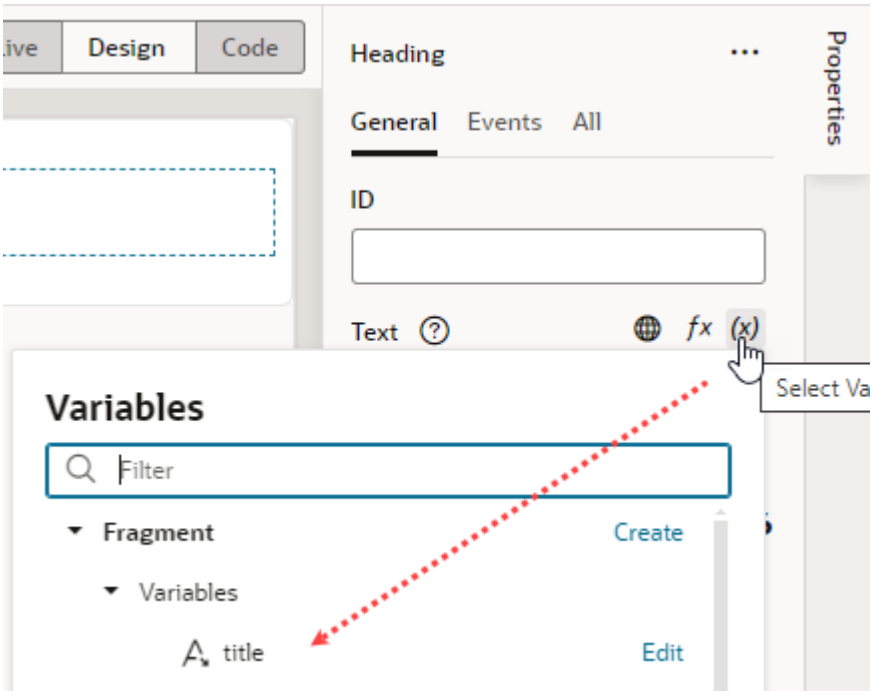
1. Define your fragment as needed, for example, an `employee-contact-details` fragment that displays an employee's contact information. Let's assume the fragment uses a heading component, as shown here:



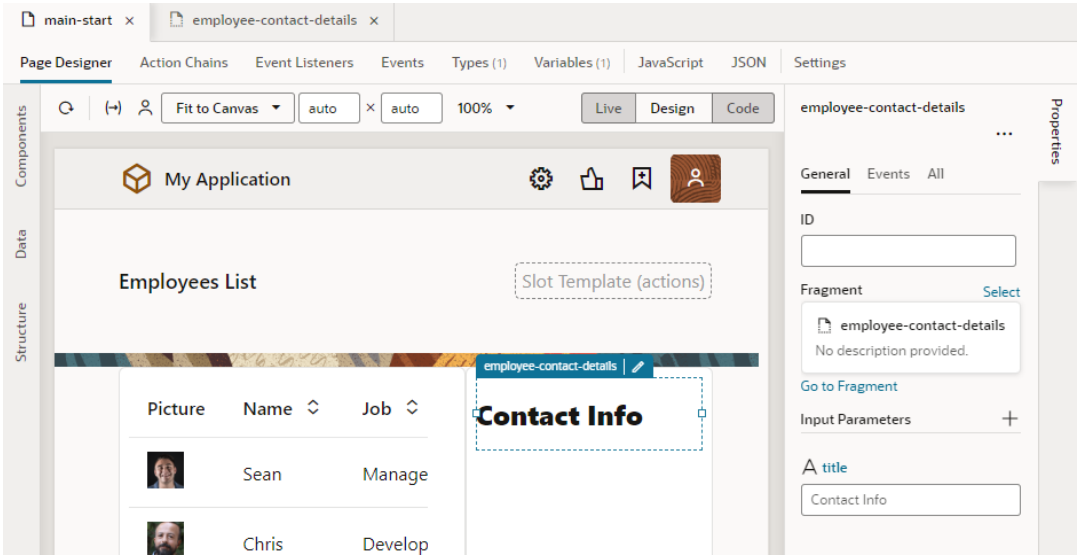
2. On the fragment's Variables tab, create a string-type variable for the title (for example, `title`) and enable it as an input parameter. Optionally, set a default value (say, `Contact Info`).



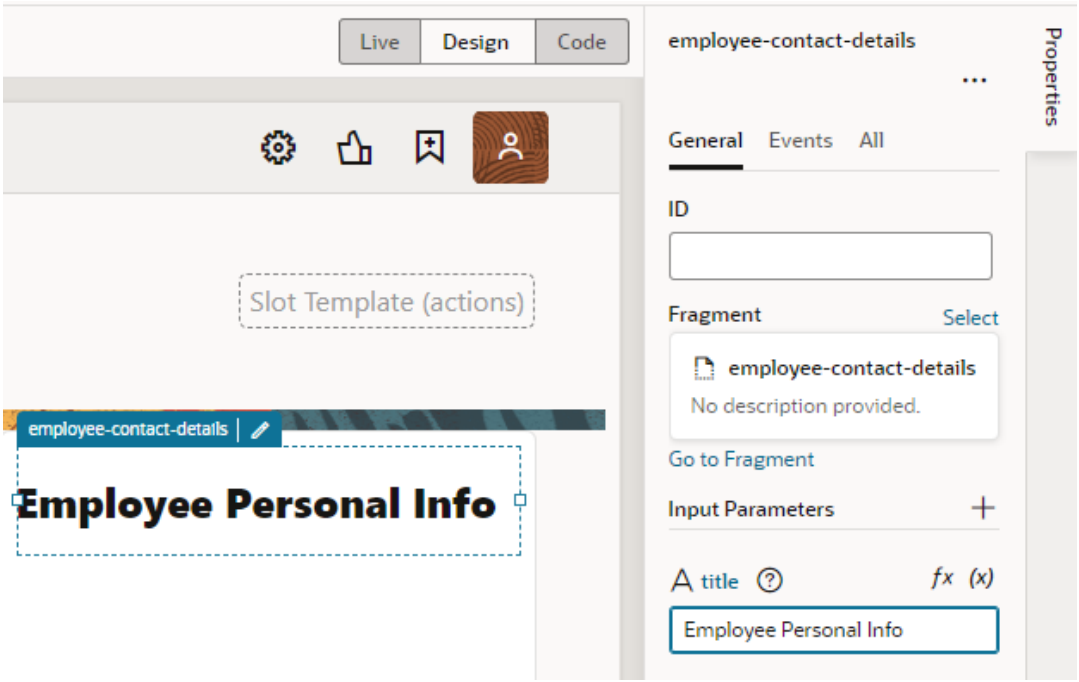
3. Switch to the Fragment Designer and bind the Heading component to the variable you just created.



4. Now open the page where the `employee-contact-details` fragment is used (drag and drop the fragment onto the canvas, if needed). Fragment variables marked as input parameters become available to you on the page.



5. You can now customize the default title to something that the page (or container) provides. To do this, in the fragment's Properties pane, under Input Parameters, update the title input parameter's value (for example, `Employee Personal Info`).

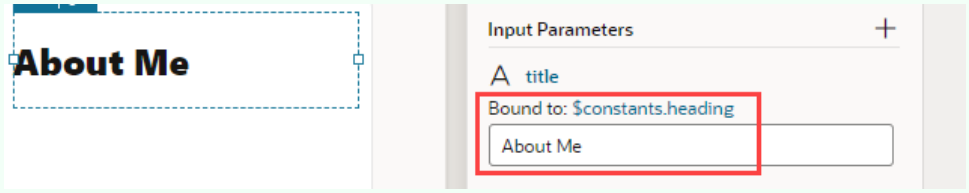


Tip:

You can extend this use case to bind the fragment input parameter to a page constant. For example, if the page author has defined a page-level constant called `heading`, when the fragment input parameter is mapped to the constant, you'll be able to edit the value of the bound constant. To do this:

a. Click ***(x)*** to open the variables picker on the `title` input parameter and select `heading` under Page and Constants.

b. Use the constant's default value if one is defined (for example, `About Me`), or enter a new value.



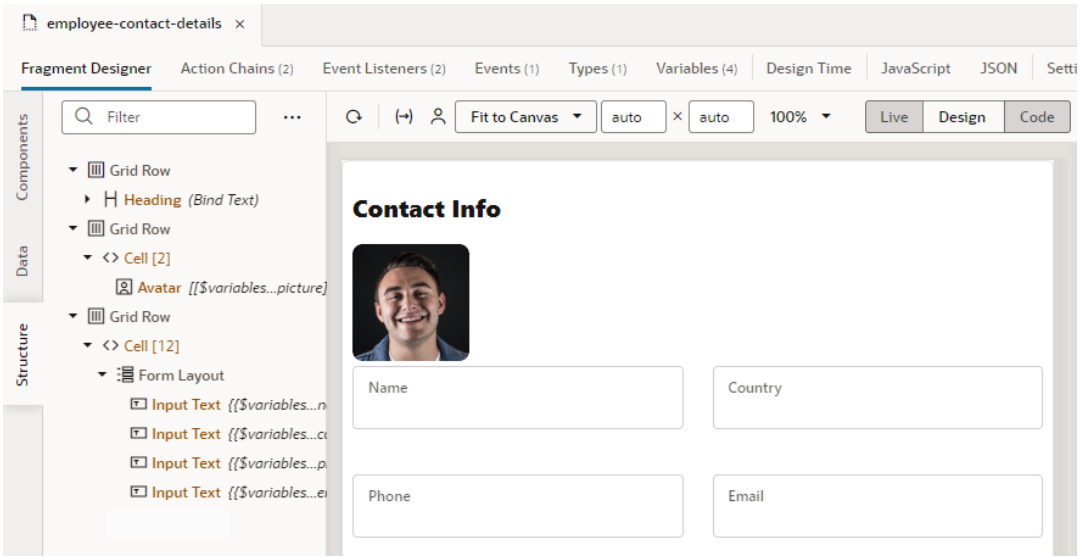
You can click the expression (`$constants.heading`) to view the constant's definition on the page-level Variables editor.

Enable Page Variables to Provide Initial Values for a Fragment's Input Parameters

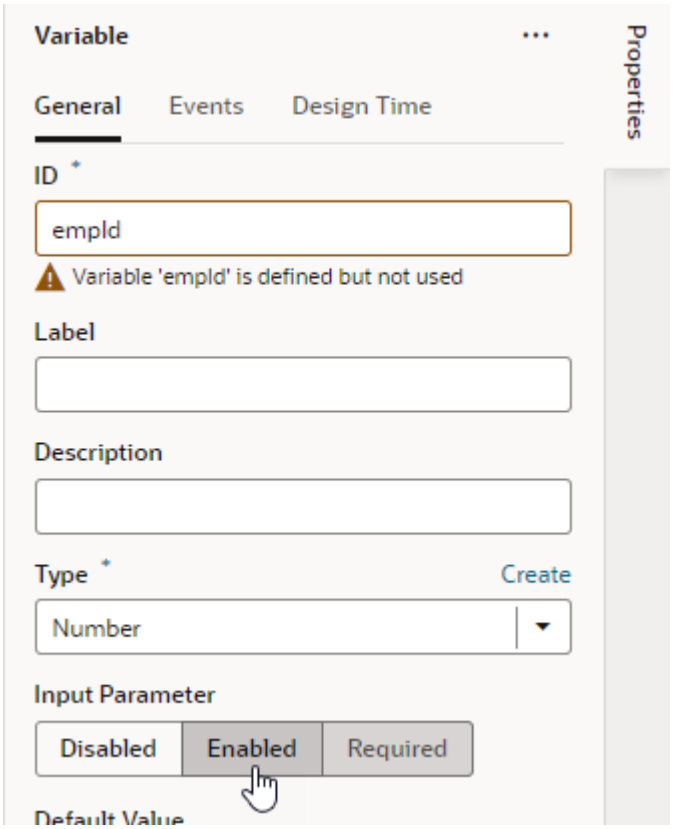
You can pass a page variable as an input parameter from the page to a fragment that it references in order to provide initial values for the fragment's input parameters.

Say you have a page that displays employee data in a table, including employee contact information defined in a fragment. When a user selects a row in the table of employees, the selected employee's contact information is displayed from the fragment. To pass the selected employee's ID from the page to the fragment, you might define a page-level variable (for example, `selectedEmp`) and pass its value to a fragment variable enabled as an input parameter (for example, `empId`) via the expression `[[$variables.selectedEmp]]`.

1. Set up a fragment to display an employee's contact information. For example, here's one that uses different components to display employee contact information:

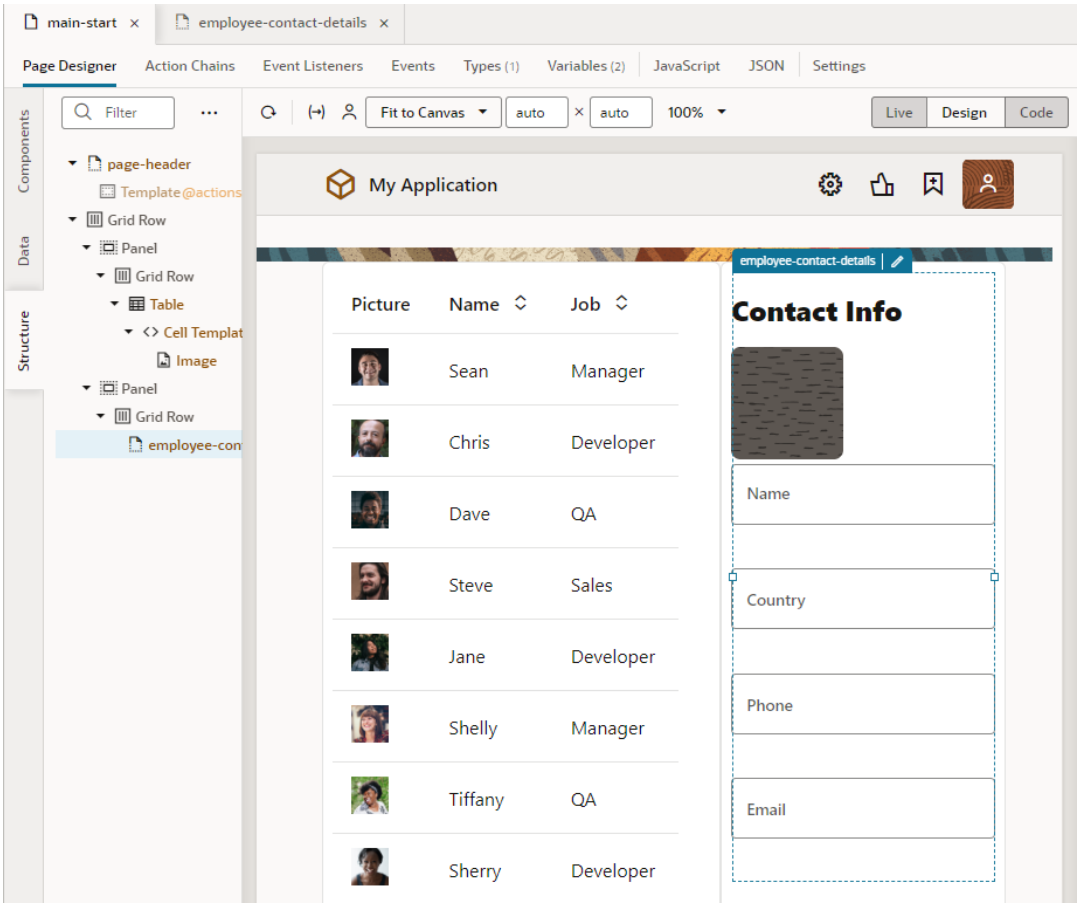


2. On the fragment's Variables tab, create a number-type variable for the employee's ID (for example, `empId`) and enable it as an input parameter.

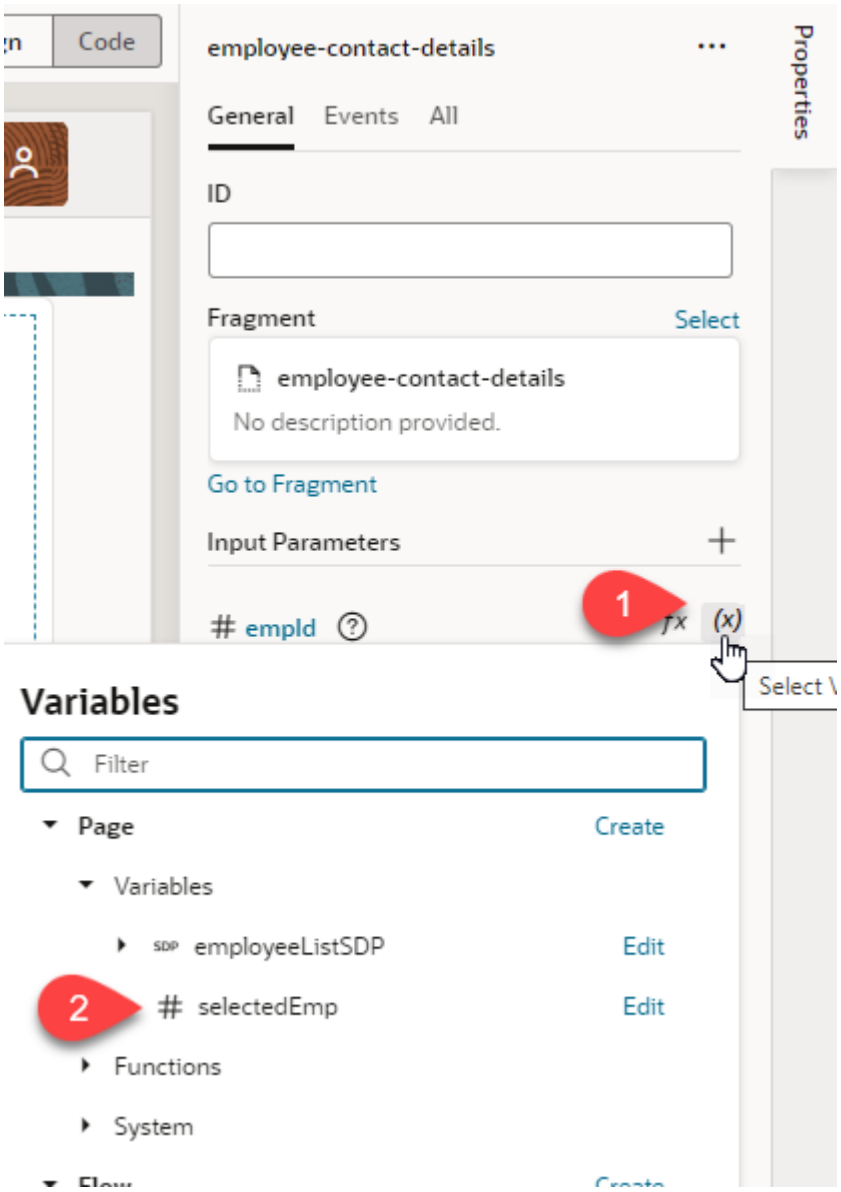


> **Note:**
>
> You can mark the variable to be automatically created on the container that uses the fragment. This way, when the fragment is dropped onto an existing page or container, the variable is created on the page and wired back to the fragment variable's value. If you don't select this option, you'll need to follow the rest of this procedure.

3. Now open the page where the `employee-contact-details` fragment is used (drag and drop the fragment onto the canvas, if needed). If it isn't already, make sure the `selectedEmp` variable is defined for the page.



4. In the fragment's Properties pane, under Input Parameters, click ***(x)*** to open the variables picker on the `empId` parameter and select the `selectedEmp` variable as the source of its value.



Assuming you've wired up the employee contact fragment to retrieve the information based on the selected employee's ID, your page will display the selected employee's record. Now any time the page variable changes (`selectedEmp`), the new value will be automatically applied on the fragment variable (`empId`).

If you want to refresh the fragment's content on the page when the variable's value changes (in other words, when another employee is selected), an `onValueChanged` event can be triggered on the fragment variable that calls an action chain to update the contact details of the newly selected employee.

Automatically Write Back a Fragment Variable's Value to Its Container Variable

When a fragment variable is enabled as an input parameter, you can mark it for writeback, allowing changes in the variable's value to be automatically written back to a variable on the fragment's parent container.

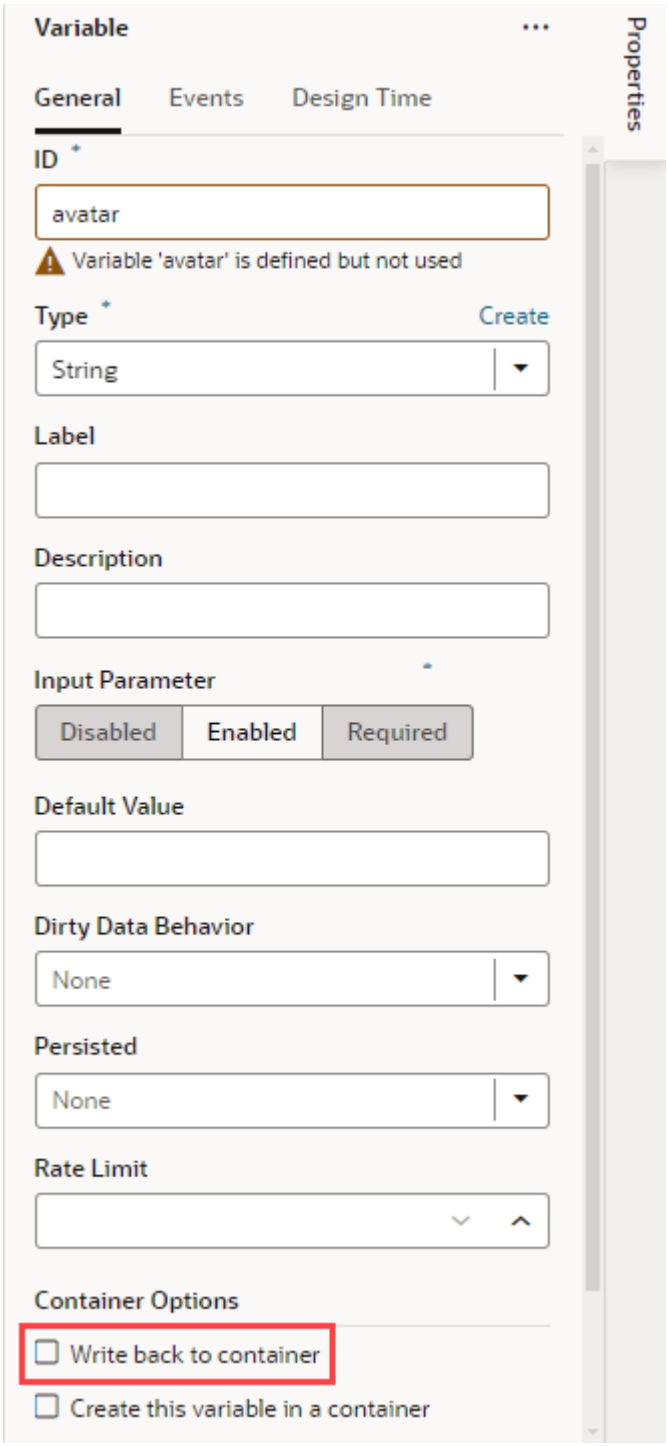
Let's say your employees page defines an `empAvatar` variable, which takes a URL as its value. When this variable is passed as an input parameter to a fragment, the fragment receives it through an `avatar` variable. This variable also has the option to write back to the container enabled. Assuming that the fragment is set up to update the employee's profile picture, when the `avatar` variable on the fragment is updated to use a new URL, the change is written back to the outer page variable (`empAvatar`). Depending on your setup, this might also update the table of employees where the new picture is shown in the currently selected row.

(To see an example of writeback in action, see the Passing Values To and From Fragments blog post.)

Writeback is supported for primitive (string, number, boolean, and any), array, and object type variables. Note that writeback is not required if an input parameter value is already passed in by reference (for example, an SDP or `dynamicLayoutContext`).

To write back updates made to a fragment variable enabled as an input parameter:

1. Open the fragment that defines the variable whose parameter value you want to be automatically updated on the parent container's variable (for example, the fragment-level `avatar` variable whose value you want to directly update on the page-level `empAvatar` variable).
2. On the fragment's Variables tab, select the variable (`avatar` in our example).
3. Select **Write Back to Container** in the variable's Properties pane:



If you don't select this option, the only other way for a parent container to be notified of updates to a fragment's variable is to raise a custom event that "emits" the event's payload to the parent container.

Events are a more formal contract that may be a better option when you want to consolidate changes made in the fragment and communicate them to the outer container, for example, when all changes made to the employees contact information are pushed to the server and the same needs to be communicated back to the page. Automatic writeback to a parent container variable, on the other hand, is desirable when you want the outer container to be notified immediately of a change to a fragment input parameter variable.

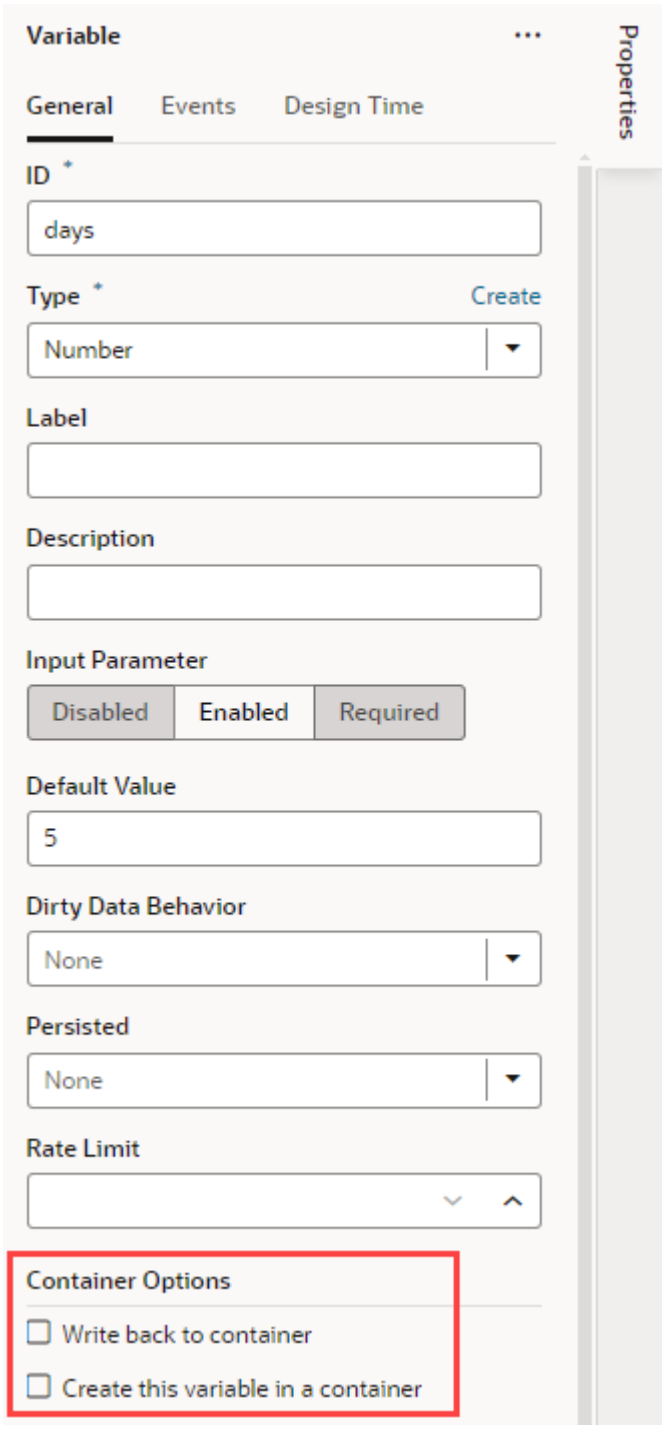
Automatically Create and Wire a Fragment Variable on Its Container

You can mark fragment variables or constants that are enabled as input parameters to be automatically created on the container that uses the fragment. This way, when a page is created from the fragment or the fragment is dropped onto an existing page or container, Visual Builder creates the variable (or constant) on the page and wires it back to the fragment variable's (or constant's) value.

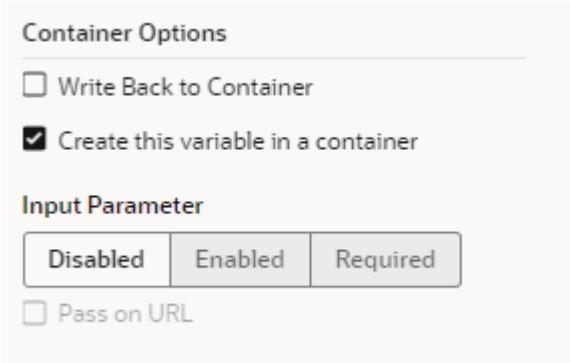
This option is especially useful when input parameters must be passed from a page for the fragment to work. By autowiring the required input parameters, you won't have to create and configure those variables on the page when the fragment is added to it, though you'd still need to assign values. Further, if the autowired variables include customizations to display an enhanced UI in the Properties pane, those customizations are also carried over to the container.

> **Note:**
>
> Only fragments tagged as `pageContent` (default) or `page` in its **Used For** setting (either from its Properties pane or Settings editor) can be autowired on their containers.

1. Open the fragment that contains the variable or constant you want to be created on the parent container.
2. On the Variables tab, select the variable or constant to view its Properties pane. When the variable or constant is enabled as an input parameter (with either **Enabled** or **Required** selected under Input Parameter), you'll see more properties under a Container Options section:



3. Select **Create variable in container** or **Create constant in container**:



When this option is selected, the `@dt.createOptions` metadata is added to the fragment's JSON definition; for example:

```
"variables": {
  "days": {
    "type": "number",
    "input": "fromCaller",
    "defaultValue": "5",
    "@dt": {
      "createOptions": {}
    }
  },
```

4. Optional: If you want to make the variable or constant on the container an input parameter of the container:
   - Select **Enabled** to make the container variable or constant an optional input parameter.
   - Click **Required** to make the container variable or constant required input parameter.
5. Optional: If you chose to pass the variable or constant as an input parameter, select **Pass on URL** to pass this input parameter to the container as part of the URL.

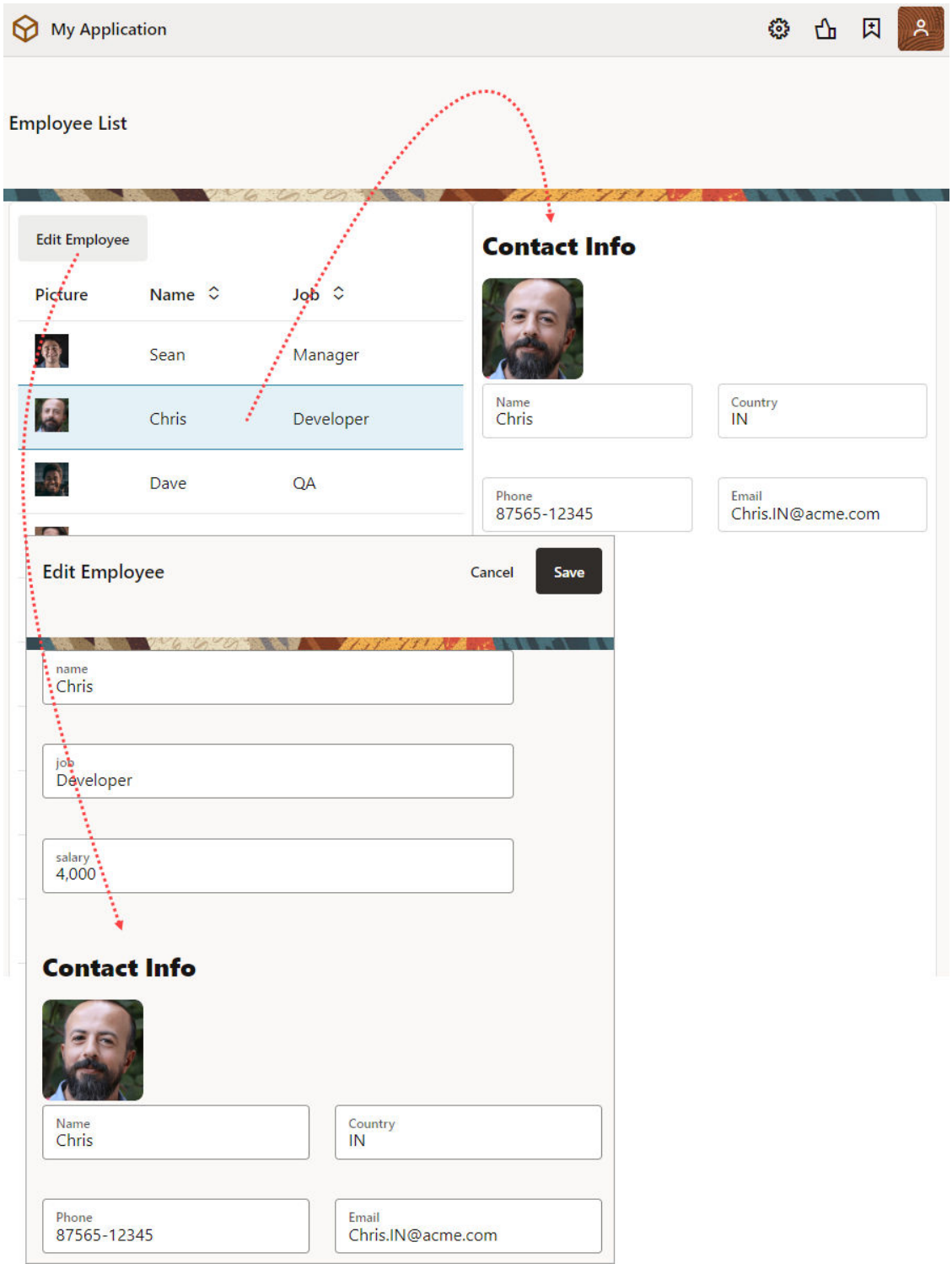
After you add the fragment to a page or a container, you'll see your variable/constant created on the page or container's Variables editor with the settings you specified.
If you were to look at your page's code, you'll see the parameters wired in HTML, for example:

```
  <oj-vb-fragment bridge="[[vbBridge]]" name="welcome" class="oj-flex-item oj-
sm-12 oj-md-12">
    <oj-vb-fragment-param name="avatar" value="[[ $variables.avatars ]]"></oj-
vb-fragment-param>
    <oj-vb-fragment-param name="days" value="[[ $variables.days ]]"></oj-vb-
fragment-param>
    <oj-vb-fragment-param name="title" value="[[ $variables.title ]]"></oj-vb-
fragment-param>
  </oj-vb-fragment>
```

Sample Scenario: Create a Fragment and Pass Values

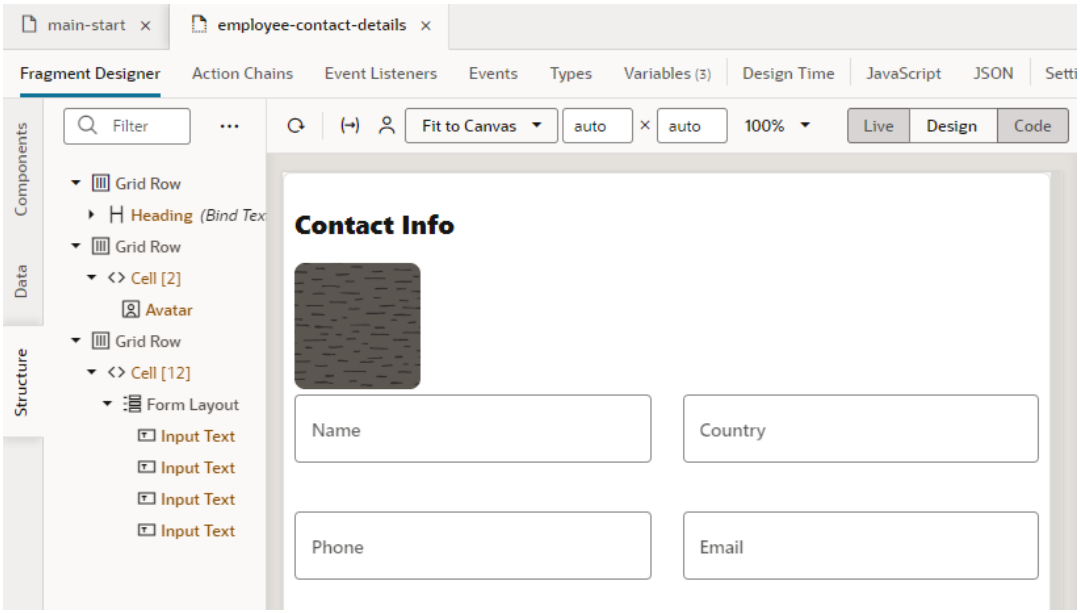
Here's a sample scenario that walks you through how to enable input parameters and pass values between a fragment and the pages that use that fragment.

Say you have a page with employee data in a table. Clicking a row in the page's table brings up the employee's contact information. You have another page that lets users edit an employee's information, including their contact details. To save time and effort, you can define the contact information part of your UI in a fragment and pull it into both pages.



In both cases, the page containing the fragment passes the selected employee's ID as an input parameter to the fragment; the fragment receives the ID, retrieves the contact information, and renders it. If a user updates the selected employee (in other words, when the input variable's value changes), the fragment raises an onValueChanged event to refresh the contact details on the page.

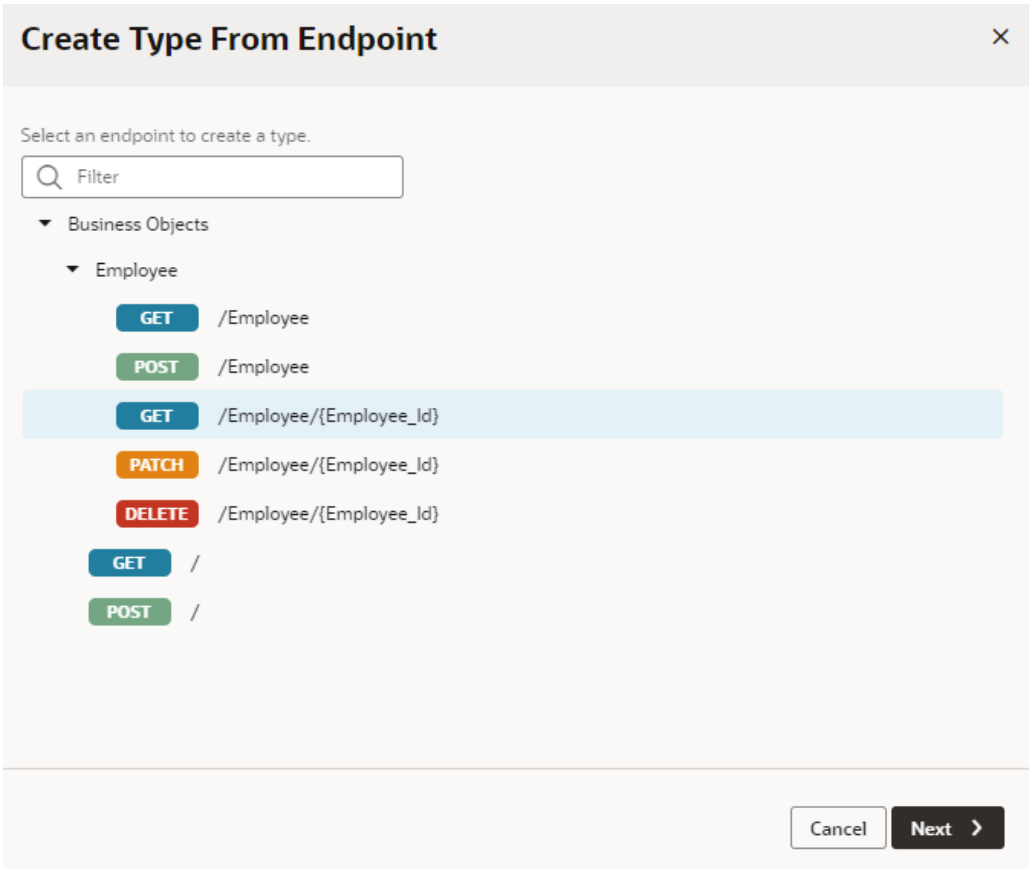
1. First off, set up a fragment to display an employee's contact information. For demonstration purposes, let's assume your fragment looks something like this, with a Heading, an Avatar, and Input Text components:



To provides values to these components, we'll create a type and a variable that holds this information, retrieved from the Employee business object's `get_employee` endpoint (our data source).

a. In the fragment where you want to define the contact details, click the **Types** tab, then click **+ Type** and select **From Endpoint**.

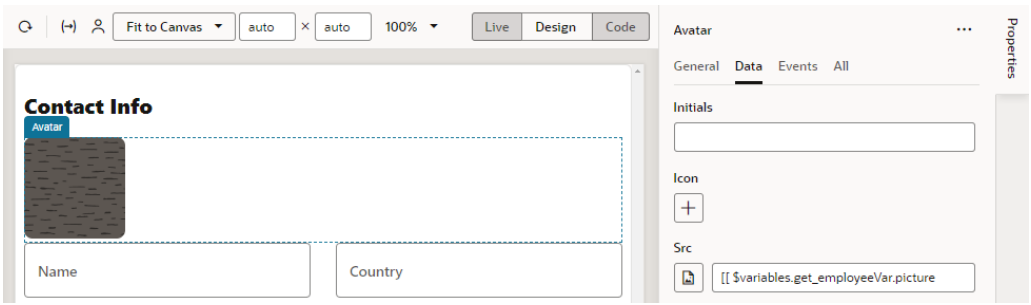
b. In the **Create Type From Endpoint** dialog box, expand Business Objects, select the `Get/Employee/{Employee_Id}` endpoint under Employee, and click **Next**.



c. Select the fields you want to display in the fragment, for example, `Picture`, `Name`, `Country`, `Email`, and `Phone`. Click **Finish**.

d. Right-click the newly created **get_Employee** type and select **Create Variable**. A new **get_EmployeeVar** (with `get_Employee` as the type) is created on the fragment's Variables tab.

e. Switch to the Fragment Designer and bind each component to the `get_EmployeeVar` variable's corresponding value. For example, to bind the Avatar to the employee's picture, click the Avatar component, then in the component's Data tab, hover over the **Src** field and click to open the variables picker. Select `picture` under the `get_EmployeeVar` fragment variable.



2. Now that we have *what* we want to display, let's set up *how* we want employee information to show in the pages that use this fragment. Some pages (like an Employees List page)

might simply display the information as read-only data while others (like the Edit Employee page) might need a way to edit it.

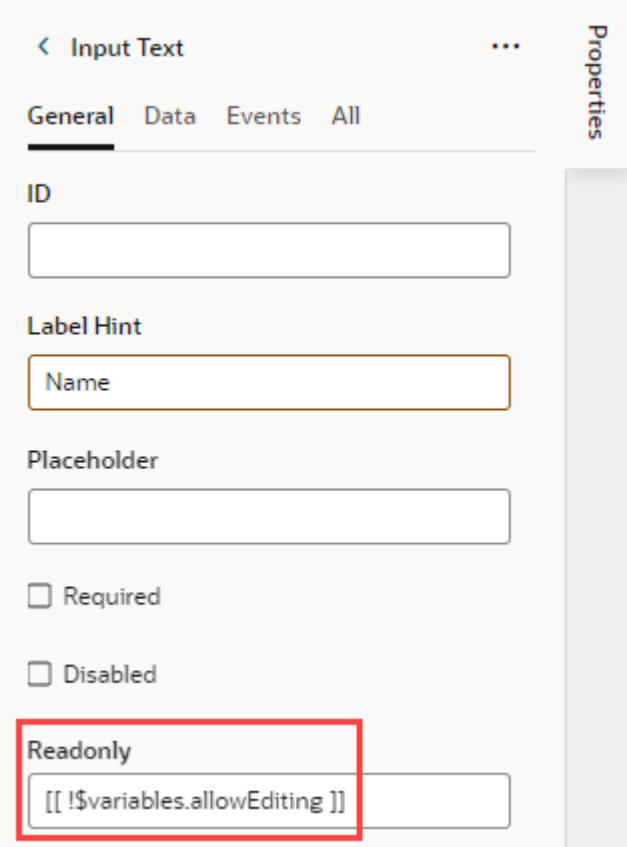
a. On the fragment's Variables tab, define a Boolean-type variable (for example, `allowEditing`).

b. Select the `allowEditing` variable, then under Input Parameters in the Properties pane, select **Enabled** to pass the variable's value as an input parameter to the pages that use the fragment.

c. Switch to the Fragment Designer and bind each Input Text component's Readonly property to the `allowEditing` variable. For example:

   i. Click the **Name** Input Text component, then in the component's **Readonly** field in the General tab, click ***(x)*** to open the variables picker and select `allowEditing` under fragment variables.

   ii. Add an exclamation mark (!) before the dollar sign ($) to indicate that the field is read-only when the fragment is *not* in edit mode.

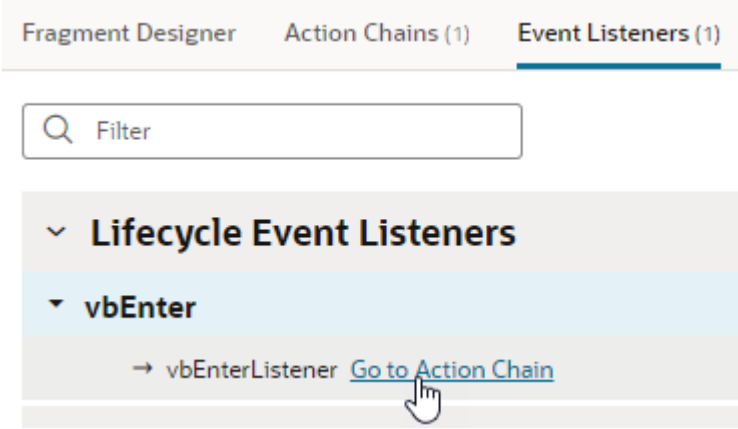


3. Because we want the selected employee's contact details to be displayed when the fragment is loaded on a page, we'll add a "vbEnter" lifecycle event that triggers an action chain to retrieve the correct employee information. This way, when the fragment is loaded, it takes the employee ID selected on the page, retrieves that employee's contact details from the data source, and passes it to a page-level variable.

   a. Click the fragment's **Events Listeners** tab, click **+ Event Listener**, and select **vbEnter** under Lifecycle Events. Click **Next** .

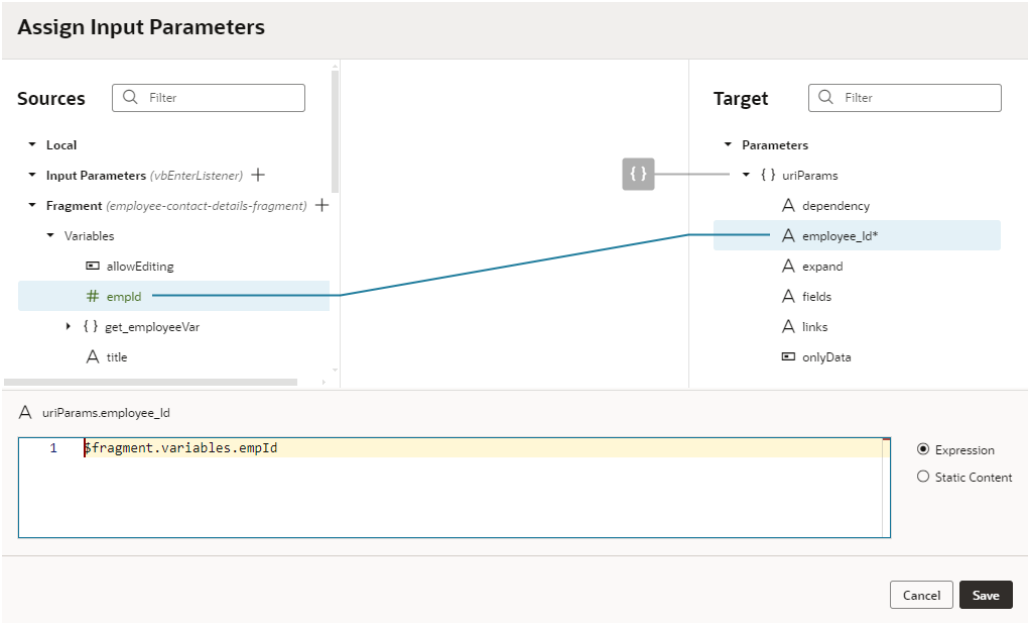
   b. Select **Create Fragment Action Chain** and click **Finish** to create an action chain called `vbEnterListener`.

c. Hover next to **vbEnterListener** (under Lifecycle Event Listeners and vbEnter) and click the **Go to Action Chain** link.

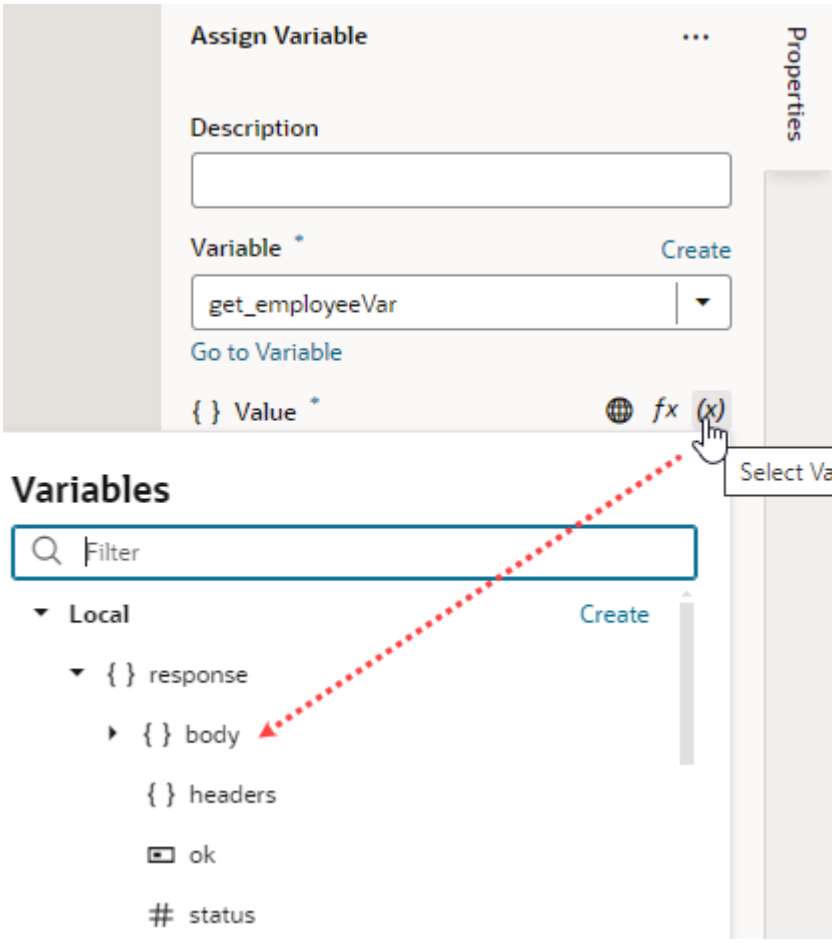


d. When the `vbEnterListener` action chain opens in the editor, drag and drop a Call REST action onto the canvas. In the action's Properties pane, click **Select** next to Endpoint and choose the `Get/Employee/{Employee_Id}` endpoint under Business Objects and Employee. Click **Select**.

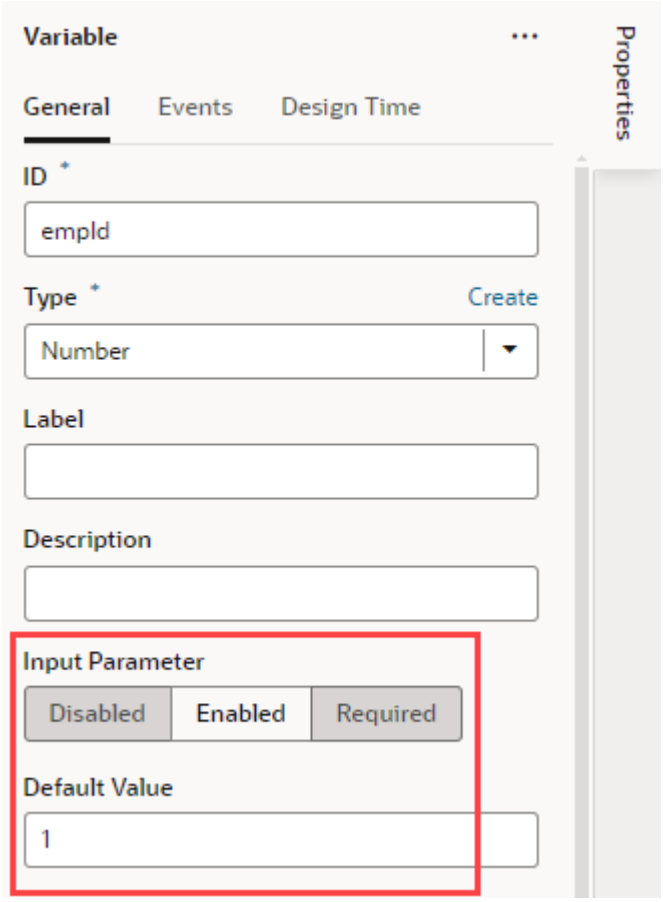
e. Under Input Parameters in the action's Properties pane, click `employee_Id` to open the Assign Input Parameters dialog. On the Sources pane, click **+** next to Fragment and create an `empId` variable of type number (you can choose to enable `empId` as an input parameter even now, but for demo purposes, we'll do this in a later step). Click **Create**. Now drag `empId` from the Sources pane to `employee_Id` on the Target pane. Click **Save**.



f. Now double click the Assign Variable action in the palette to add it to the end of the action chain. In the action's Properties pane, select `get_EmployeeVar` under Fragment in the **Variable** drop-down list. Hover over the **Value** property and open the Variables picker, then select `body` under Local and `response`.

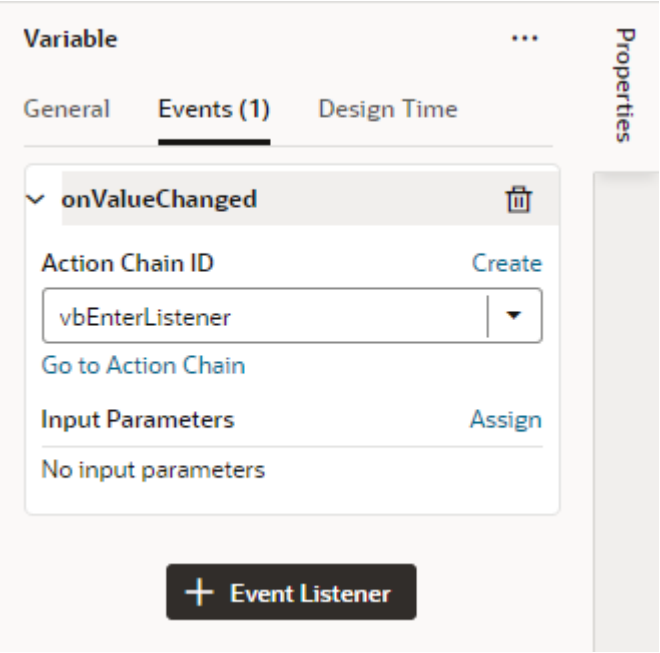


g. Switch to the **Variables** tab and look for the `empId` variable you created. Select it, then under Input Parameters in the Properties pane, select **Enabled** to pass the variable's value as an input parameter to the page consuming the fragment. Enter `1` as the **Default Value** for the input parameter.



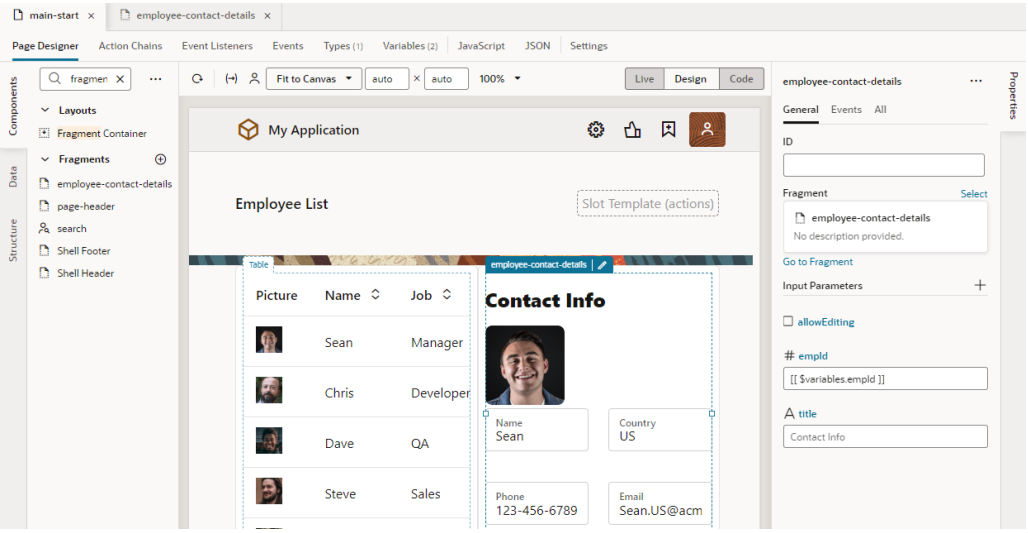
h. With the variable enabled as an input parameter, select **Create this variable in a container**. This option automatically creates this variable on the page that uses this fragment and wires its value back to the value of the fragment input parameter.

i. Click the variable's **Events** tab, then click **+ Event Listener** and select the `vbEnterListener` action chain that was previously created for you. Click **Select**.



This way, when the input variable's value (which is the selected employee ID) changes, an "onValueChanged" event triggers the `vbEnterListener` action chain, telling the fragment to update its content on the page.

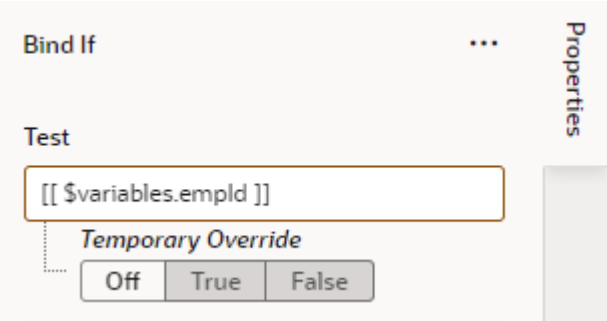
4. Now add the employee contact information fragment to a page.
   a. Open the page with employee data that you want to add the fragment to.
   b. In the Components palette on the Page Designer tab, search for the employee contact information fragment, then drag and drop it where you want it to display.



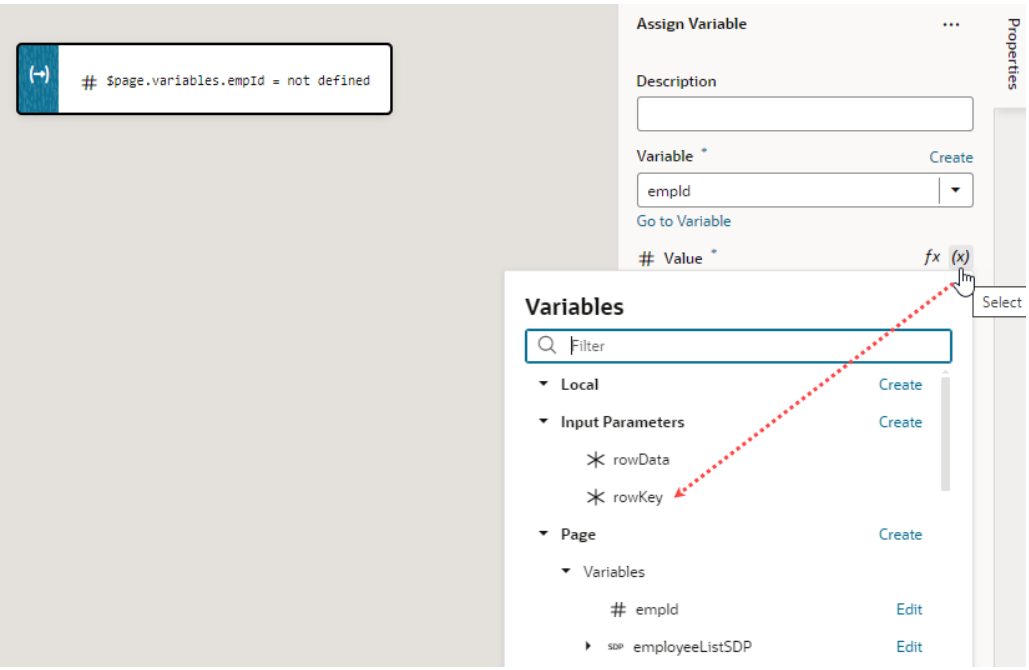
Take note of the fragment's Properties pane on the page. Because we marked the `empId` fragment input parameter to be autowired on the fragment container, the `empId` variable is automatically created on the page (look in the Variables tab) and wired back to the fragment input parameter's value (`1` by default).

   c. Now let's wrap the Fragment in an If, so that it shows only when a row is selected in the employee table. To do this, select the fragment, right-click, then select **Surround**

and **If**. If necessary, select the **Bind If** component in the Structure view; then in the Properties pane's Test condition, use the Variables picker to select the `empId` variable.



5. Set up the employee table to retrieve information based on the employee ID in the selected row.
   a. Click the table component on the page, select the **Events** tab, then click **+ Event Listener** and select **On 'First Selected Row'** to retrieve information about the selected row.
   b. When the `TableFirstSelectedRowChangeChain` action chain is created, drag and drop an Assign Variable action.
   c. In the action's Properties pane, select the `empId` variable under Page in the **Variable** drop-down list. Now use the Variables picker on the **Value** property and select `rowkey` under Input Parameters.



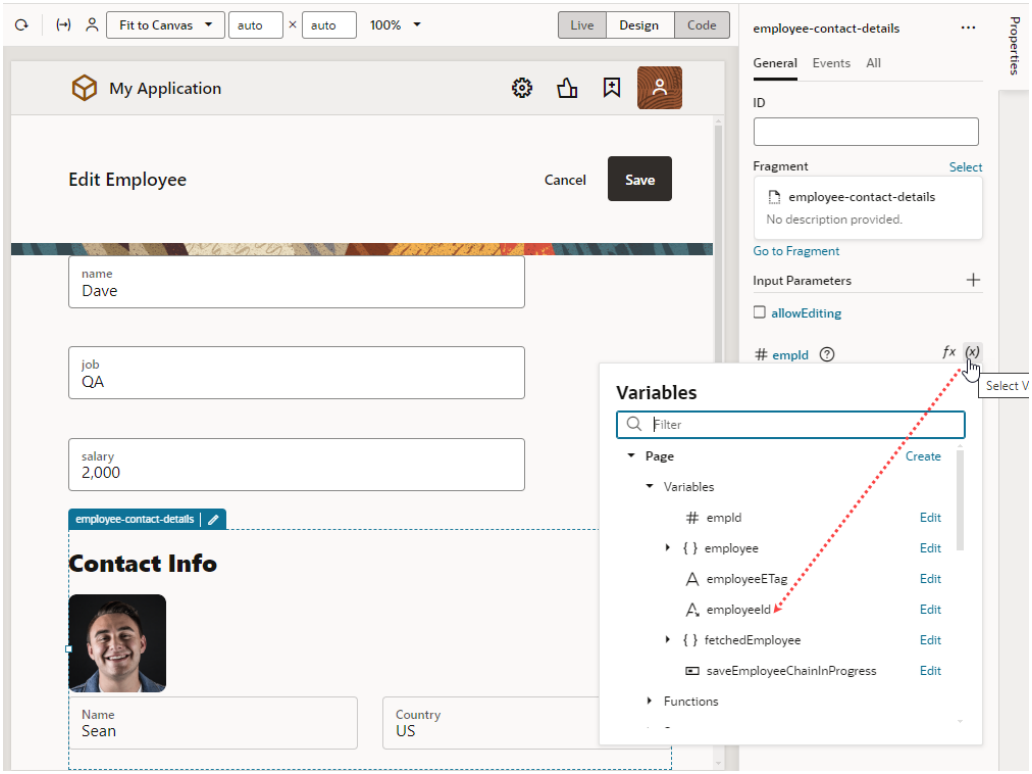
   d. Return to the page designer and click **Live**, then select a row in the table to see employee contact information reflected in the fragment based on the ID.
   e. Return to **Design** view.
6. Add the same contacts fragments to another page.

a. In the Page Designer, select the table and add an edit page using the Add Edit Page quick start.

b. Click **Live**, select an employee in the table, and click **Edit Employee** to open the newly created edit page.

c. Add the contacts fragment to a page, similar to how you added it to the other page previously.

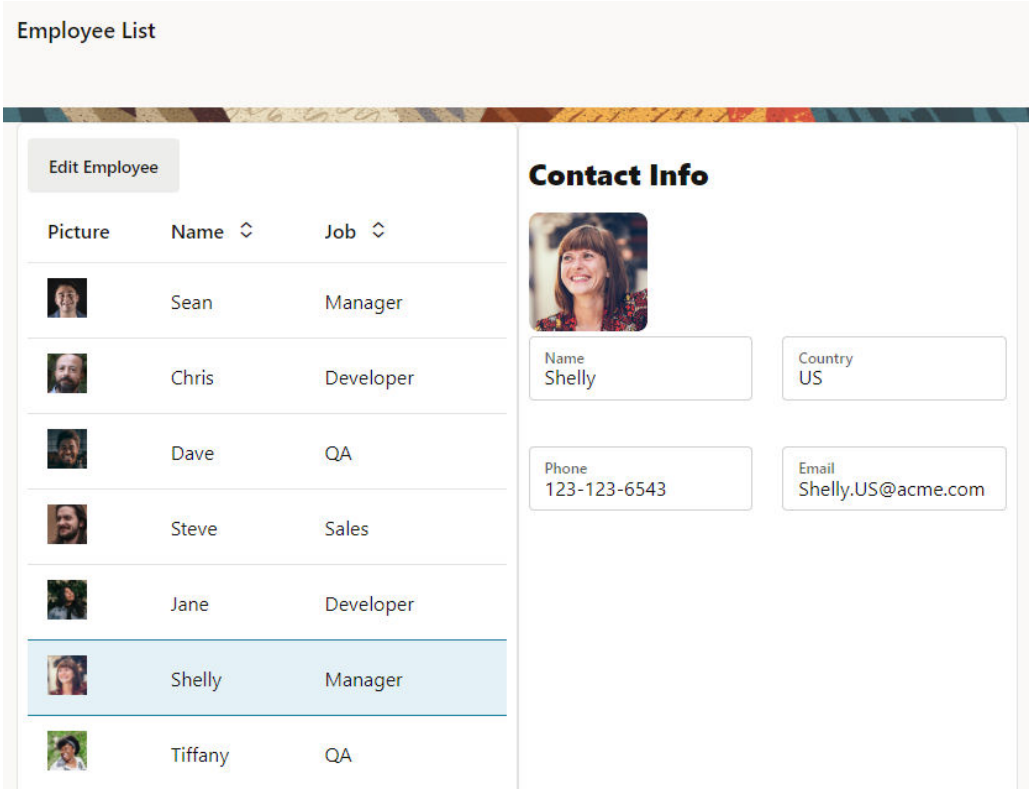
d. To make sure the selected employee ID is passed between the page and the fragment, select the Fragment and on the `empId` input parameter in the Properties pane, select the `employeeId` variable.



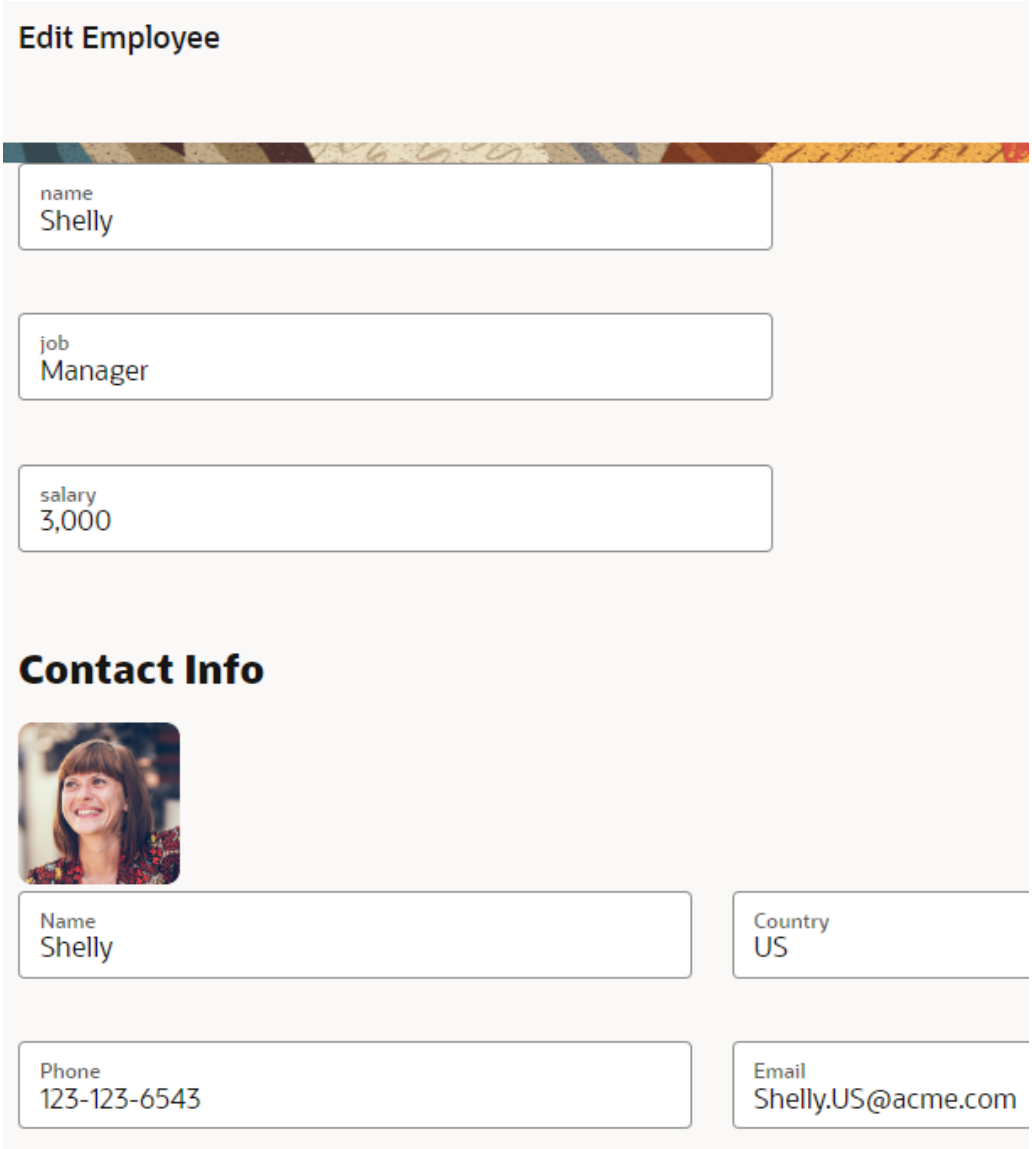
e. Because we want the fragment's contact information to be editable on this page, select the `allowEditing` input parameter in the Properties pane.

7. As a final step, click the Preview icon in the header.

a. Select a row in the table to see the employee's contact details display on the right. Notice how contact details from the fragment show as read-only values.



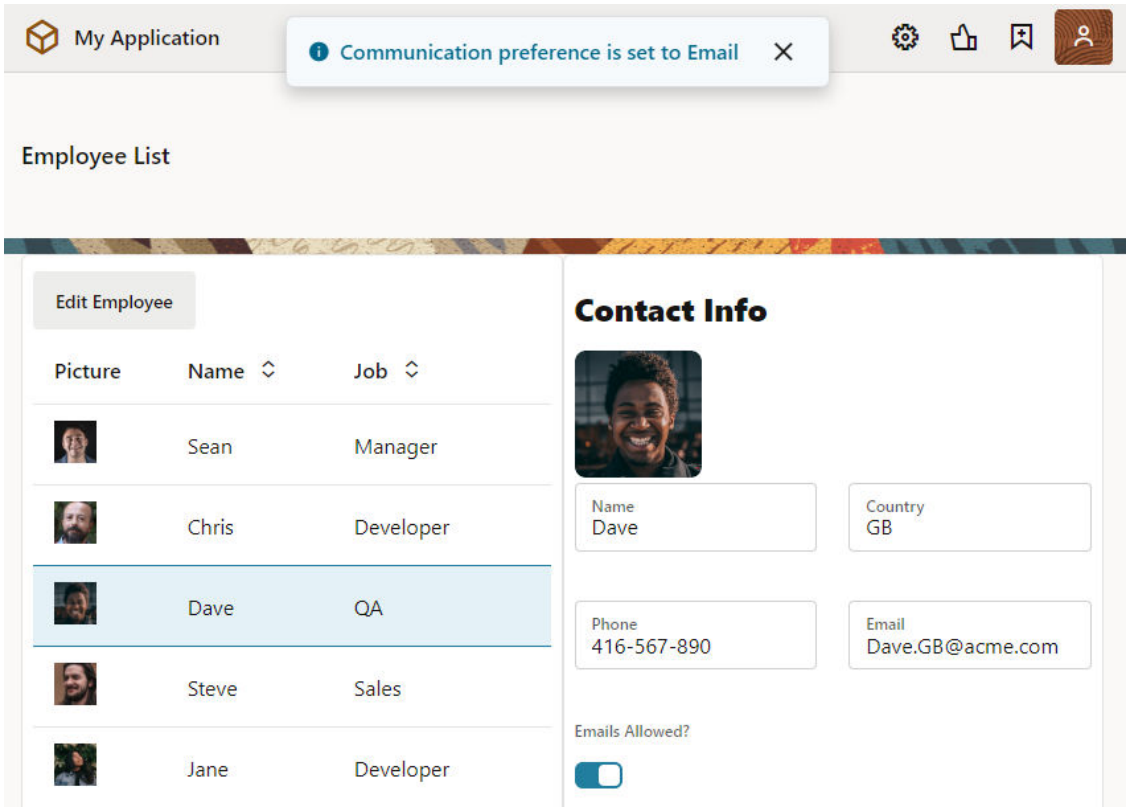
b. Click **Edit Employee** for the selected row to view and edit the employee's information, including contact details, on the Edit Employee page. Notice how contact details from the fragment show as editable values.



Create Custom Events that Emit to a Fragment's Parent Container

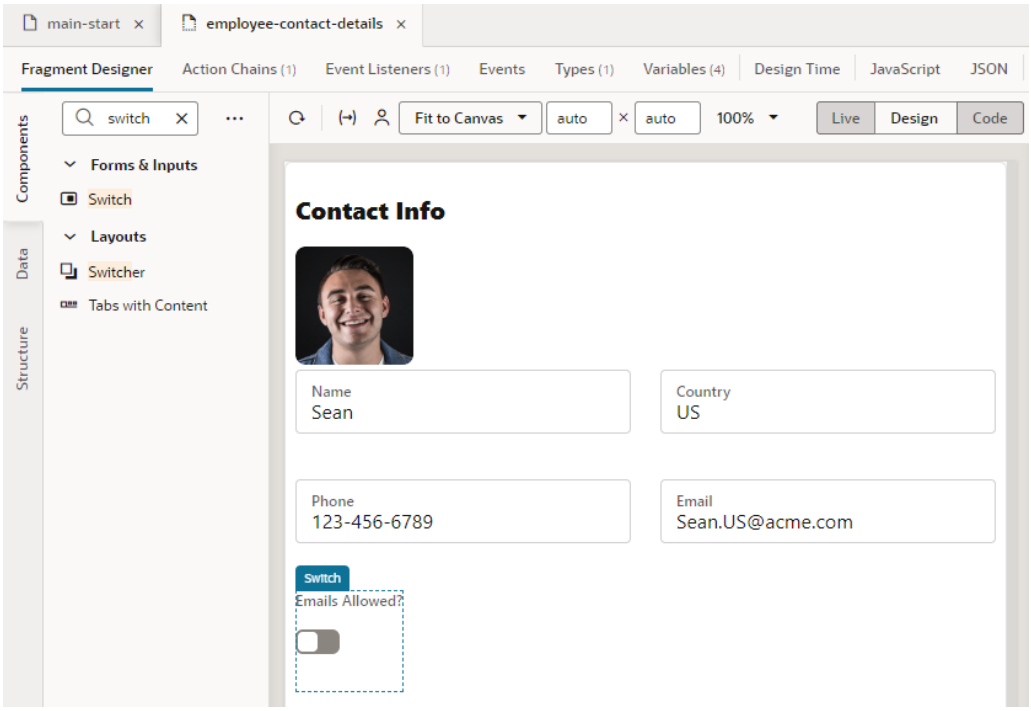
One way to pass data from a fragment to its parent container (say, a page or another container like a different outer fragment) is by raising custom events that "emit" to the container.

Let's extend the employee contacts use case to see how to do this. Here, the contacts fragment lets you specify whether email is an employee's preferred means of communication. A user who toggles the **Emails Allowed?** switch in the fragment will see a notification that their communication preference has been set to email:



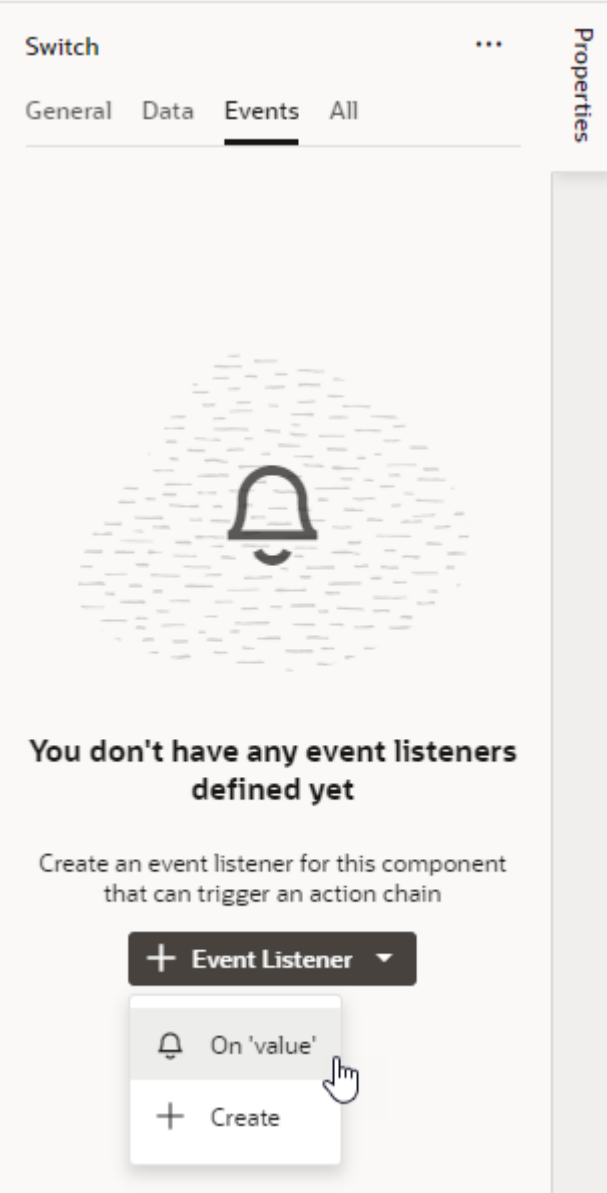
Behind the scenes, when the user toggles the **Emails Allowed?** switch, an action chain defined on the fragment fires an event that "emits" the event's payload to the fragment container. An event listener on the fragment container, watching for this fragment event to fire, triggers a page-level action chain to perform some action—which, in our example, is firing a notification that the selected employee's email preference has been set.

1. Configure your fragment to raise a custom event that the parent container listens to.
   a. Drag and drop a Switch component to your contacts fragment.



b. In the Properties pane, change the **Label Hint** text in the **General** tab to `Emails Allowed?`.

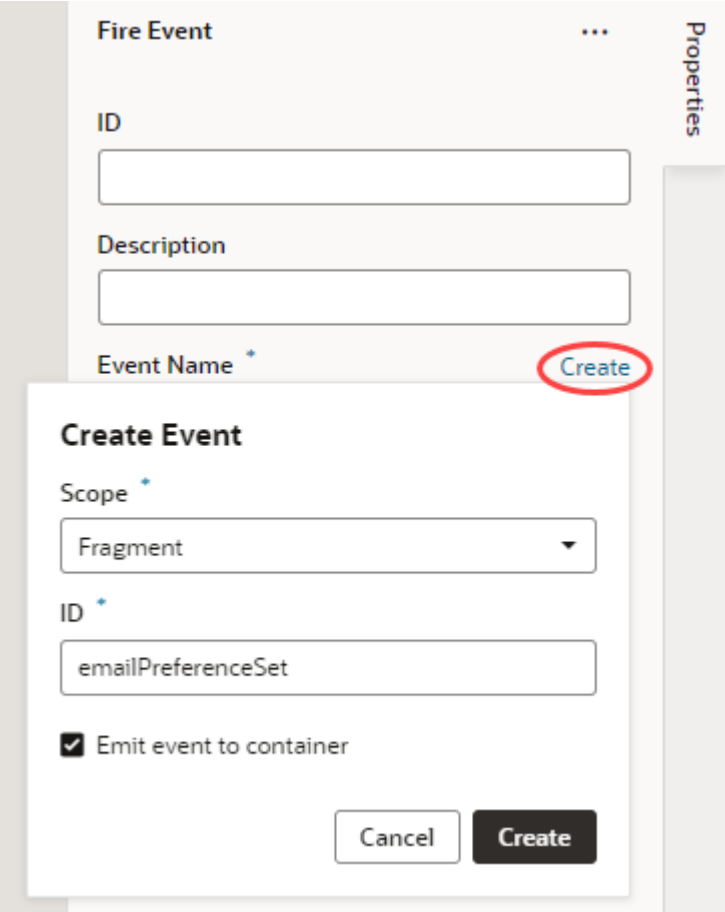
c. Switch to the component's **Events** tab, then click **+ Event Listener** and select **On 'value'**.



d. In the `SwitchValueChangeChain` action chain, drag and drop a Fire Event action.

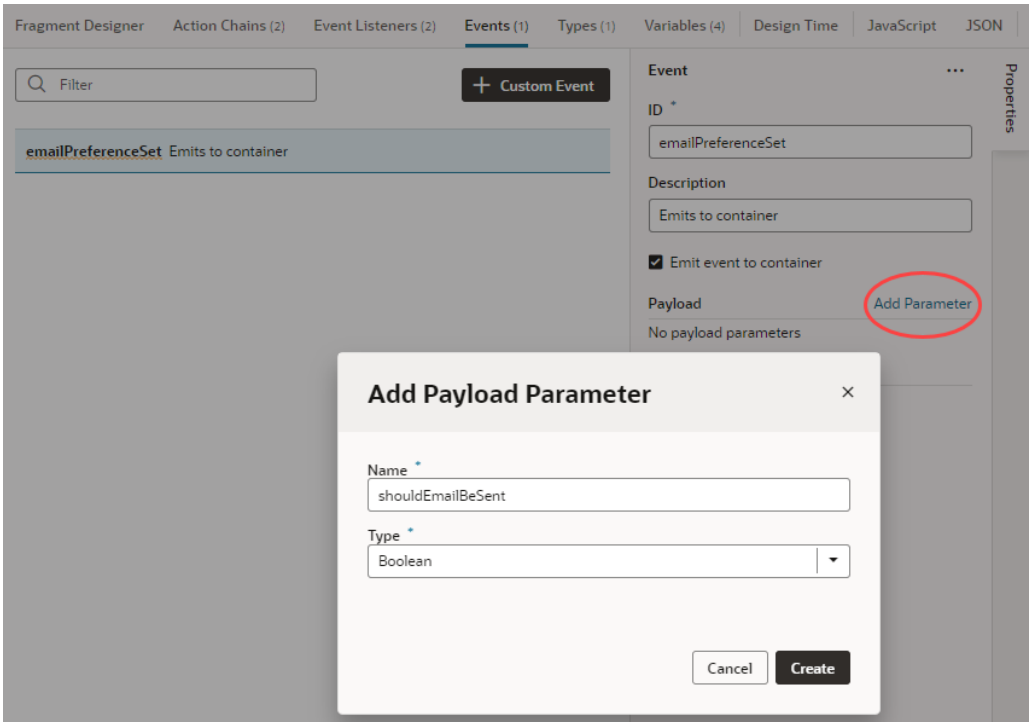
e. Click **Create** next to Event Name in the Properties pane.

f. With the Scope set to Fragment, enter an Event ID (for example, `emailPreferenceSet`), select the **Emit event to container** option, and click **Create**. Make sure the event name starts with a lowercase letter, though camel case is allowed. Hyphens are not supported.



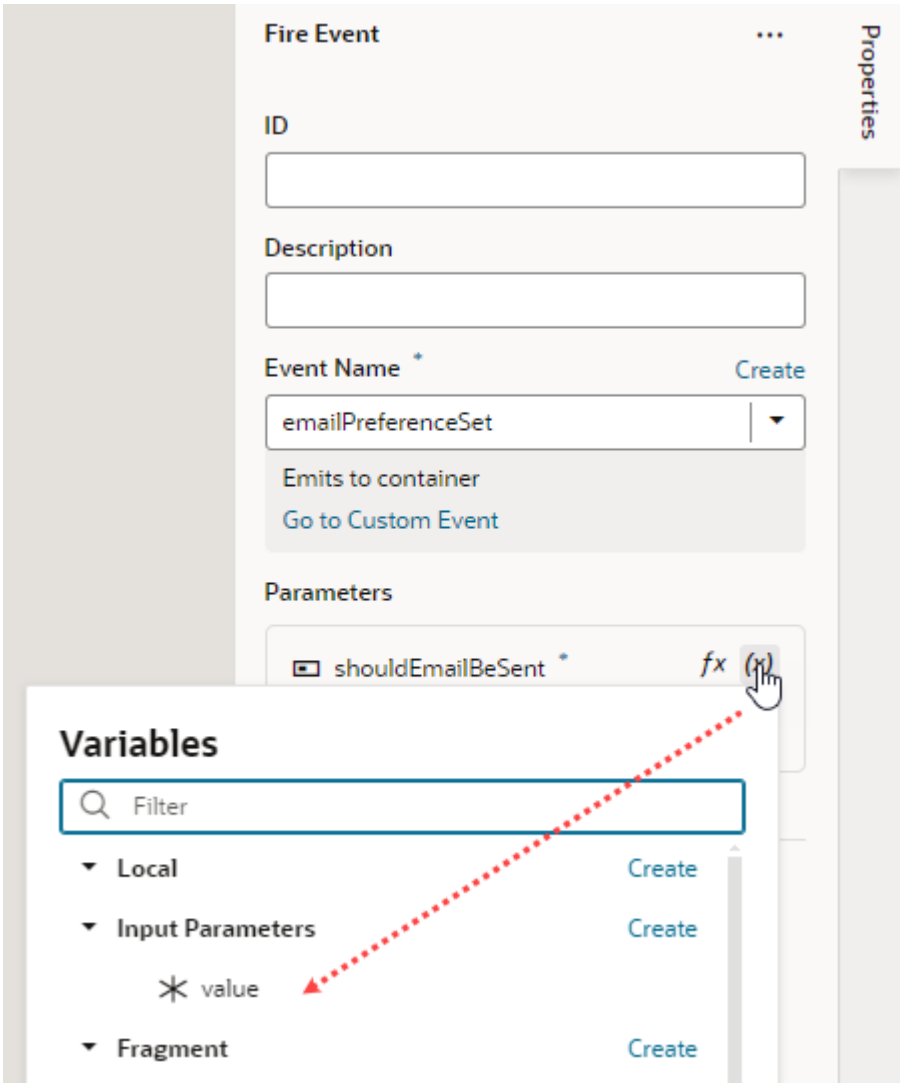
g. Click **Go to Custom Event** under the new event to go to the Events editor, where you can define the event's payload.

h. In the `emailPreferenceSet` event's Properties pane, click **Add Parameter** next to the Payload property. In the Add Payload Parameter dialog, enter an ID (say, `shouldEmailBeSent`), select the type as Boolean, and click **Create**.



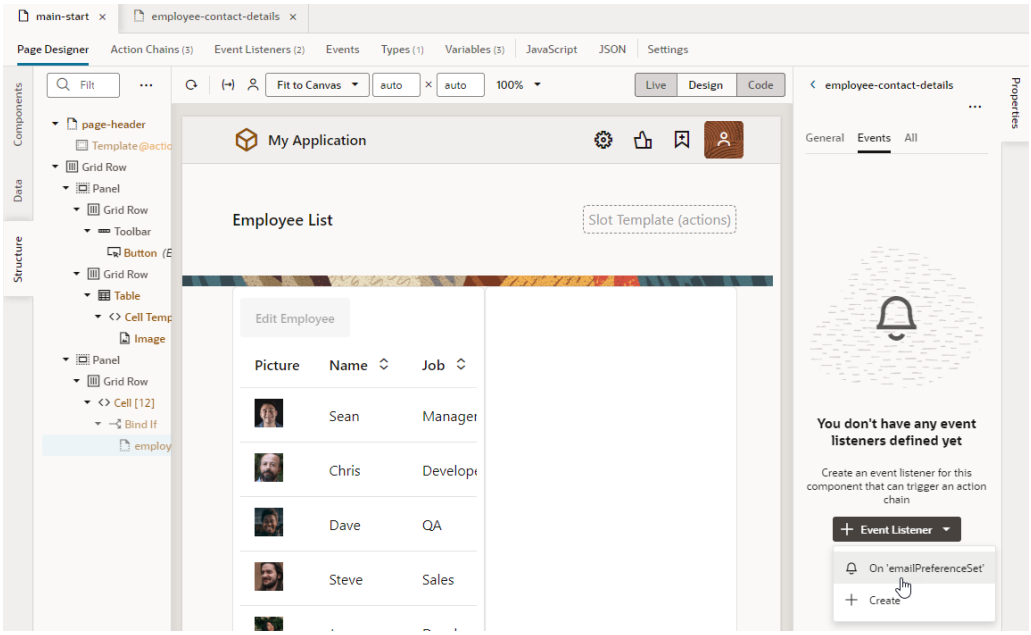
i. Under Triggered by, click **SwitchValueChangeChain** to return to the action chains editor. Switch to **Design** view.

j. In the Fire Event action's Properties pane, use the Variables picker next to `shouldEmailBeSent` under Parameters and select **value** under Input Parameters.



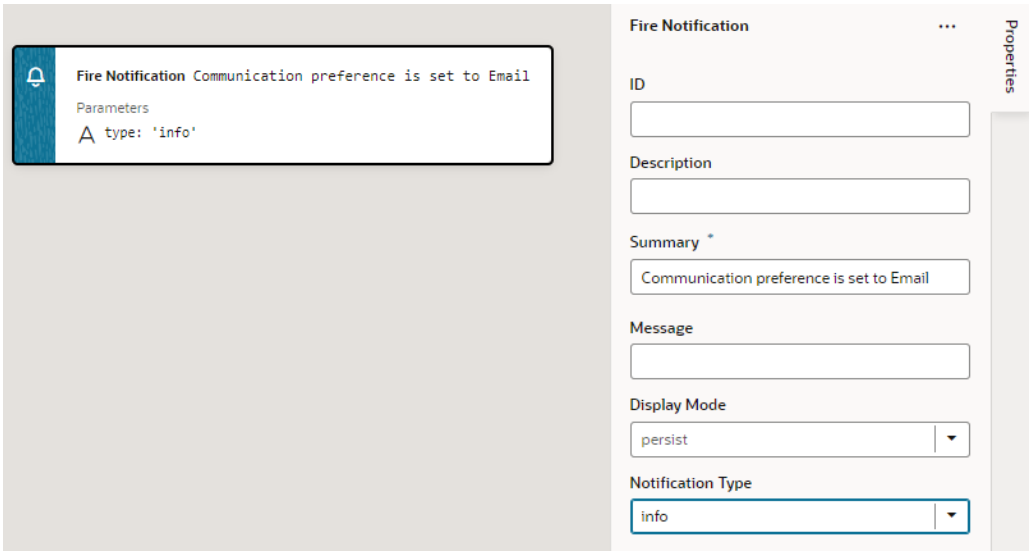
2. Configure your page's fragment container to receive and process the fragment's custom event.

   a. In the Page Designer, select the particular fragment in Structure view, then in the fragment Properties pane's **Events** tab, click **+ Event Listener** and select the **On 'emailPreferenceSet'** custom event.



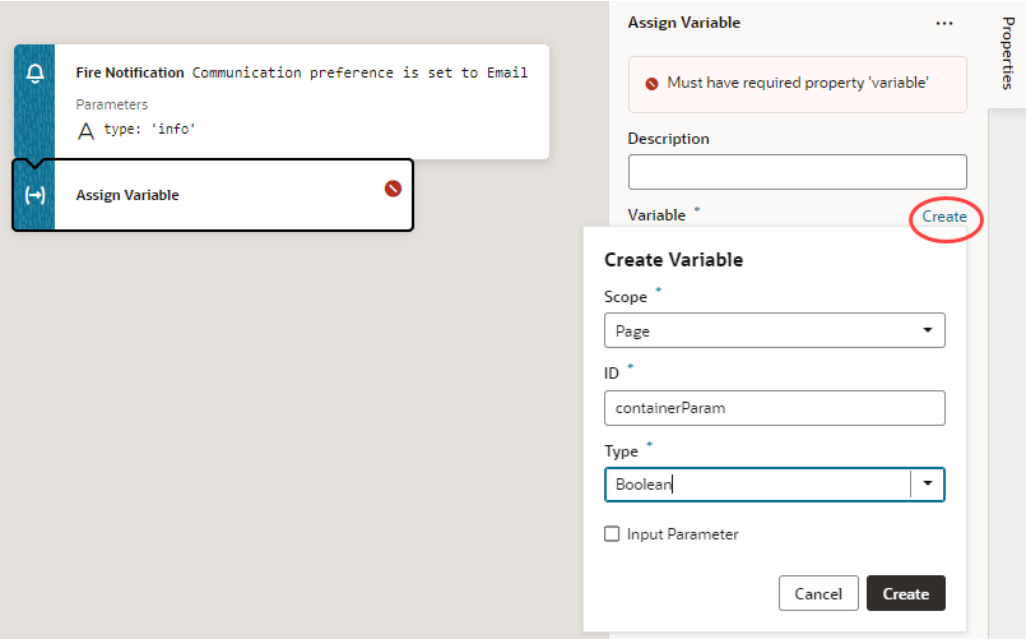
b. Drag and drop a Fire Notification action to the `FragmentEmailPreferenceSetChain` action chain.

c. Enter a summary in the Properties pane, something like `Communication preference is set to Email`, then select **Info** as the Notification type.

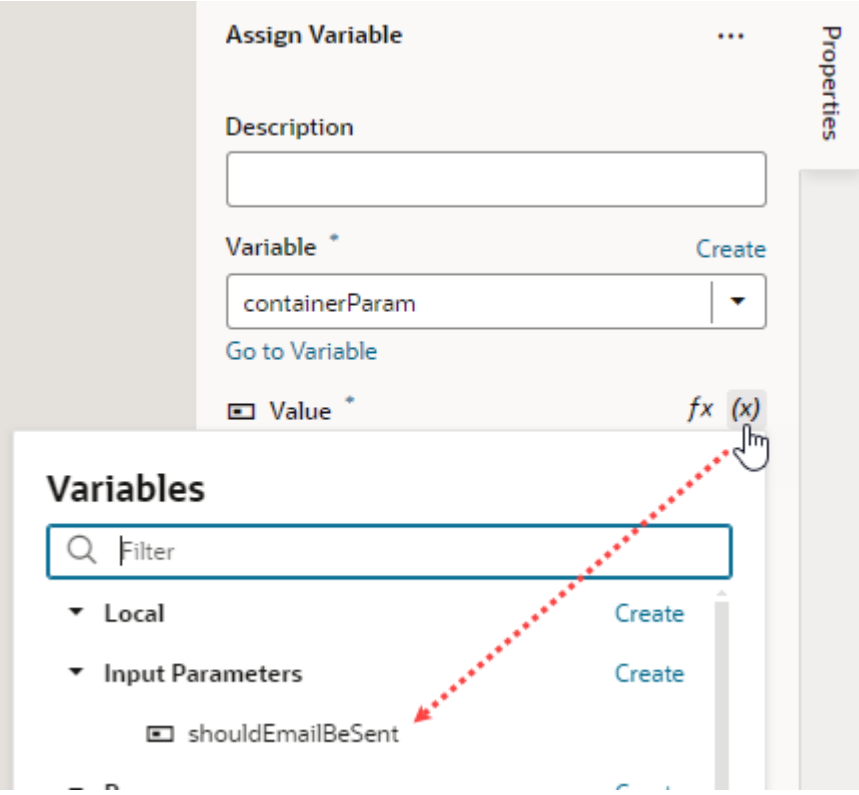


d. Now drag and drop an Assign Variable action under the Fire Notification event.

e. In the action's Properties pane, click **Create** next to the **Variable** property. In the Create Variable dialog, add a new page-level variable (for example, `containerParam`) of type Boolean and click **Create**.



f. Hover over the **Value** property and click ***(x)*** to open the Variables picker, then select **shouldEmailBeSent** under Input Parameters.



3. Now click the Preview icon in the header, select an employee, and toggle the **Emails Allowed?** switch. You'll see a message that the employee's email preference is set.

Set the Binding Type for Variables in Dynamic Components

When a variable is used as an input parameter in a dynamic component in a fragment, you can assign a subtype to the variable to indicate how the input parameter is to be used.

Subtypes are typically used to configure how variables are displayed in the Properties pane, but there are some special subtypes that are used to set the *binding type* for variables in fragments, and do not affect how the variables are displayed in the Properties pane.

Specifying a binding type provides information that Visual Builder requires to generate suitable metadata and expressions. The subtype you select should be based on the type of component where the variable is used. For example, if the variable will be used in a Dynamic Form Template, you would set the subtype to Dynamic Field.

To assign a subtype to a fragment variable:

1. Open the fragment's Variables editor.
2. Select the variable or constant.
3. Open the **Design Time** tab in the Properties pane.
4. Select the subtype for the fragment variable. Here are the subtypes that are used to set a binding type for a variable:

Subtype	valueOptions	Usage
Dynamic Field		Use Dynamic Field if the parameter will be bound to a Dynamic Field Binding (`oj-dynamic-bind-field`) component (which renders fields inside a Dynamic Form Template). In this case, the appropriate expression will be generated when a field is added to the fragment parameter, for example, `value="[[ $fields.EmployeeName.name ]]"`.
Dynamic Field Array	*none*	Use Dynamic Field Array if the parameter will be bound to a For Each (`oj-bind-for-each`) component where its template contains a Dynamic Field Binding (`oj-dynamic-bind-field`) component. In this case, the appropriate expression will be generated when fields are added to the fragment parameter, for example, `value="[[ [$fields.FirstName.name, $fields.LastName.name] ]]"`. Using the template, the For Each binding duplicates markup sections for each field in the array and binds each field to the corresponding `oj-dynamic-bind-field` in the markup section. '
Dynamic Container	`section`	Use Dynamic Container if the parameter will be bound to a Dynamic Container (`oj-dynamic-container`) component that will be configured differently on the pages it is used (meaning, to show some sections on one page and another set of sections on another page). In this case, the dynamic container rule set is generated so as to wire up the component correctly when the fragment is dropped onto a page or template.

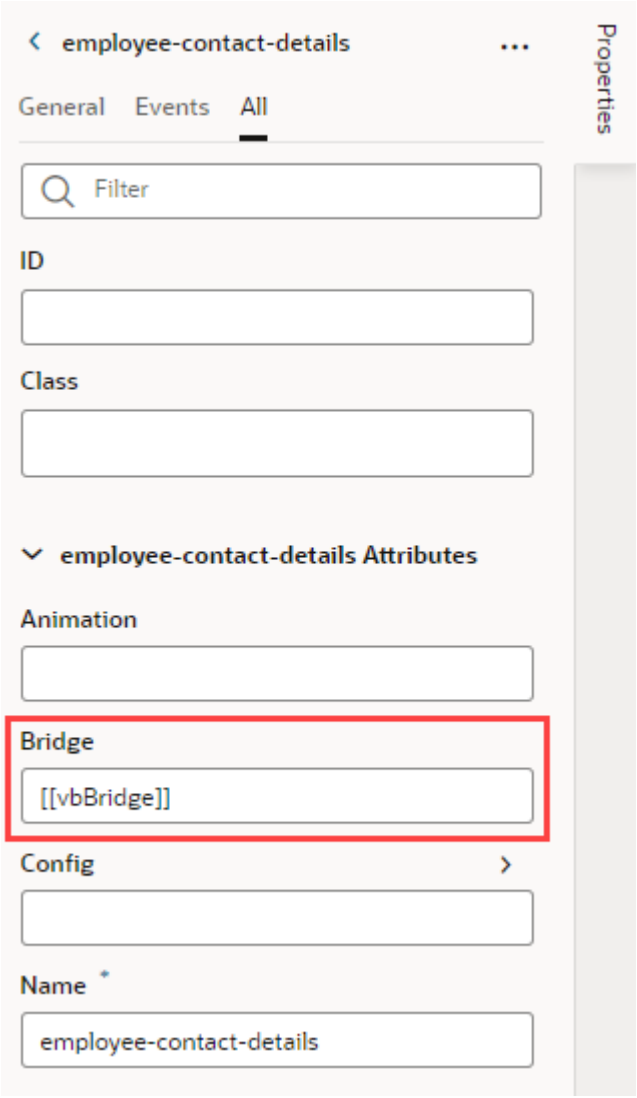
Subtype	valueOptions	Usage
Dynamic Layout Context	*none*	The Dynamic Layout Context option is typically not something you'd set manually. When you plan to use a fragment within a Dynamic Form's or Dynamic Table's field or form template and you tag it as `formTemplate` or `fieldTemplate` in the Settings editor, a new variable called `dynamicLayoutContext` is created automatically and marked as a required input parameter with its binding type set to this option. Because `dynamicLayoutContext` is an umbrella variable which contains all other layout-related context variables such as `$value`, `$metadata`, and so on, you'd be able to drop the fragment on a field or form template and gain access to the parent dynamic layout context through this variable.

Pass a Fragment's Context to VDOM or Custom Web Components

When you use fragments with VDOM (based on JET's Virtual DOM architecture) or custom web components, you need to allow the fragment to access its parent container's context. For example, you might have an `oj-dyn-form` (the VDOM variant of a dynamic form) that defines a form template in a fragment. To allow the fragment to access the layout context used to render the form template, you'll need to define the **Bridge** property on the fragment container.

The **Bridge** property is required within VDOM, particularly dynamic VDOM components that use fragments; it can also be used with custom web components.
To pass a fragment's context to a VDOM or custom web component:

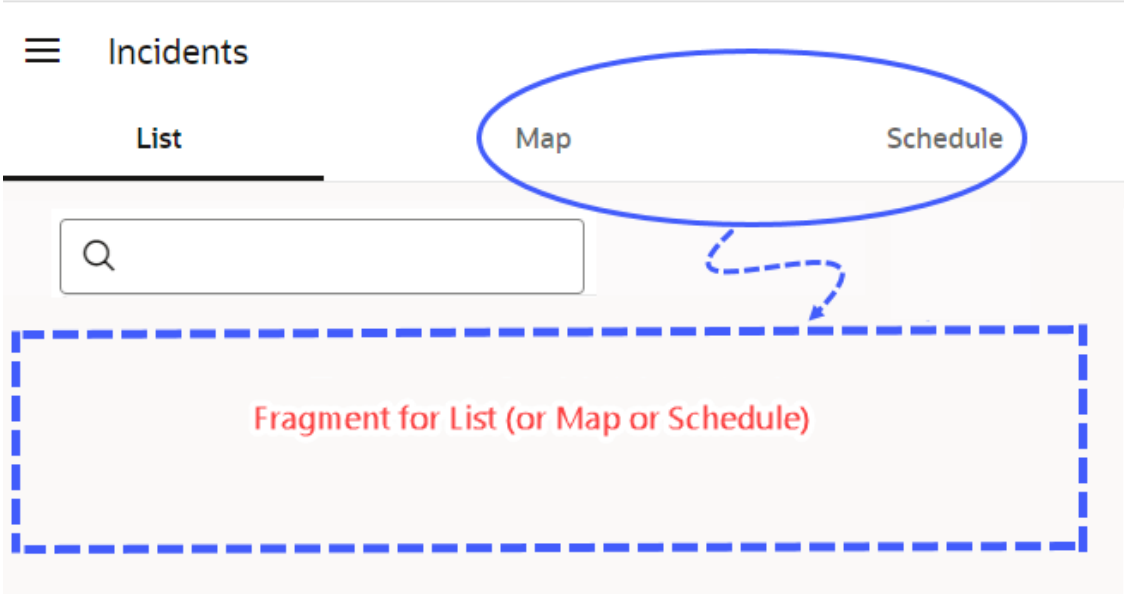
1. Open the page with the VDOM or custom web component that references a fragment.
2. In **Structure** view, select the fragment used on the page.
3. Click the **All** tab in the fragment container's Properties pane.
4. In the **Bridge** property, enter `[[vbBridge]]`.



Defer Rendering of a Fragment's Content

By default, a fragment loads immediately when its page renders, but you can change this behavior so a page renders faster initially. For example, say an Incidents page has three tabs —List, Map, and Schedule—all defined in separate fragments. When the Incidents page needs only the contents of the List fragment to display, you can wrap the Map and Schedule fragments in an `oj-defer` element to delay the rendering of those fragments at runtime.

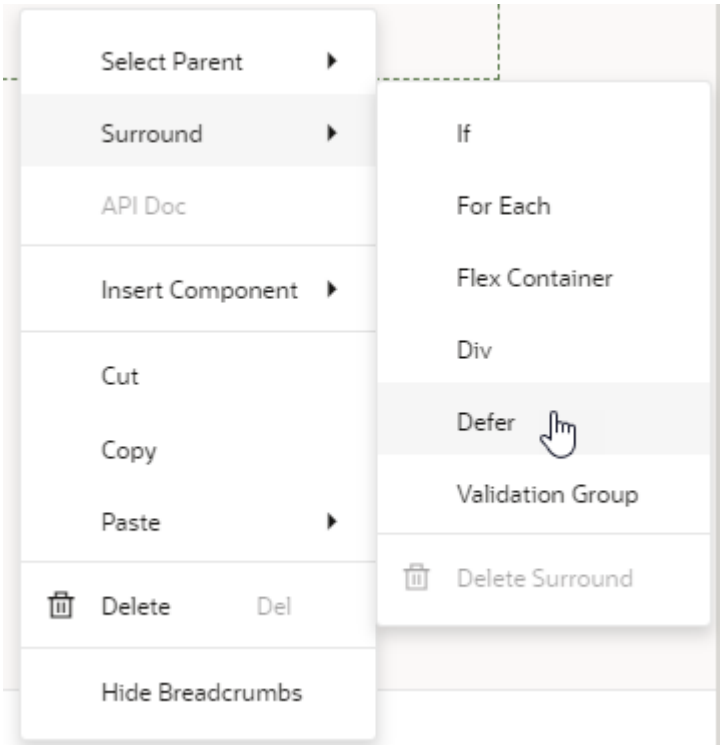
What triggers hidden fragments to render is configurable. It could be a button click, selecting a tab, opening a dialog, or an `oj-bind-if` that uses conditions to display content. In these cases, UI events or application state determines when the fragment is loaded. In the Incidents example, the hidden Map or Schedule fragment renders on the page only when a user clicks either tab to view its content:



You can also delay a fragment from rendering until it is "visible". Say, your page has a lot of content that encourages users to scroll. Rather than load the entire page, including sections hidden from the viewport, you might want to load some sections only when the user brings them into view. In this case, you can section your page into different fragments, then add a trigger to render a fragment only when it comes into view.

To set up a fragment for deferred rendering:

1. Open the web application's page that contains fragments.
2. Select the fragment container whose content you want to render later, either on the canvas or in the Structure view.
3. Right-click the container, select **Surround**, then **Defer**.



The Defer element is added to the fragment container, both on the canvas and in Structure view. If you click Code view, you'll see `oj-defer` surrounding `oj-vb-fragment`. Here's a code snippet for the Incidents tab bar with List, Map, and Schedule tabs, where everything except the first tab is hidden initially:

```
<oj-tab-bar selection="{{ $variables.incidents }}">
<ul>
  <li id="list">List</li>
  <li id="map">Map</li>
  <li id="Schedule">Schedule/li>
</ul>
</oj-tab-bar>
<oj-switcher value="[[ $variables.incidents ]]">
  <div slot="list">
      <oj-vb-fragment id="incidentslist" name="incidentsList"></oj-vb-
fragment>
  </div>
  <div slot="map">
    <oj-defer>
      <oj-vb-fragment id="incidentsmap" name="incidentsMap"></oj-vb-
fragment>
    </oj-defer>
  </div>
  <div slot="schedule">
    <oj-defer>
      <oj-vb-fragment id="incidentsschedule" name="incidentsSchedule"></oj-
vb-fragment>
    </oj-defer>
  </div>
</oj-switcher>
```

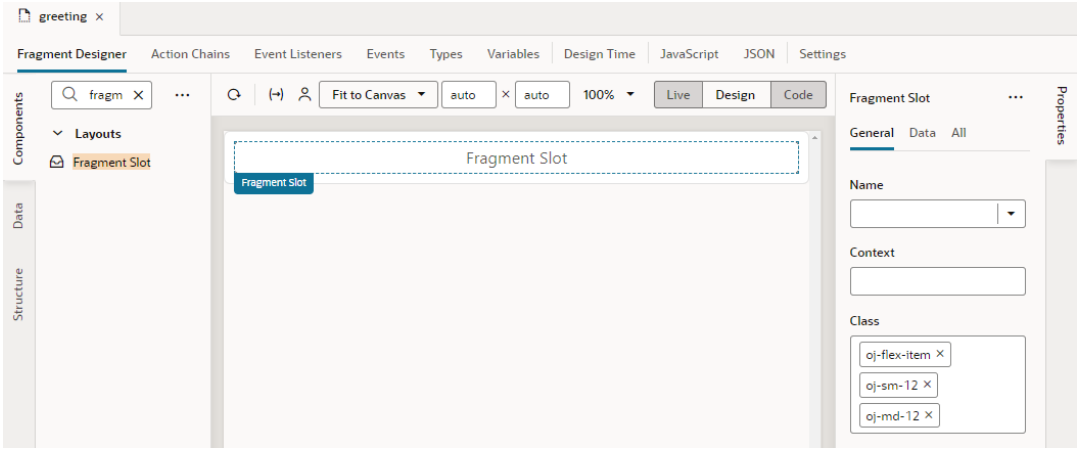
Add Slots to a Fragment

As a fragment author, you can add one or more slots to your fragment as placeholders, so those who consume the fragment can drop in their own components or content. Let's say you want a greeting area for users to add their own content. To do this, you'd define a slot where your fragment's consumers can add whatever they want, be it text or images.

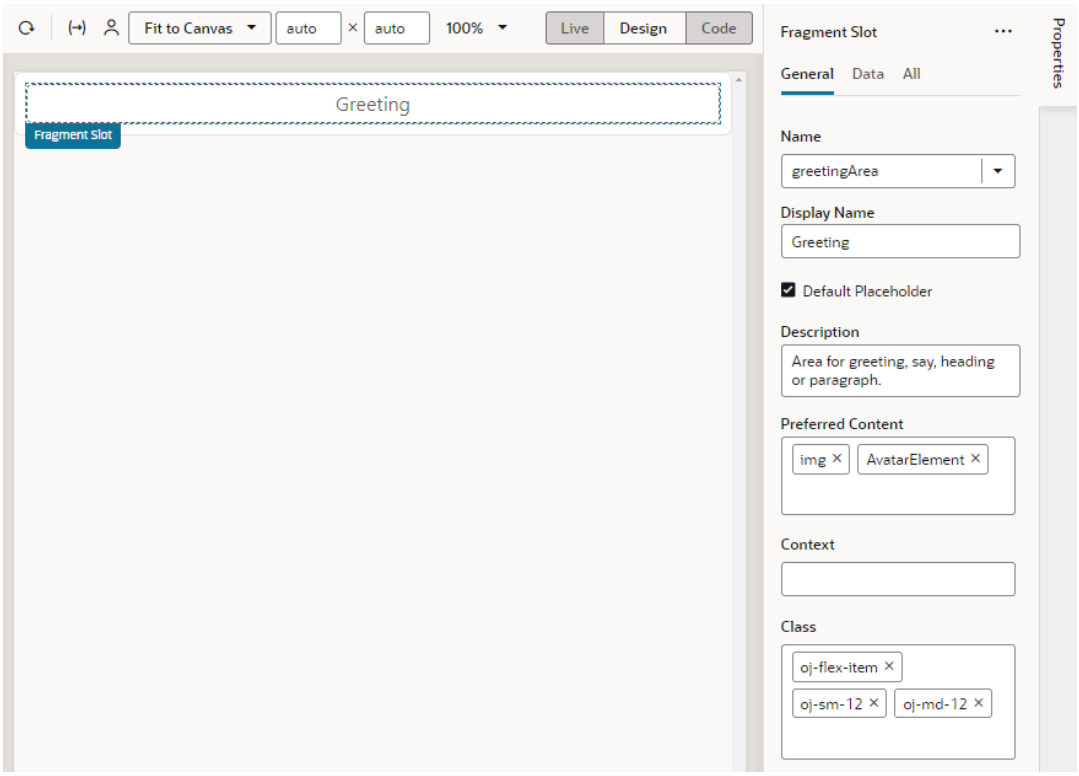
Fragment slots are similar to component slots. They can be used—or left unused—just like component slots. The only difference is that fragment slots cannot have a "default" slot.

To add slots to a fragment:

1. Open your fragment in the Fragment Designer. For demo purposes, let's assume you're working with the `greeting` fragment to define an area for your users to provide some greeting text on a page.
2. From the Components palette, drag and drop a Fragment Slot onto the fragment.



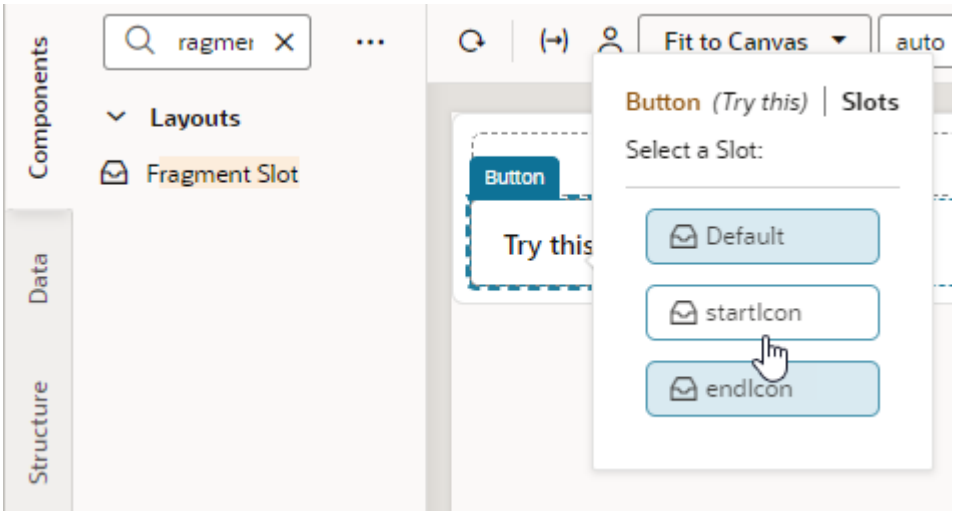
3. In the Fragment Slot's Properties pane, enter a slot name in the **Name** property (for example, `greetingArea`).
4. Define other properties as needed:
   a. To use a more descriptive identifier instead of the slot name, enter a **Display Name** that will appear in the slot's placeholder area as well as wherever the slot name is shown.
   b. To provide a visual cue to users that something can be dropped into the area, select **Default Placeholder** to generate a placeholder for the slot based on its name or display name.
   c. To let fragment users know what the fragment slot is meant for, provide a **Description**.
   d. To indicate the type of content that the slot can contain, select from the **Preferred Content** list. For example, if you expect the slot to hold image elements, you might search and select the Avatar and Image components.
   e. To allow fragment users to see contextual data in the custom slot component, set up the **Context**.



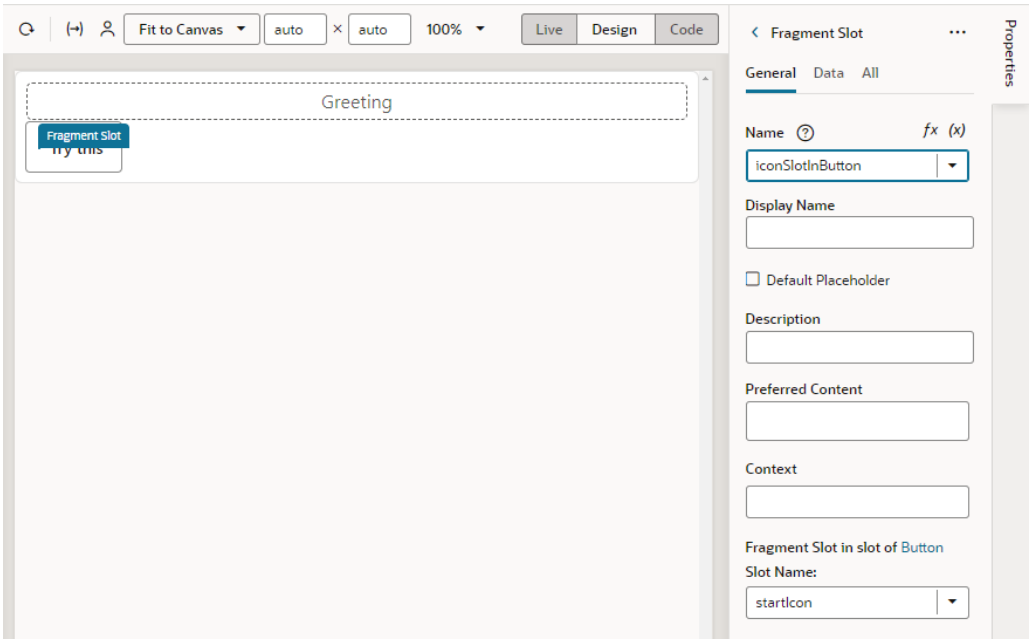
5. It's also possible to add a fragment slot to a slot inside another component (for example, a slot inside a button), allowing fragment users to customize those slots in the fragment. To do this:

   a. Drag and drop a Button onto the fragment canvas, then set it up as desired.

   b. Drag a Fragment Slot and drop it onto the Button, then select a slot in the button (for example, `startIcon`):

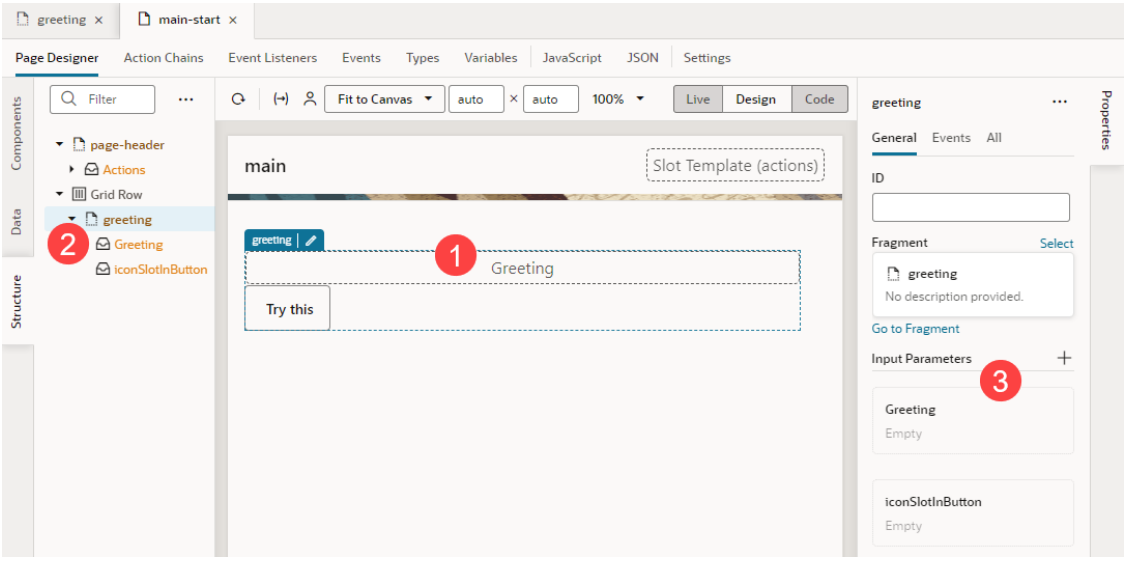


   c. Enter a name for the fragment slot in the button's slot (for example, `iconSlotInButton`); optionally, define other properties:



> **Note:**
>
> As a best practice, fragment authors should provide default content for any slots they define, just in case the user does not provide their own. See Add Default Content to a Fragment Slot.

Now when the fragment is used on a page, it reveals its slots (`greeting` and `iconSlotInButton` in our example) on the page canvas (label 1), the page structure (label 2, shown here with **Show Slots** selected), and the fragment's properties (label 3):

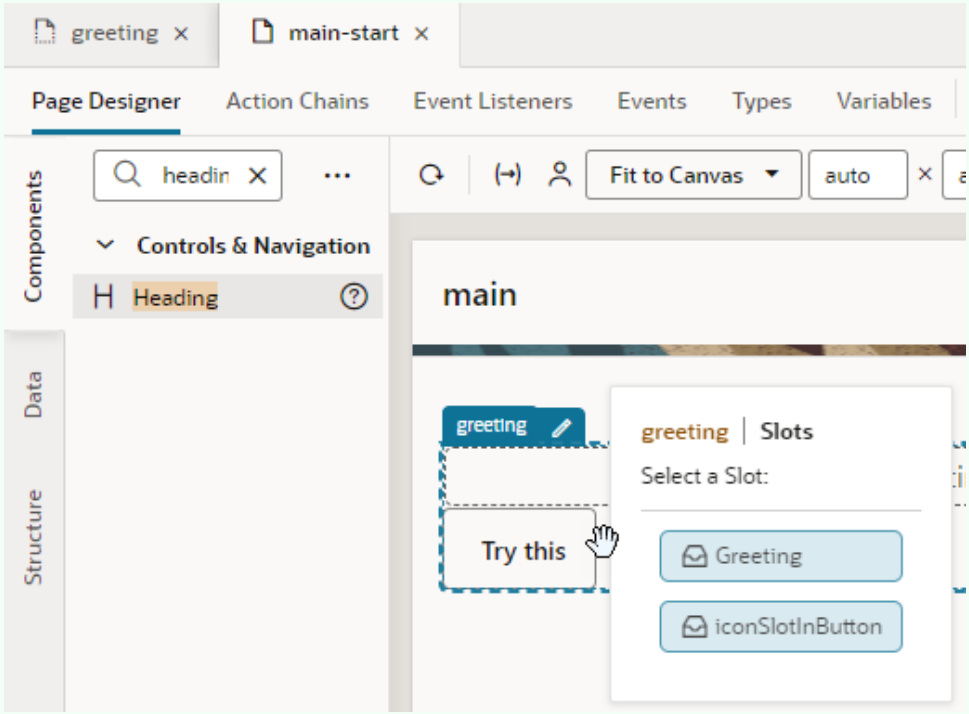


As a fragment consumer, you can now add the component of your choice to the slots revealed in the fragment. For demo purposes, let continue our greeting example and add a heading to the fragment slot on the canvas.

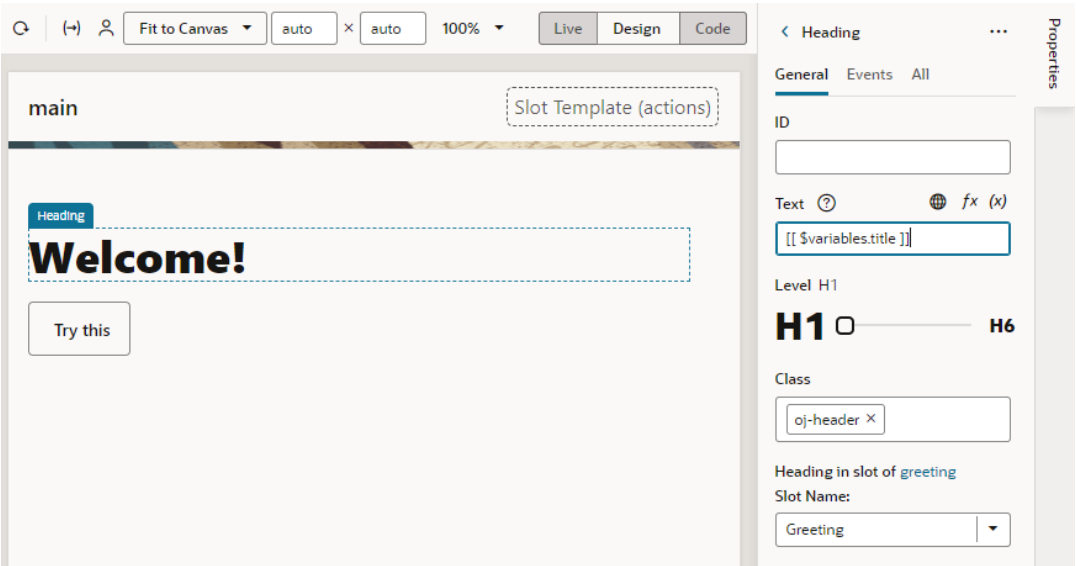
1. Drag and drop a Heading onto the Fragment Slot (`greeting`, for example).

Tip:

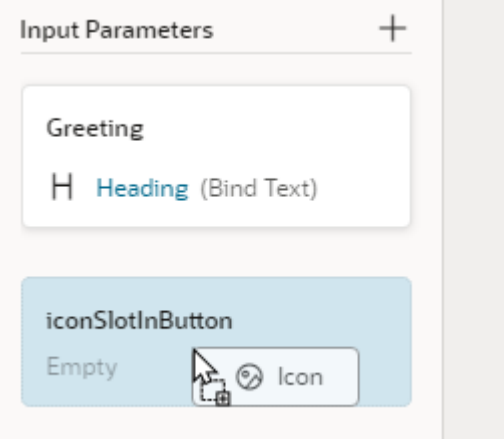
You can add components to a fragment slot on the canvas and in Structure view just as you would component slots. For example, when you drag a Heading component and drop it directly onto the *fragment* in the canvas, you'll be prompted to select a slot declared in your fragment. This way, you'll be able to drop content into slots that don't include a default placeholder:



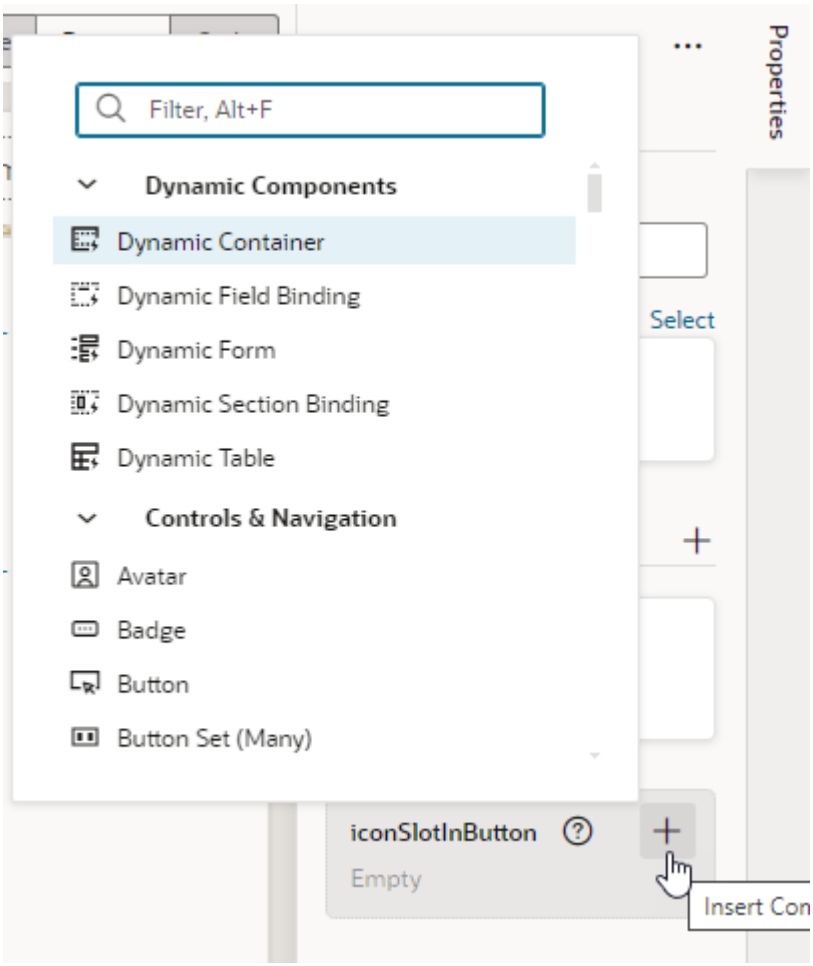
2. Update the slotted component's properties as needed. For example, you might bind the Heading's text to a page-level variable to display your greeting:



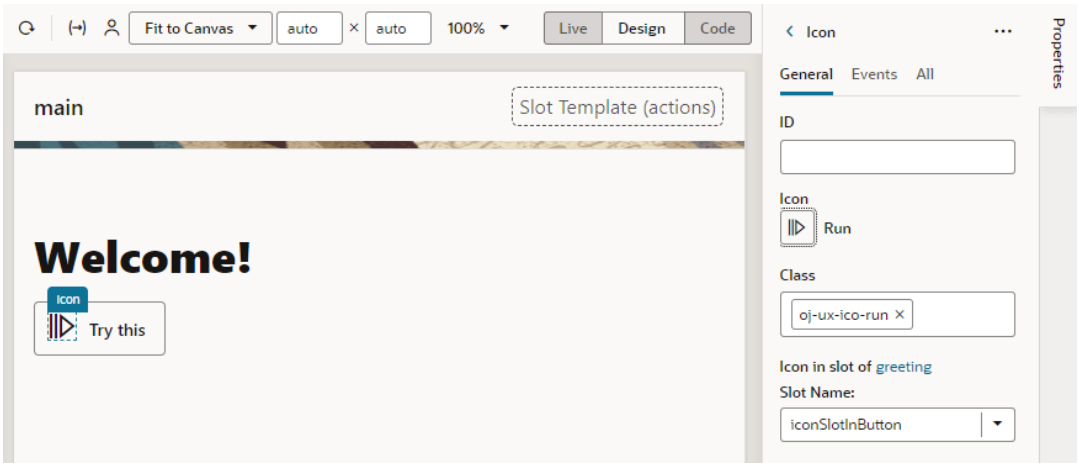
3. To customize the fragment slot in the slot inside the button, select the fragment to view the fragment's Properties pane, then simply drag and drop a component of choice onto the fragment slot. For example, drag an Icon from the Components palette onto the `iconSlotInButton`.



Alternatively, hover over the `iconSlotInButton` slot in the fragment's Properties pane and click the **Insert Component** icon ( ). Components marked as Preferred Components for the slot show in this view. Select a preferred component or any other component of your choice.



4. After you've dropped the icon to the fragment slot, you can select the icon to further customize it:



Add Default Content to a Fragment Slot

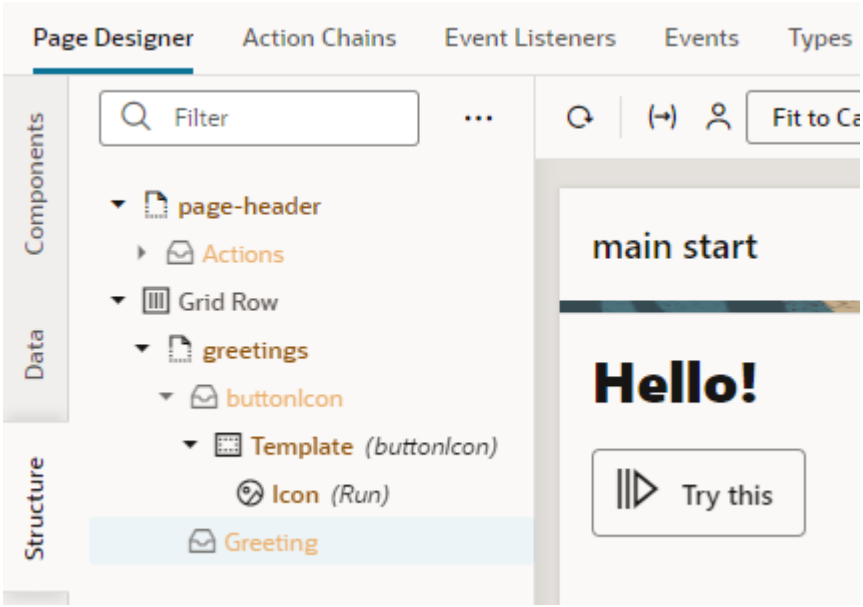
As a best practice, it's recommended that you provide some default content for a fragment slot in case the fragment user doesn't provide their own content. To do this, you add the required elements within a `<template>` in your fragment slot's HTML.

For example, you can default to a standard greeting of `Hello!` in the `greetingArea` slot in case the fragment user does not specify something themselves. To do this:

1. In the Fragment Designer, select the fragment slot in the Structure view, then click **Code** to view the fragment's HTML source.
2. Add your default content wrapped in a `<template>` element to the fragment slot's definition within `<oj-vb-fragment-slot>`. For example:

```
<oj-vb-fragment-slot bridge="[[vbBridge]]" class="oj-flex-item oj-sm-12 oj-
md-12" name="greetingArea">
  <template>
    <h3>Hello!</h3>
  </template>
</oj-vb-fragment-slot>
```

Now when this fragment is used on a page, here's what it looks like:



Set Data Context for a Fragment Slot

Because fragment slots are similar to component slots, they rely on the idea of template slots for data context in much the same way as component slots. Template slots allow components

in a slot to access data from the component's surrounding page model as well as from within the custom component. Here's how we can break down this concept:

What is Data Context?

Fragment slots, by design, allow fragment authors to leave placeholders for users to customize with their own content. A typical example is a toolbar used within a fragment view that provides a set of default buttons, but which also supports a slot for fragment users to add some of their own. In this case, slot content is supplied by the user but "managed" from a layout perspective by the fragment (or a component used within it).

Often, fragment authors use components like `oj-table`, `oj-list-view`, or `oj-bind-for-each`, which stamp a list of rows where the data for each row is supplied by the fragment. A fragment author who wants the user to have the ability to shape how each row should be rendered as it is stamped out, may also provide access to (current row/item) data that the stamping component sets up. For example, an `oj-list-view` supports an `"itemTemplate"` slot, which provides contextual item data via a standard variable called `$current`.

> **Tip:**
>
> The `data-oj-as` attribute can be used to define an alias for `$current`. If `data-oj-as` is defined, then the alias is used instead of `$current`.

List View Example

Here's an example of an `oj-list-view` used in a fragment to display a list of user names and roles. To define the shape of the data (user names and roles) that the component (`oj-list-view`) will expose through to the slot via `$current` (or the assigned `data-oj-as` name), you'd first define the `data` attribute in the fragment's metadata:

```
"slots": {
  "userRowTemplate": {
    "data": {
      "$current": {
        "type": "object",
        "properties": {
          "data": {
            "type": "object",
            "properties": {
              "name": {
                "type": "string"
              },
              "role": {
                "type": "string"
              }
            }
          }
        }
      }
    }
  }
}
```

You might then implement the fragment view as shown here:

```
<oj-list-view data="[[ $variables.userArrayDataProvider ]]">
  <template slot="itemTemplate">
    <oj-vb-fragment-slot bridge="[[vbBridge]]" name="userRowTemplate"
context="[[ { $current: $current } ]]">
      <!-- Default template used if fragment consumers do not provide their
own. -->
      <template>
        <p>
          <oj-bind-text value="[[$current.data.name]]"></oj-bind-text>
          <oj-bind-text value="[[$current.data.role]]"></oj-bind-text>
        </p>
      </template>
    </oj-vb-fragment-slot>
  </template>
 </oj-list-view>
```

In this example, the template slot `"itemTemplate"` contains a fragment slot that provides the current row's data via the `$current` variable. The fragment slot's default template displays the user name and role as simple text (`oj-bind-text`). Note that the value for `"name"` and `"role"` is provided by the fragment, or more accurately the stamping component used within the fragment, not the surrounding page model.

Now when the fragment is used on a page, fragment users by default see a list of user names and user roles in text and can choose to customize how the list items render, for example:

```
<h1>Users</h1>
 <oj-vb-fragment bridge="[[vbBridge]]" name="user-list">
  <!-- Custom template for row with user data. -->
  <template slot="userRowTemplate">
    <h2 class="oj-header">
      <oj-bind-text value="[[ $current.data.name ]]"></oj-bind-text>
    </h2>
    <div>
      <oj-bind-text value="[[ $current.data.role ]]"></oj-bind-text>
    </div>
  </template>
 </oj-vb-fragment>
```

Table Example

An `oj-table` supports a template slot `"rowTemplate"` that provides contextual row data via `$current` (or the assigned `data-oj-as` name). Here's an example of an `oj-table` in a fragment:

```
<oj-vb-fragment name="products" bridge="[[vbBridge ]]">
  <template slot="productRowTemplateSlot" data-oj-as="productRow">
        <oj-bind-text value='[[ productRow.data[ "name" ] + ":" +
productRow.data[ "code" ] ]]'></oj-bind-text>
  </template>
</oj-vb-fragment>
```

In this example, the `data-oj-as` attribute on the `template` tag defines an alias for `$current`, specifying that the template slot's data can be accessed through the `productRow` variable. If an alias is not specified, then `$current` can be used.

Customize How Fragment Properties Display in the Properties Pane

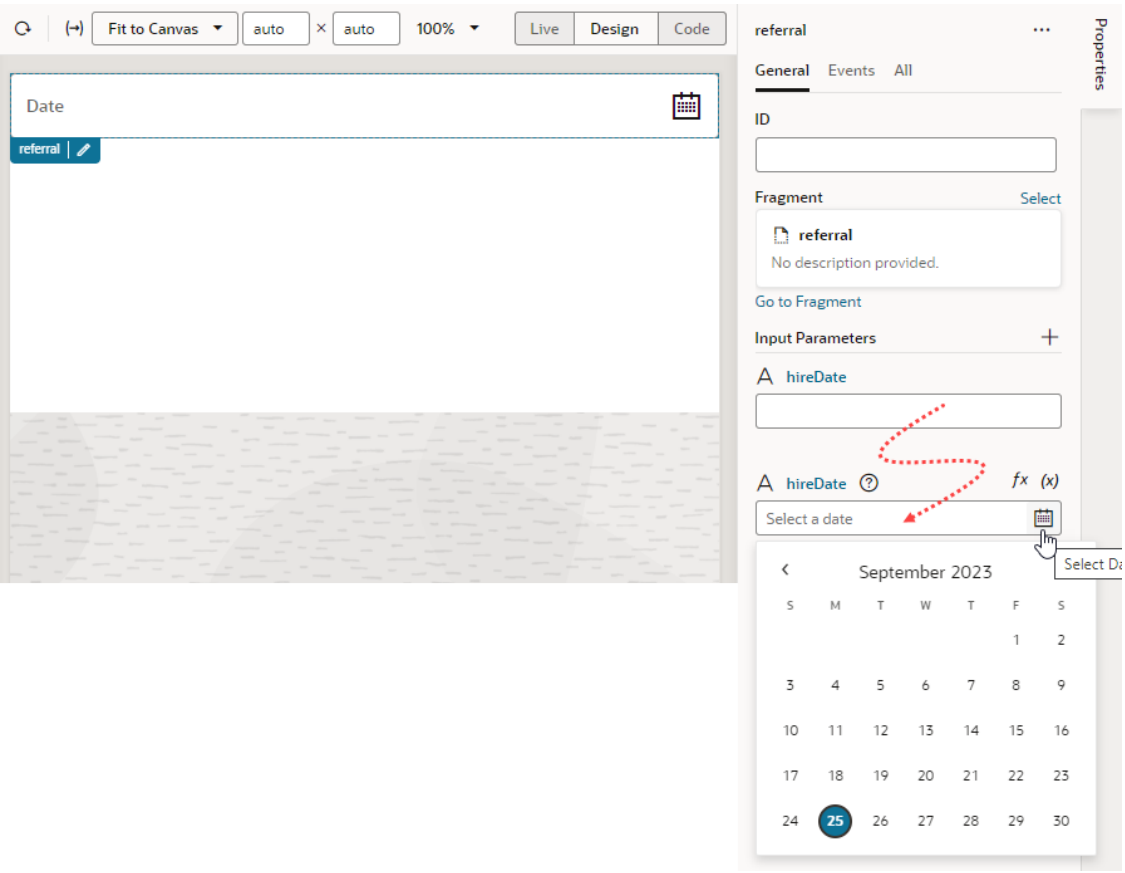
Customize how a fragment's properties display in the Properties pane as a way to enhance the design experience when fragments are used. Here are the customization options available to you:

Option	See
Display an enhanced UI for each fragment input parameter in the fragment's Properties pane when a fragment is selected on a page.	Customize How a Fragment Variable is Displayed in the Properties Pane
Display a fragment's input parameters and other properties in sections on the fragment's Properties pane. If the fragment is used as a page template, this sectioned view also shows on the page's properties pane (which displays when no component or element is selected on the page).	Section Fragment Properties for Display in the Properties Pane

Customize How a Fragment Variable is Displayed in the Properties Pane

Customize the UI component displayed for fragment input parameters in the Properties pane, which can make the task of editing those parameters in the Page Designer easier.

When working with a fragment in the Page Designer, the fragment's Properties pane by default displays text field components for editing the values of fragment variables enabled as input parameters. For some input parameters, a different UI component can make editing the parameter easier or more intuitive. For example, if a parameter is used to specify a date, a Date Picker component might be easier to use than a text field. To do this, you customize the fragment variable, so that a date picker shows instead of a text field when the fragment's input parameters are edited in the Page Designer:



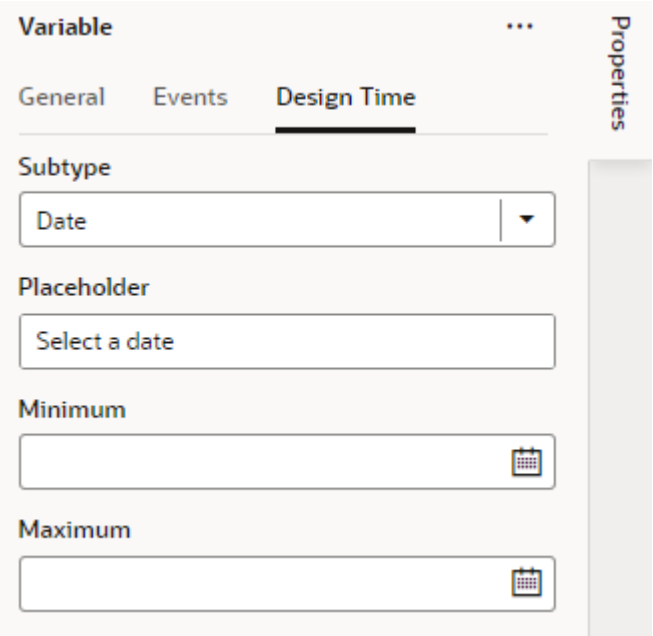
To customize the UI component displayed for a fragment variable in the Properties pane, you use the Design Time tab in the Variables editor. You can also edit the fragment's JSON directly in the JSON editor.

> **Note:**
>
> Some UI customization options are not available in the Design Time tab. You'll need to edit the JSON directly to configure these advanced options. See Customize Fragment JSON with Metadata for a list of options.

Customize a Variable in the Variables Editor

To configure the UI used to edit a fragment's input parameters in the Page Designer:

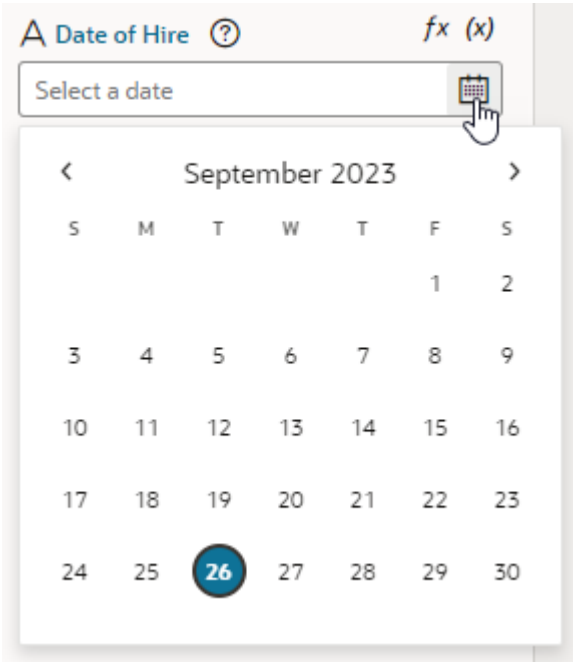
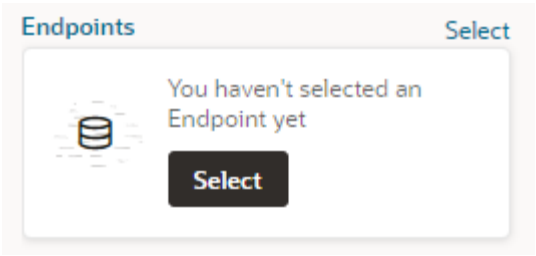
1. Open the fragment's Variables editor.
2. Select the variable or constant you want to customize.
3. Open the variable or constant's **Design Time** tab in the Properties pane.
4. Select properties to customize how the component for editing the variable will look in the Page Designer.
   The properties you see in the Design Time tab will depend upon the variable's type, and the Subtype property you select in the tab. For example, if Date is selected as the Subtype for a string-type variable, you'll see fields for setting the date's Minimum and Maximum limits:

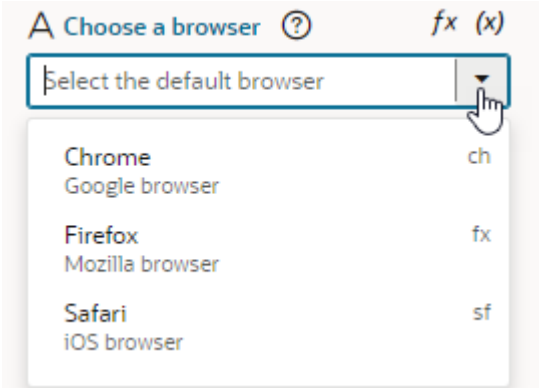


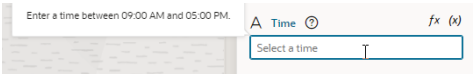
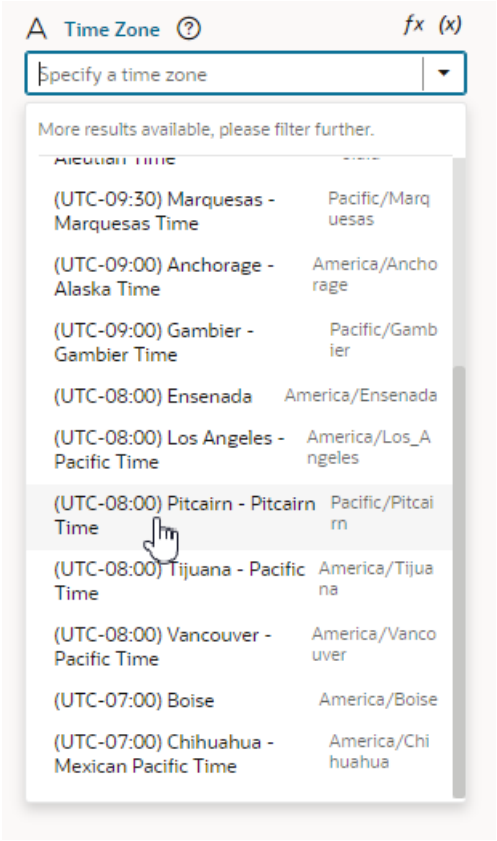
Here are steps for some common customization options for string-, object-, and number-type variables and constants:

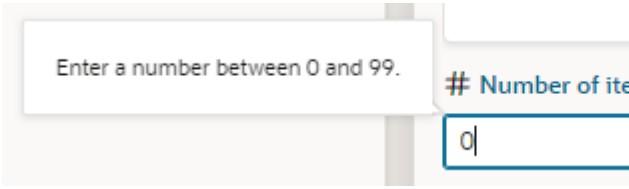
> **Note:**
>
> When working with string variables and constants, you have the option of adding translation metadata if you want a particular variable (or constant) to display the translation icon on the Page Designer's Properties pane to support translation.

Customization Option	Steps in Design Time Tab	Result in Page Designer
To display a color picker:	For a string-type variable or constant: **a.** In the **Design Time** tab, select the **Subtype** as **Color**. **b.** Optional: In the **Placeholder** field, specify a hint text for the variable; for example, `Choose a color`. **c.** Optional: Switch to the **General** tab and set these additional properties: • In the **Label** field, enter a user-friendly name for the variable. • In the **Default Value** property, use the color picker to set a default color.	

Customization Option	Steps in Design Time Tab	Result in Page Designer
To display a date or date-and-time picker:	For a string-type variable or constant: **a.** In the **Design Time** tab, select the **Subtype** as **Date** or **Date Time**. **b.** Optional: In the **Placeholder** field, specify a hint text for the variable; for example, `Select a date`. **c.** Optional: In the **Minimum** field, set the bottom (inclusive) limit of a date or date-and-time range for the value in the Properties pane. **d.** Optional: In the **Maximum** field, set the top (inclusive) limit of a date or date-and-time range for the value in the Properties pane. **e.** Optional: Switch to the **General** tab, then in the **Label** field, enter a user-friendly name for the variable.	
To display an endpoint picker:	For an object-type variable or constant: **a.** In the **Design Time** tab, select the **Subtype** as **Endpoint**. **b.** Optional: To filter endpoints available in the endpoint picker by REST action type, for example, to only list `Get One` REST calls, select one or more of the predefined filters in **Endpoint Action Hint**. **c.** Optional: To filter endpoints available in the endpoint picker by service connection type, for example, to only list service connections using an ADF Describe, select one or more of the predefined filters in **Service Type**. **d.** Optional: Switch to the **General** tab, then in the **Label** field, enter a user-friendly name for the variable. **Note:** The **Placeholder** field does not take effect in the Properties pane when you use the Endpoint subtype.	 Clicking **Select** launches a Configure Endpoint wizard in which fragment users can select a suitable endpoint and choose its URI parameters. **Tip:** If you cannot find the endpoint you want or prefer to manually set up your endpoint, click the Manual Setup of Endpoint icon ( ) in the wizard, then select from the available endpoints and confgure its URI parameters.

Customization Option	Steps in Design Time Tab	Result in Page Designer
To display a drop-down menu containing an array of possible values:	For a string-type variable or constant: **a.** In the **Design Time** tab, select the **Subtype** as **Enum Values**. **b.** Optional: In the **Placeholder** field, specify a hint text for the variable; for example, `Select the default browser`. **c.** Click + next to Enum Values, enter the **Label**, **Value**, and **Description** for your first value. For example, you might enter `Chrome` as the label, `ch` as the value, and `Google Browser` as the description. Click **Create**. If you want to make changes, click ✎, update the values, and click **Save**. Click 🗑 to delete a value. To reorder your list, drag the ⋮ next to the value and drop it where you want it. **d.** Repeat step c to create your entire list of values. **e.** Optional: Switch to the **General** tab, then in the **Label** field, enter a user-friendly name for the variable.	

Customization Option	Steps in Design Time Tab	Result in Page Designer
To display a time picker:	For a string-type variable or constant: **a.** In the **Design Time** tab, select the **Subtype** as **Time**. **b.** Optional: In the **Placeholder** field, specify a hint text for the variable; for example, `Select a time`. **c.** Optional: In the **Minimum** property, set the bottom (inclusive) limit of a time range for the value in the Properties pane. **d.** Optional: In the **Maximum** property, set the top (inclusive) limit of a time range for the value in the Properties pane. **e.** Optional: Switch to the **General** tab, then in the **Label** field, enter a user-friendly name for the variable.	
To display a drop-down menu with a list of time zones:	For a string-type variable or constant: **a.** In the **Design Time** tab, select the **Subtype** as **Time Zone**. **b.** Optional: In the **Placeholder** field, specify a hint text for the variable; for example, `Select a time zone`. **c.** Optional: Switch to the **General** tab, then in the **Label** field, enter a user-friendly name for the variable.	

Customization Option	Steps in Design Time Tab	Result in Page Designer
To limit the input values to a number in a range:	For a number-type variable or constant: a. In the **Design Time** tab, specify a hint text for the variable, for example, `Enter Quantity`, in the **Placeholder** field. b. Optional: In the **Minimum** and **Maximum** properties, set the inclusive bottom and top limits of a range for the value in the Properties pane; for example, to limit the input value to a number in the range 0 - 99. c. Optional: Switch to the **General** tab, then in the **Label** field, enter a user-friendly name for the variable.	

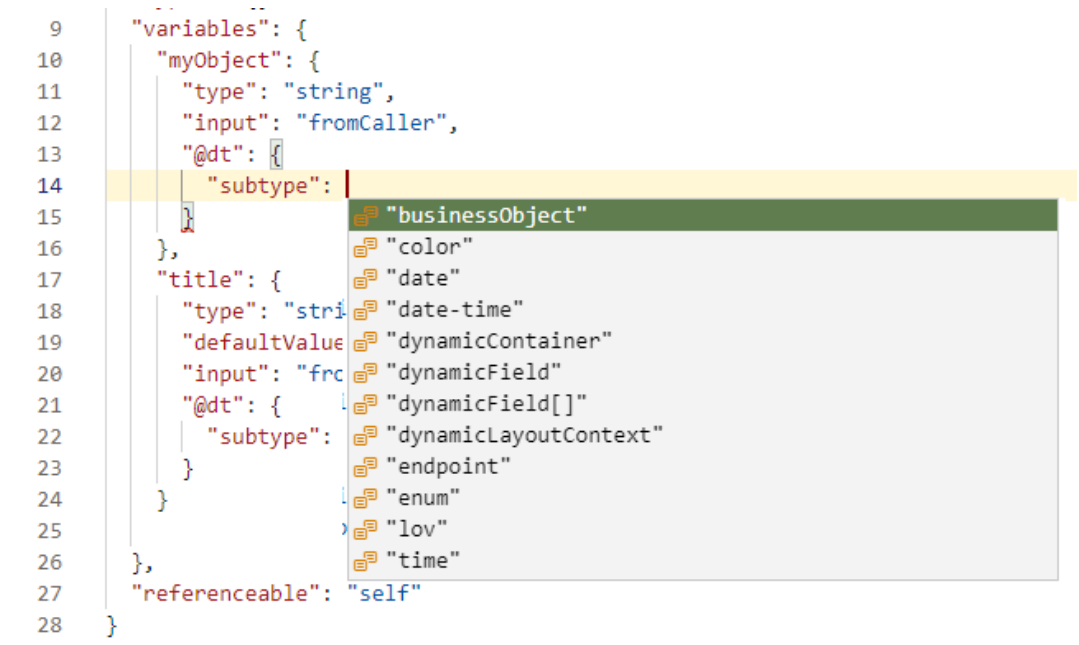
When you set properties in the Design Time tab, the metadata in the fragment's JSON is automatically updated. You can open the JSON editor to view the metadata. For example, here's what you might see for a variable that is customized to use the Date Picker component:

```
"variables": {
    "hireDate": {
      "type": "string",
      "input": "fromCaller",
   "@dt": {
        "label": "Date of Hire"
        "subtype": "date",
        "valueOptions": {
          "placeholder": "Select a date"
     }
   }
},
```

Customize Fragment JSON with Metadata

While you can use a fragment variable's Design Time tab for some simple UI customization, you'll need to edit the JSON directly for advanced options. To do this:

1. Open the fragment's JSON editor.
2. Update the variable or constant's definition by setting the `@dt` element, then use the `subtype` property to specify the component you want displayed in the Page Designer. The JSON editor displays a hint to help you select the value for the `subtype` property:

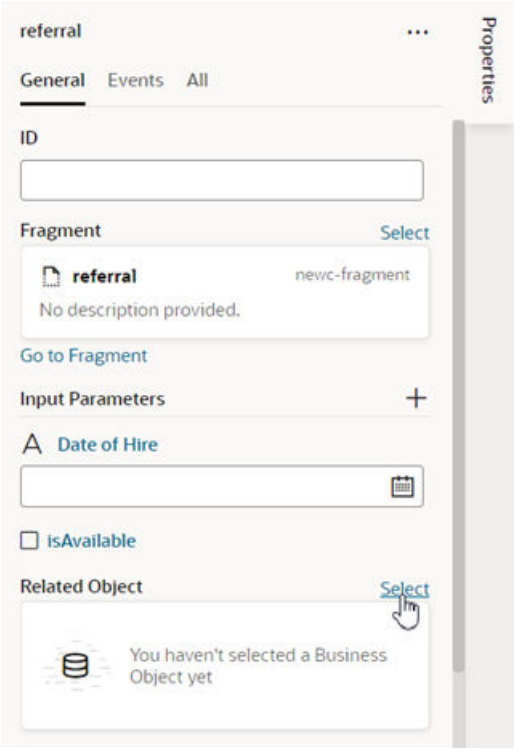


For example, here's how you can show a component for selecting a business object by setting the `subtype` property to `businessObject`:

```
"myObject": {
   "type": "string",
   "input": "fromCaller",
   "@dt": {
         "subtype": "businessObject",
         "label": "Related Object"
    }
 },
```

You can also use the `label` property to change the variable's display name in the Properties pane.

The Properties pane in the Page Designer will now show a component for selecting a business object and the new display name for the variable wherever the fragment is used.

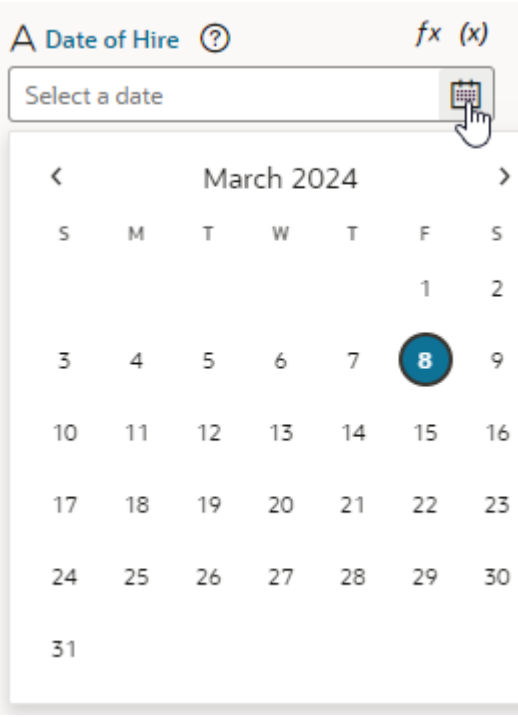


For more details about the JET components and properties, see JET metadata in Oracle JavaScript Extension Toolkit documentation.

Property Options for Variable Metadata

The following table describes the metadata properties that you can use in JSON to customize how fragment variables are displayed in the Properties pane:

Property	Type	Description
`label`	string	Use this property to specify a user-friendly name for the variable.

Property	Type	Description
`subtype`	Available subtypes: • `businessObject` • `color` • `date` • `date-time` • `endpoint` • `enum` • `lov` • `time` • `timezone`	Use this property to create a more specific type of customizer for simple types. For example, you can choose `date` to use a date picker component for a string type:  Most simple customizations can be configured from the **Design Time** tab (see Customize a Variable in the Variables Editor).

Note:

The following subtypes are also available, but they are used to set the binding type, and do not affect how the variables are displayed in the Propert

Property	Type	Description
		ies pan e: • dynamicContainer • dynamicFieldd • dynamicFieldd[] • dynamicL

Property	Type	Description
		`layoutContext` For details, see Set the Binding Type for Variables in Dynamic Components.
`valueOptions`	object	The `valueOptions` available to you depend on the selected subtype. When no subtype is selected, the only `valueOptions` is `placeholder`. See the tables below for a list of `valueOptions` properties.

Property Options for Fragment Metadata

The valid `subtype` and `valueOptions` properties you can use depend on the variable's type. For example, the `color` subtype can only be applied to variables that are strings. This section describes the `subtype` and `valueOptions` properties that are valid for each type.

Subtypes and valueOptions for objects

When the variable type is object, the following table describes the `subtype` and `valueOptions` that can be used:

Subtype	valueOptions	Usage

dynamicContainer	section	The only valueOption for the dynamicContainer subtype is section:
		
		"@dt": {
		"subtype": "dynamicContainer",
		"valueOptions": {
		"section": {
		"preferredContent": ["SpFoldoutPanelElement","SpFoldoutPanelSummarizingElement"]
		}
		}
		}
		
		You create the metadata for dynamic container sections in the parent container.
		Use preferredContent to list the interfaces that components in the root of the section templates must implement.

Subtype	valueOptions	Usage
empty	`fields`	When no subtype is selected, you can use the `fields` `valueOption` to customize the display/editing of object values. Instead of displaying a simple single text area for the whole value, the Properties pane will display individual customizers for the various fields of the object.

> **Note:**
>
> This property is only supported for displaying the first level of object fields.

You can specify an array of fields of the associated variable or constant that you want displayed in-line in the Properties pane when editing the object's values. You can customize how each field is displayed by using `label`, `description`, `subtype`, and `valueOptions`.

When using the `fields` property, each field **must** have the ID of the object field it maps to. The order of fields in the array is the order they will be displayed in the Properties pane.

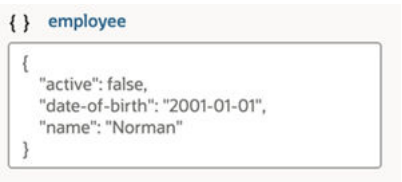
You can use the following field properties:

- `id` (Required). A string to match the DT field definition to the ID of the object type.
- `label` (Optional). A string for the user displayable value for the field.
- `description` (Optional). A string to appear in the '?' help pop-up for the field.
- `subtype` (Optional). Use to further define the type of the field value. See the table above.
- `valueOptions` (Optional). Use these values to further customize the editing experience.

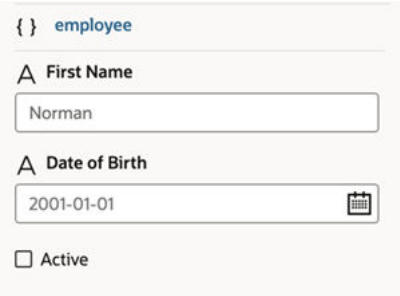
A variable described with the following metadata:

```
"variables": {
      "employee": {
        "type": "person",
        "input": "fromCaller",
        "defaultValue": {
          "active": false,
          "date-of-birth": "2001-01-01",
          "name": "Norman"
        }
      }
    }
```

would look similar to this in the Properties pane when displayed with the default text area:



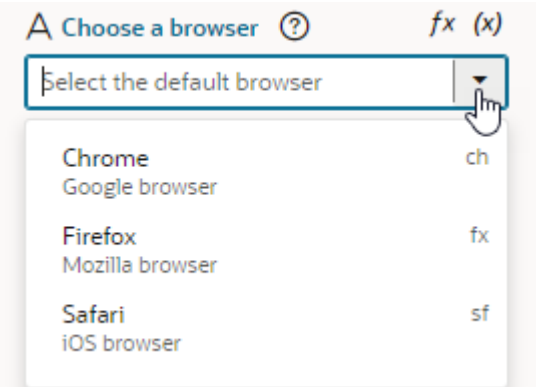
Subtype	valueOptions	Usage

		The `fields` property can be used to customize how the object is displayed:
		
		`"variables": {`
		`"employee": {`
		`"type": "person",`
		`"input": "fromCaller",`
		`"defaultValue": {`
		`"active": false,`
		`"date-of-birth": "2001-01-01",`
		`"name": "Norman"`
		`},`
		`"@dt": {`
		`"valueOptions": {`
		`"fields": [`
		`{`
		`"id": "name",`
		`"description": "The first (given) name"`
		`"label": "First Name"`
		`},`
		`{`
		`"id": "date-of-birth",`
		`"label": "Date of Birth"`
		`"subType": "date"`
		`},`
		`{`
		`"id": "active",`
		`"description": "Is the employee active?"`
		`"label": "Active"`
		`}`
		`]`
		`}`
		`}`
		`}`
		`}`
		
		The customized object would look similar to this in the Properties pane:
		
		 

Subtypes and valueOptions for arrays

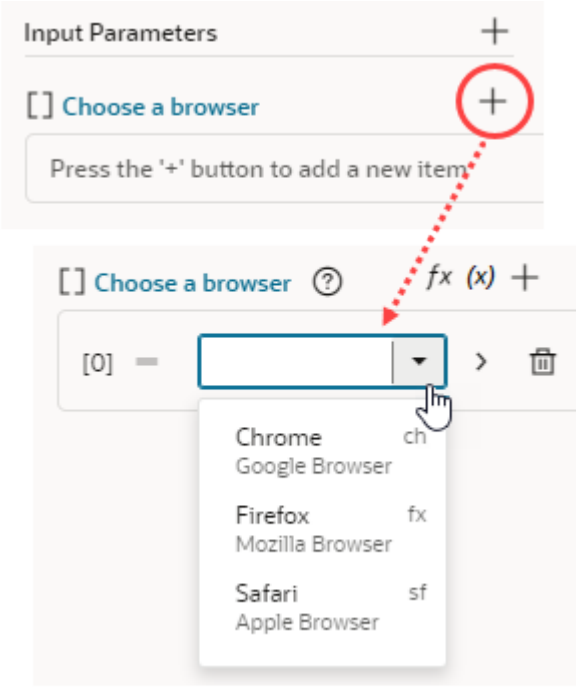
When the variable type is array, the following table describes the `subtype` and `valueOptions` that can be used:

Subtype	valueOptions	Usage

enum	values
	placeholder
		Use enum to display a drop-down menu showing all values for each item in the array. The only valueOptions property for the enum subtype is values. In addition to a value, each item in the array can have an optional label and description.
		
		"selectBrowser": {
		"type": "object[]",
		"input": "fromCaller",
		"@dt": {
		"subtype": "enum",
		"valueOptions": {
		"values": [
		{
		"value": "ch",
		"label": "Chrome",
		"description": "Google browser"
		},
		{
		"value": "fx",
		"label": "Firefox",
		"description": "Mozilla browser"
		},
		{
		"value": "sf",
		"label": "Safari",
		"description": "iOS browser"
		}
		],
		"placeholder": "Select the default browser"
		},
		"label": "Choose a browser"
		}
		},
		
		The Properties pane will show a drop-down menu that can have items with descriptions:
		
		 

Subtype	valueOptions	Usage

		If you are using an array of primitives (say, `string[]`), you can use `enum` to display a drop-down menu showing all values for each item in the array:
		
		`"selectBrowser": {`
		`"type": "string[]",`
		`"input": "fromCaller",`
		`"@dt": {`
		`"subtype": "enum",`
		`"valueOptions": {`
		`"values": [`
		`{`
		`"value": "ch",`
		`"label": "Chrome",`
		`"description": "Google Browser"`
		`},`
		`{`
		`"value": "fx",`
		`"label": "Firefox",`
		`"description": "Mozilla Browser"`
		`},`
		`{`
		`"value": "sf",`
		`"label": "Safari",`
		`"description": "Apple Browser"`
		`}`
		`],`
		`"placeholder": "Select the default browser"`
		`},`
		`"label": "Choose a browser"`
		`}`
		`},`
		
		The Properties pane will show a drop-down menu with the three values for each item in the array:

Subtype	valueOptions	Usage
		
empty	`placeholder`	When no subtype is selected, the only `valueOptions` is `placeholder`.

Subtypes and valueOptions for booleans

When the variable type is boolean, there are no `subtype` or `valueOptions`. Variables with a boolean type are displayed as switch components in the Properties pane.

Subtypes and valueOptions for numbers

When the variable type is number, you can use the Design Time tab to configure the `subtype` and `valueOptions`. See Customize a Variable in the Variables Editor.

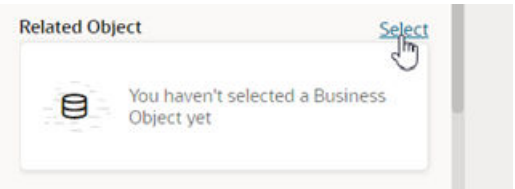
Subtypes and valueOptions for strings

When the variable type is string, you can use the Design Time tab to configure the `subtype` and `valueOptions` for simple customization options. See Customize a Variable in the Variables Editor.

The following table describes the `subtype` and `valueOptions` that can be used in JSON to configure advanced options for string-type variables:

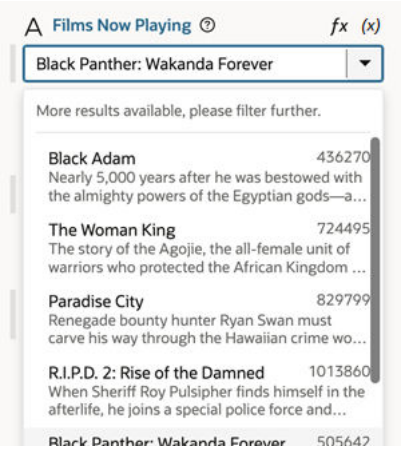
Subtype	valueOptions	Usage

empty	`placeholder`
	`translatable`
		Use the `placeholder` to specify a hint text for the variable. This can be used for all variables when there is no subtype.
		If a default value is supplied by the fragment variable, then that default value is used as the default placeholder. If both a placeholder value is used and the default value is specified, then the placeholder will be used.
		An example of values for the `placeholder` and `translatable` properties:
		`}`
		`"placeholder": "Search",`
		`"translatable": true`
		`}`

`businessObject`	`placeholder`	Use the `businessObject` subtype to display a business object picker in the Properties pane.
		`"type": "string",`
		`"input": "fromCaller",`
		`"@dt": {`
		`"subtype": "businessObject",`
		`"label": "Related Object"`
		`}`
		The Properties pane displays a component for selecting a business object:
		 
		
		Note:
		The `placeholder` `valueOption` is not displayed in the Properties pane when you use the `businessObject` subtype.

Subtype	valueOptions	Usage

`lov`	`service`
	`placeholder`
		Use to create a drop-down of values retrieved from a service, such as a REST endpoint.
		The service endpoint must be already set up in Visual Builder, and must be available to the application. You can then use `@dt` metadata to call the service and fetch items to populate the drop-down list.
		The service response must be in JSON format, and the response items in an array.
		In the example of the `lov` subtype below, the `now` constant will be offered a choice of values to pick from, which are determined by the response from a REST endpoint:
		
		`"now": {`
		`  "type": "string",`
		`  "description": "wow",`
		`  "defaultValue": "505642",`
		`  "input": "none",`
		`  "@dt": {`
		`  "label": "Films Now Playing",`
		`"subtype": "lov",`
		`"service": {`
		`"request": {`
		`"endpoint": "my-app:Petstore/getNowPlaying",`
		`"uriParameters": {`
		`"api_key": "4174b7d9a7b4bf87342c98e2289c6ee6"`
		`}`
		`},`
		`"response": {`
		`"itemsPath": "results",`
		`"mapping": {`
		`"value": "id",`
		`"label": "title",`
		`"description": "overview"`
		`}`
		`}`
		`}`
		`  }`
		`  }`
		`}`
		
		Here's how the example above displays in the Properties pane:

Subtype	valueOptions	Usage
		 For details about the `lov` metadata property values, see LOV Metadata Property Values.

LOV Metadata Property Values

You can assign the `lov` subtype to a variable if you want to display a drop-down list of values (LOV) for the variable in the Properties pane. To use the `lov` subtype, you'll need to set `valueOptions` property values to specify where the LOV data is retrieved from, and to configure how the drop-down list will look in the Properties pane:

Name	Description	Example
`service`	Type: Object Describes the service to retrieve the LOV data from, and how to use it.	See the `lov` subtype example above.
`request`	Type: Object Describes what service to call, and how to call it.	See the `lov` subtype example above.
`request.endpoint`	Type: string The fully-qualified name of a Visual Builder service that you are able to access.	`"my-app:Petstore/getNowPlaying"`
`request.pathParameters`	Type: Object Maps endpoint path parameter names, and the values to replace them with. The values can also be Visual Builder constants (see below).	`"pathParameters": { "name": "honeybadger" "department": "accounts" }`

Name	Description	Example
`request.uriParameters`	Type: Object Maps URI path parameters to the values they should be replaced with. The values can also be Visual Builder constants (see below).	`"uriParameters": { "api_key": "4174b7d9a7b4bf87342c98e2289c6ee6" "session_name": "cabbage" }`
`response`	Type: Object Describes how to unpack the payload returned by a successful response.	See the `lov` subtype example above.
`response.itemsPath`	Type: string A dot-separated path from the root of the response object to the array containing the LOV values.	`results`
`response.mapping`	Type: Object Describes how to populate the LOV from the response object. The mapping should indicate which response fields are to be used for the `label`, `value`, and `description` in the LOV.	See the `lov` subtype example above.
`response.mapping.description`	Type: string (optional) Describes the field from the response object that is used in the drop-down item description. It appears below `label` and `value`.	`overview`
`response.mapping.label`	Type: string (optional) Describes the field from the response object that is used as the primary display name of the item in the drop-down menu and in the input.	`title`
`response.mapping.value`	Type: string Describes the field from the response object that is used as the actual value of the variable/constant. It is visible to the right in the drop-down menu.	`id`

Using dependent parameters for `lov` metadata property values

The path and URI parameters might depend on other constants. For example, a REST service can use the result of an earlier selection as part of its own request. To do this, use expression notation in the parameter values to indicate which constant values to use:

```
"pathParameters": {
  "department": "[[ $constants.dept ]]"
}
```

The expression instructs this service request to use the current value of the "dept" constant as the value to use for the path parameter "department".

When writing the expression:

- Only simple direct references may be used. Calculated expressions such as `"[[ $constants.dept + "_" ]]"` will not work as expected.

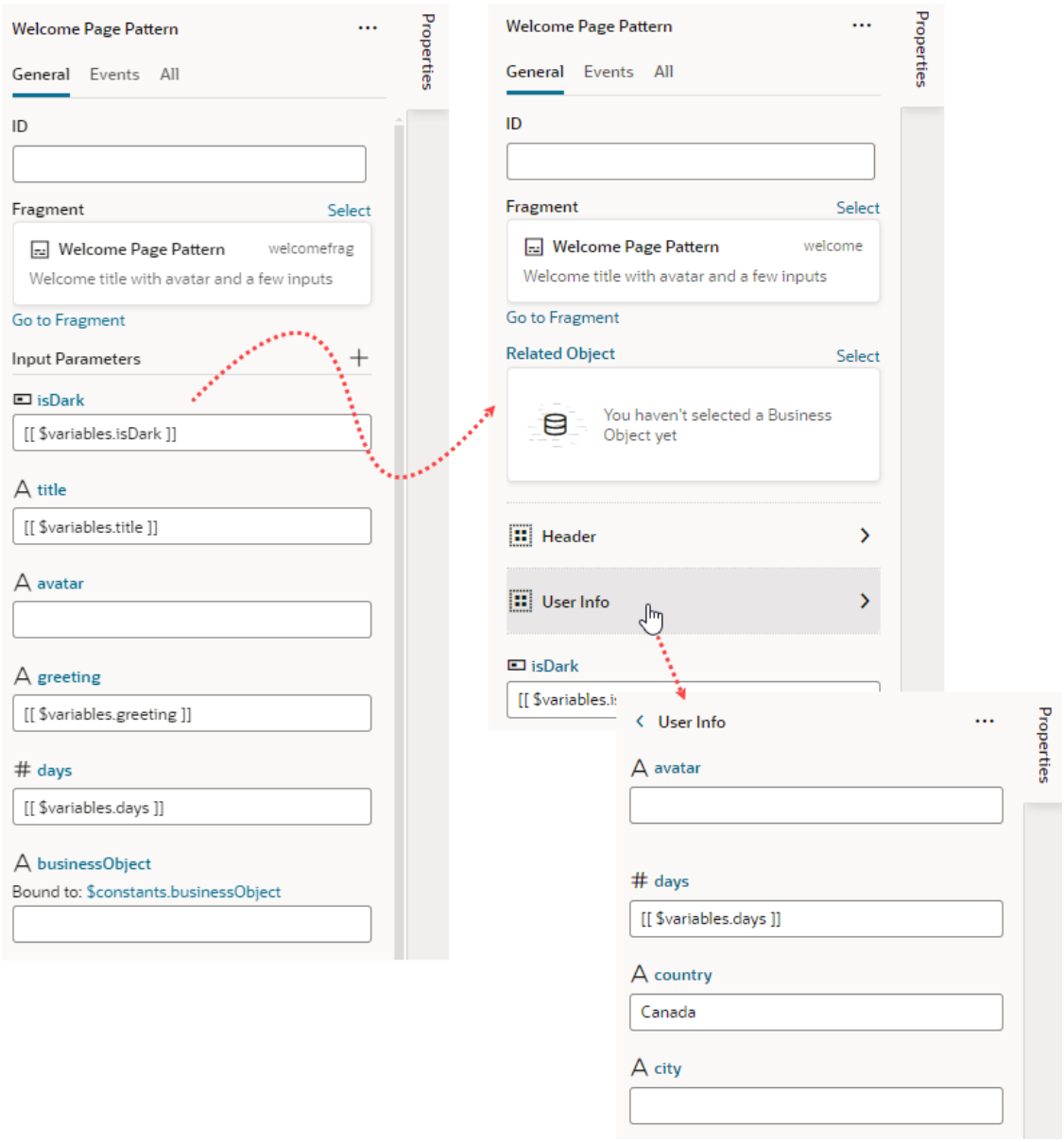
- Only constants can be used. Variables cannot be used.
- The referenced constants must be accessible to the extension performing the LOV service call.

Section Fragment Properties for Display in the Properties Pane

Define a custom layout of sections to display a fragment's most important properties on the Properties pane when the fragment is selected on the page or container that uses it.

Typically, when a fragment is added to a page or container, its input parameters display in the Properties pane based on the order they appear in the fragment's source. To best represent the fragment's structure for those who use the fragment, you can further organize its input parameters—as well as any other components you want to highlight—in sections.

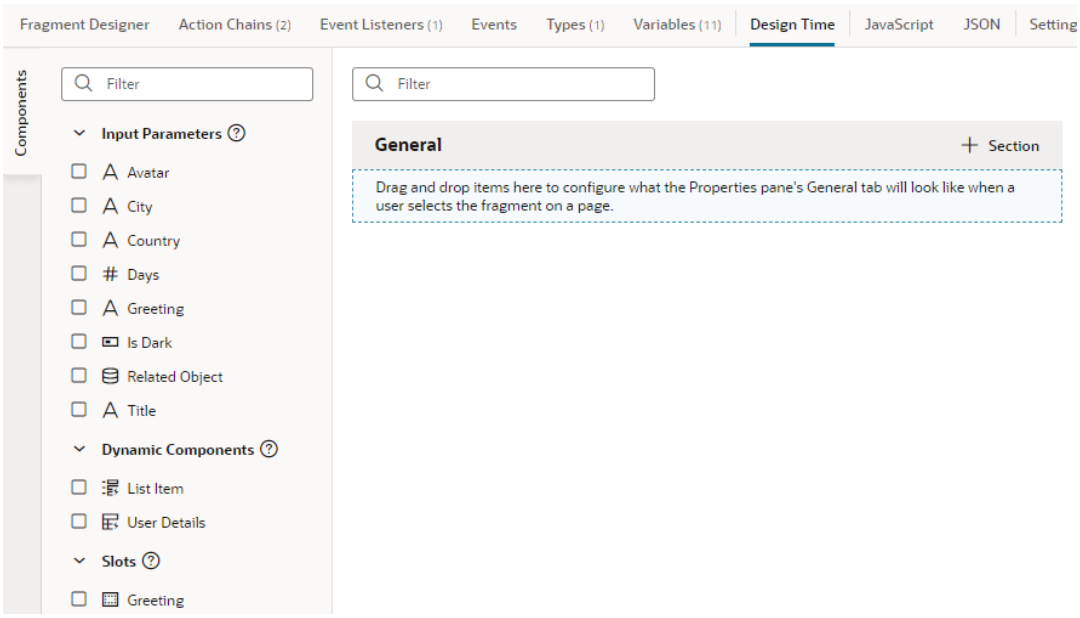
The ability to section a fragment's properties is most useful in pages where a fragment is used as a page template. Here's an example of a page based on a `Welcome Page Pattern` fragment: what you see on the left is the standard view of input parameters; what you see on the right is the sectioned view:



Users will be able to drill down to view input parameters in each section, as shown in the image, where clicking **User Info** shows the **avatar**, **days**, **country**, and **city** input parameters. Take note also of the **Related Object** property, which has been customized to show a component for selecting a business object.

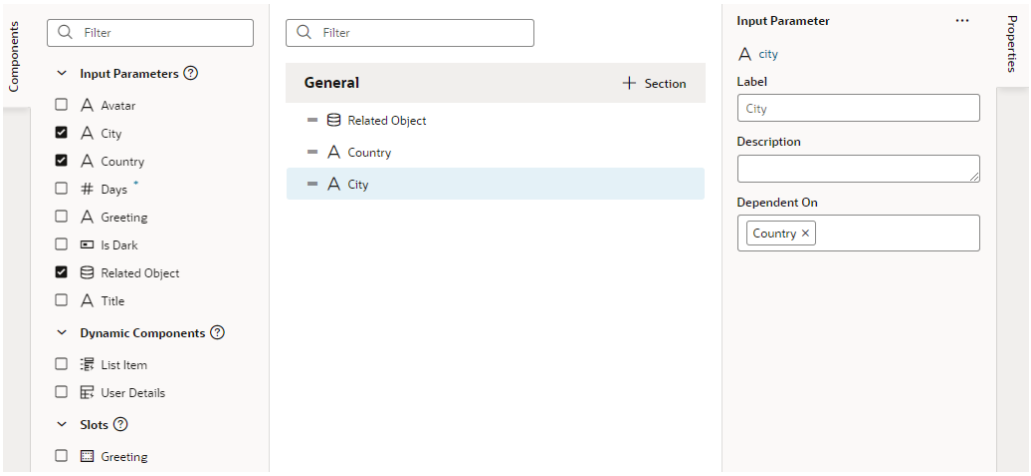
Let's use the same `Welcome Page Pattern` example to see how to section a fragment's properties:

1. Open the fragment's **Design Time** editor.



2. Review the list of input parameters, as well as dynamic components and slots (if any), and decide which ones you want to display as properties in the fragment's page or container. If you want to group them by sections, decide which properties go into each section.
3. Drag and drop items onto the valid area under **General**, or simply select an item's checkbox in the Components palette to add it to the end of your layout.
4. If you want, use each item's Properties pane to set additional properties:
   - Add a **Label** to display a user-friendly name for the item. The Label field is particularly useful for dynamic components with data that may take a while to display in the Page Designer. Instead of a generic `Dynamic Form` or `Dynamic Table` label, users can get a better idea of what the component will display.
   - Enter a **Description** to provide helpful information about the item.
   - If the item relies on other items to display its data, select those items in the **Dependent On** list.

     Suppose your fragment has two input parameters: `Country` and `City`. To indicate that `City` depends on `Country` for its data (allowing fragment users to select a list of cities in that country), you'd select `Country` in `City`'s Dependent On list. This way, when the fragment is selected on the page or container that uses the fragment, `City` will not show in the fragment's Properties pane until `Country` has a value.



5. To create a section and add items to it:

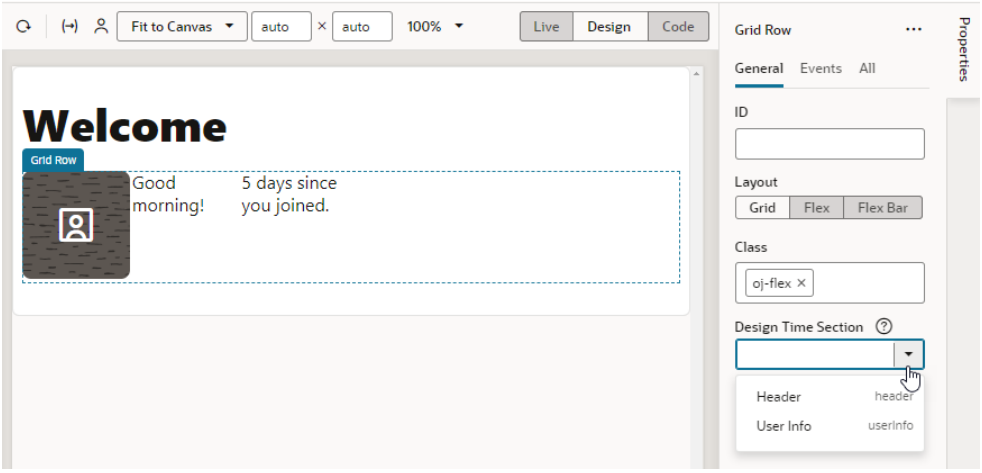
   a. Click **+ Section**, enter a section label in the pop-up (for example, `User Info`), and click **Create**.

   b. Drag the items you want to add to the section and drop it onto the section header (for example, drag the `avatar` and `days` input parameters (among others) and drop them onto the `User Info` header).

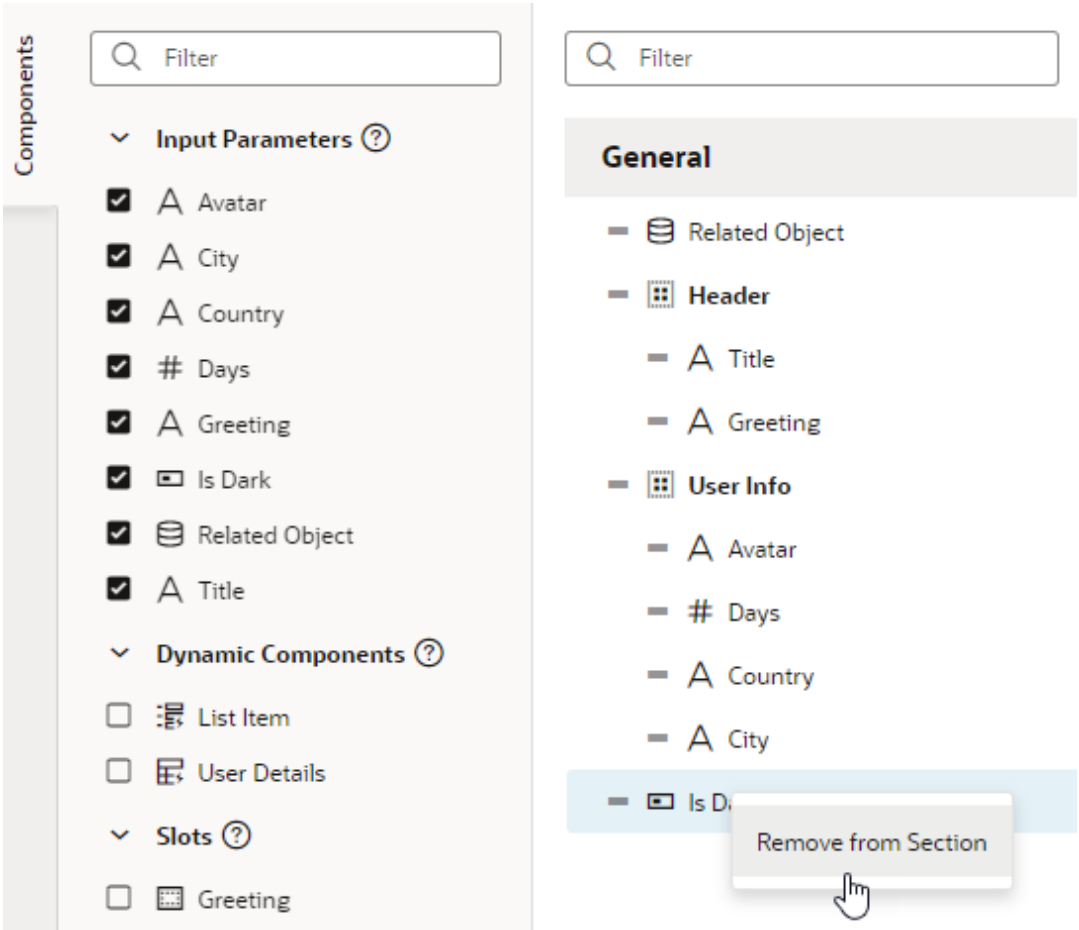
   c. Optionally, select the header and update its properties in the Properties pane:

      - Change the default icon to more easily identify the section: click the **Default Icon**, select an icon from the Icon Gallery, and click **Select**.
      - If you want to hide the section in the fragment's Properties pane until items in that section have values, select those dependent items in the **Dependent On** list.
      - If you want fragment users to see the section's items as a group when they click it on the page or container it's been added to, you can associate the section with a component in the Fragment Designer. To do this, click the **associate a fragment component** link under Associated Component to go to the Fragment Designer, select a component to associate with the section, then in the component's Properties pane, select the section in the **Design Time Section** list.

        Let's say you want fragment users to view all `User Info` properties (such as `avatar` and `days`) when the section is selected on a page. In the Fragment Designer, you might select the **Grid Row** component that contains the `avatar` and `days` items, then in the Grid Row's Properties pane, select **User Info** in the **Design Time Section** list.



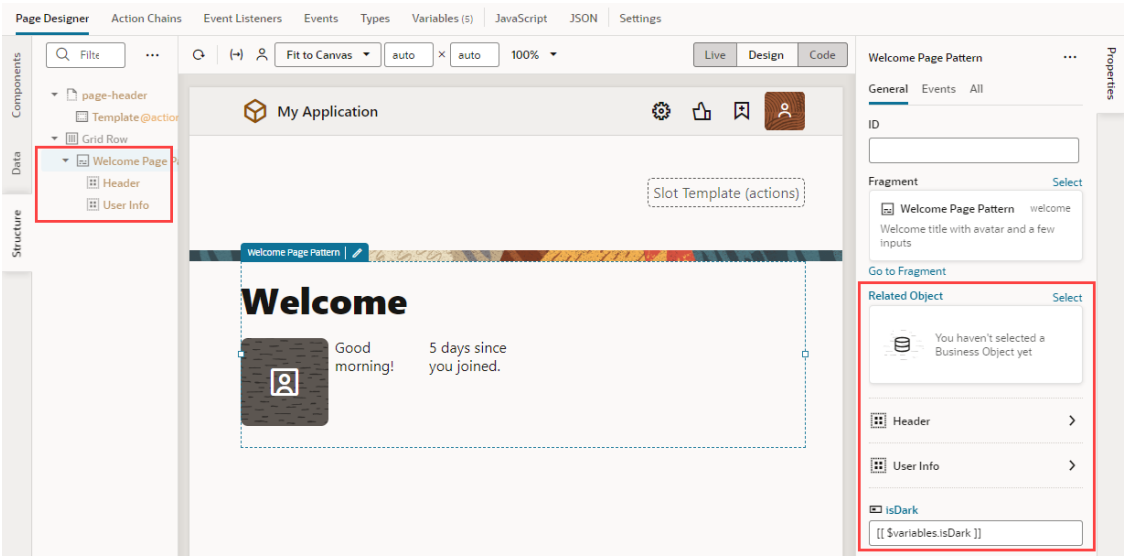
To remove a parameter you added from a section, right-click the parameter and select **Remove from Section**, or deselect the item in the Components palette.



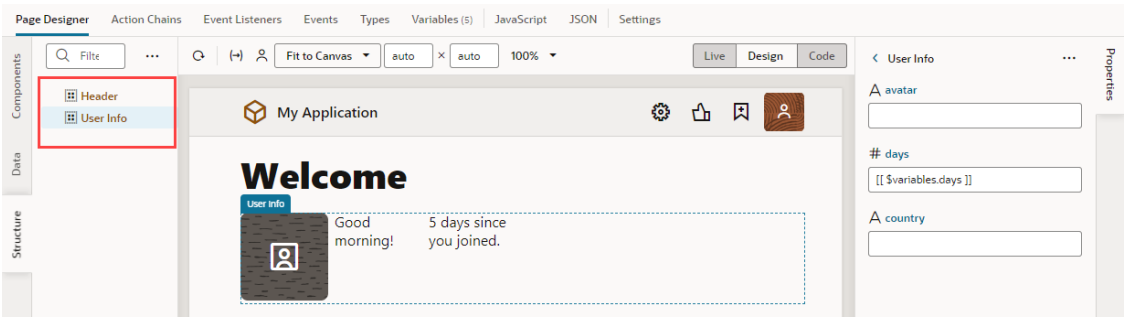
To delete a section, right-click the section and click **Delete Section**.

If you want to reorder the list, drag a property's Handle icon ⁼ and move it up or down as desired.

Now after the fragment is added to a page or a container, you'll see its properties display as sections in the Properties pane when the fragment is selected on the page. The sections also show in the Structure view.



If the fragment is used as a page template to create a page, the sectioned view shows on the page's Properties pane as well as in the page's Structure view, with the fragment considered the root element instead of the page:



Note how selecting **User Info** in the Structure View highlights the fragment components by section on the canvas. Notice also the section's items that show in the Properties pane, allowing users to more easily access all the section's items. If **country** had a value, **city** would also show in the Properties pane.

Part IV
Augment Applications

Once you have the basic building blocks of your application, you can augment the user experience by enabling your app as a Progressive Web App (PWA), adding offline capabilities, and implementing search engine optimization. You can also integrate processes to automate tasks that users must perform and create translatable strings. Most importantly, you'll want to take steps to secure access to the application.

Topics:

- Enable Progressive Web App Support
- Secure the Application
- Add Offline Capabilities to Your Application
- Optimize Your App for Search Engines
- Work With Business Processes
- Work With Translations

17

Enable Progressive Web App Support

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

Unlike apps that run in your browser, PWAs can be installed on your device and have a native look-and-feel. You won't need to build the app as an `.apk`, `.aab`, or `.ipa` file or distribute it through the Apple App Store or Google Play.

A PWA created in Visual Builder:

- Provides a QR code in the browser that your users can scan to install the app.
- Allows your users to install the application from the browser window and add the application to the Home screen.
- Runs on the device in a separate window without an address bar, like a native app.
- Works offline similar to a native app if configured for offline support. To take advantage of offline support, PWA-enabled web apps must be deployed via Grunt. Visual Builder's offline support takes care of making pages, CSS, JavaScript, and all other resources that make up your application available offline. Offline support for your application-specific data must be implemented with the Offline Persistent Toolkit (OPT) or by using standard cache headers.

Guidelines for Using PWA Support

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

- 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 a few limitations for PWAs running on the iOS platform. These issues may be resolved in future iOS releases.
  - Install the PWA using the share icon (because the `Add *PWAappName* to Home screen` message is not displayed).
  - The PWA 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.
  - Some PWA configurations and behavior such as service worker cache size, cache eviction policy, and web manifest support depends on the level of support provided by the browser. Refer to your browser documentation for more information.
  - Orientation lock is not supported.
  - Deep linking is not supported. As a result, clicking a deep link URL on an Apple device will take the user to the Safari browser, instead of the PWA installed on the device.

- 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.

Configure Progressive Web App Support

You can enable and configure Progressive Web App (PWA) support for both web and mobile applications from your app's Settings page.

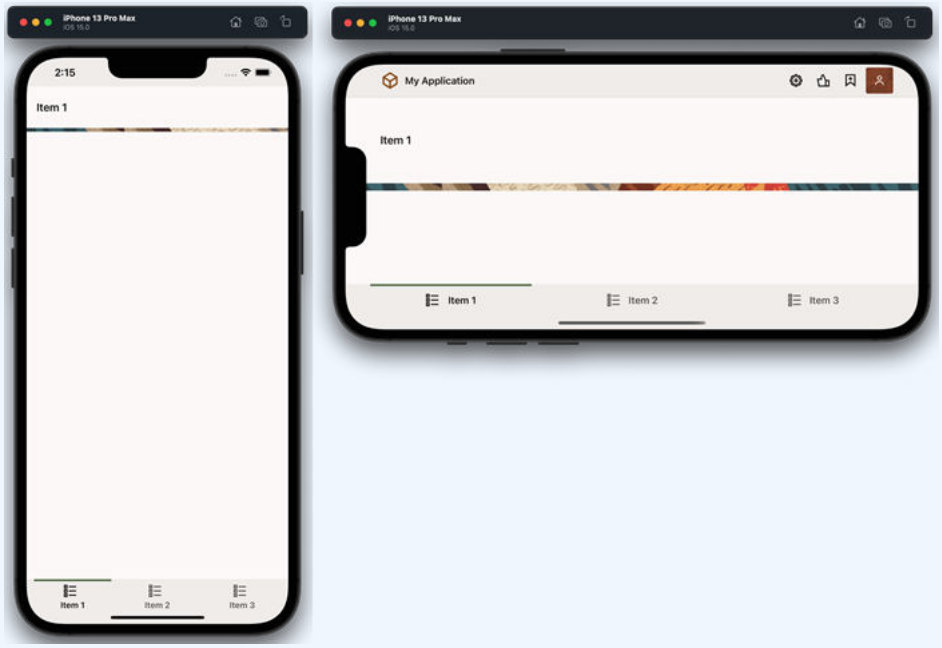
> **Note:**
>
> Mobile apps have been deprecated starting with version 23.10. You can no longer create a mobile app, but you can import an existing mobile app and deploy it as a PWA to be able to use it. You can use your PWA-enabled mobile apps until July 2024 when mobile apps reach End of Life (EOL). To use your mobile-enabled PWAs beyond July 2024, we strongly urge you to transition your mobile app as a web app and deploy it as a PWA.

When you enable an application for PWA, Visual Builder adds the necessary files to your project such as a web manifest file (a JSON configuration file) as well as required icons and splash screens. The web manifest stores configuration settings required by the PWA. You can modify these settings from the Manifest Settings section of the PWA page. For more information on this file, see this page.

Oracle provides Redwood-themed icons and splash screens for your app. You can replace these images with your own branded versions if desired. For PWAs that run on Android 8.0 and higher devices, you can also use adaptive icons that display as a variety of shapes on different device models.

You can also add an offline fallback page to your PWA and customize it to suit your needs. This offline fallback page is displayed when the user performs an action that requires a connection but the device is offline.

1. Click **Web Apps** (or **Mobile Apps**) tab to navigate to the app that you want to configure as a PWA.

> **Note:**
>
> Responsive web app templates (starting with version 22.04) are designed to be visual appealing and can adjust to the size of the user's screen, ranging from small phones to wide-screen desktops. Here's how a web-enabled PWA that uses the Bottom Tabs navigation template renders in portrait and landscape modes; notice how the header items adapt based on device orientation:
>
> 
>

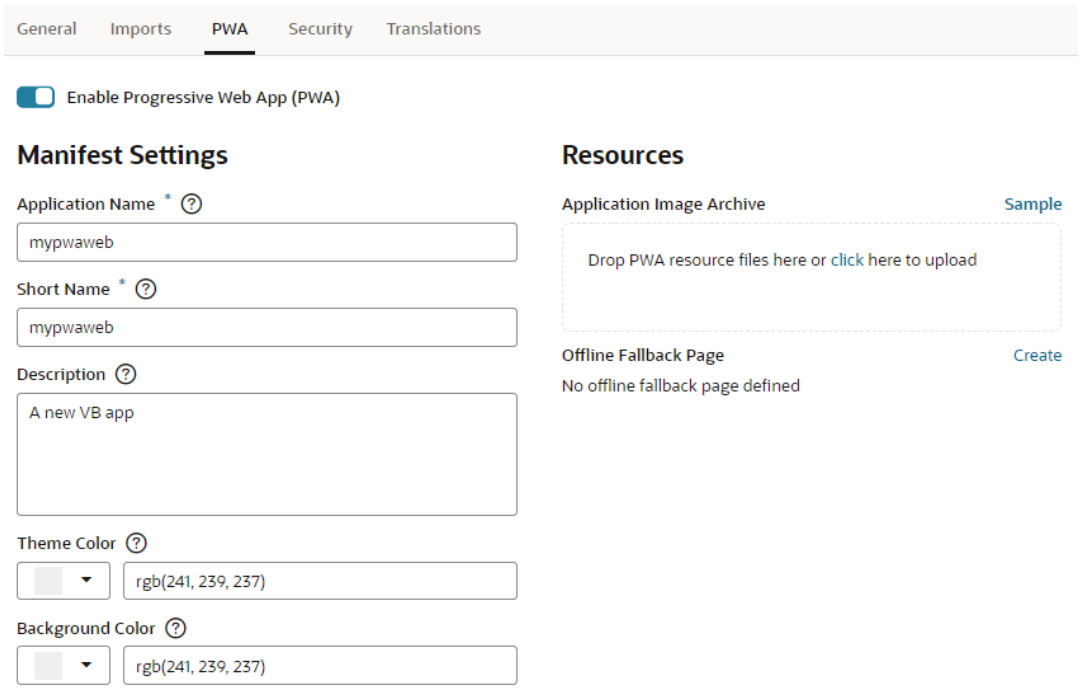
2. Click your app's *<app name>* node, then click **Settings**.

   The General tab is displayed.

3. Select the **PWA** tab.

4. Click **Enable Progressive Web App (PWA)**.

   The Manifest Settings and Resources sections are displayed. Visual Builder also adds the necessary PWA resources to your app's folder tree.



If you're working with a mobile app, you might see settings for advanced file caching. Advanced file caching is not supported for web apps.

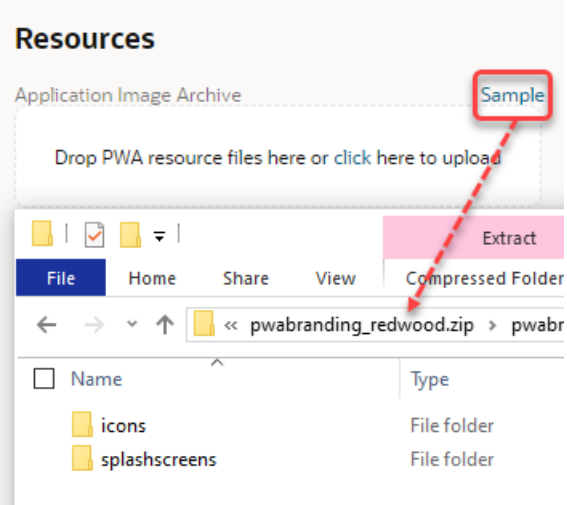
5. Review and edit, if needed, the information in the Manifest Settings section.
   a. Specify the Application Name.

      The application name appears on the dialog that displays to prompt 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. (Optional) Customize the splash screen and icons to use with your app.
   a. From the Resources section, click **Sample** to download the `pwabranding_redwood.zip` archive file.



b. Modify the icons and splash screens as needed.

   When you design adaptive icons, ensure that important information is within a "safe zone" circle, with a radius equal to 40 percent of the image size. By default, Redwood theme icons generated for PWAs in Visual Builder follow this guideline, as do the icons included in the sample archive. It is recommended that you download the sample from the Resources section and use the images within as a guideline to create your adaptive icons.

c. Upload your modified application image archive either by dragging it to the drag-and-drop area or by clicking the link to launch the Open dialog box.

7. If you are using adaptive icons, modify the web manifest file to include the `"purpose": "any maskable"` setting as follows:

```
{
 "sizes": "512x512",
 "src": "resources/icons/icon-512x512.png",
 "type": "image/png",
 "purpose": "any maskable"
}
```

8. To include an offline fallback page for your application, click **Create** next to Offline Fallback Page.

   Visual Builder adds a Redwood-themed offline page (`offlinePage.html`) to your project and provides a link under Offline Fallback Page. If you want to customize the offline fallback page, click the link to open it in the Designer and modify it as required.

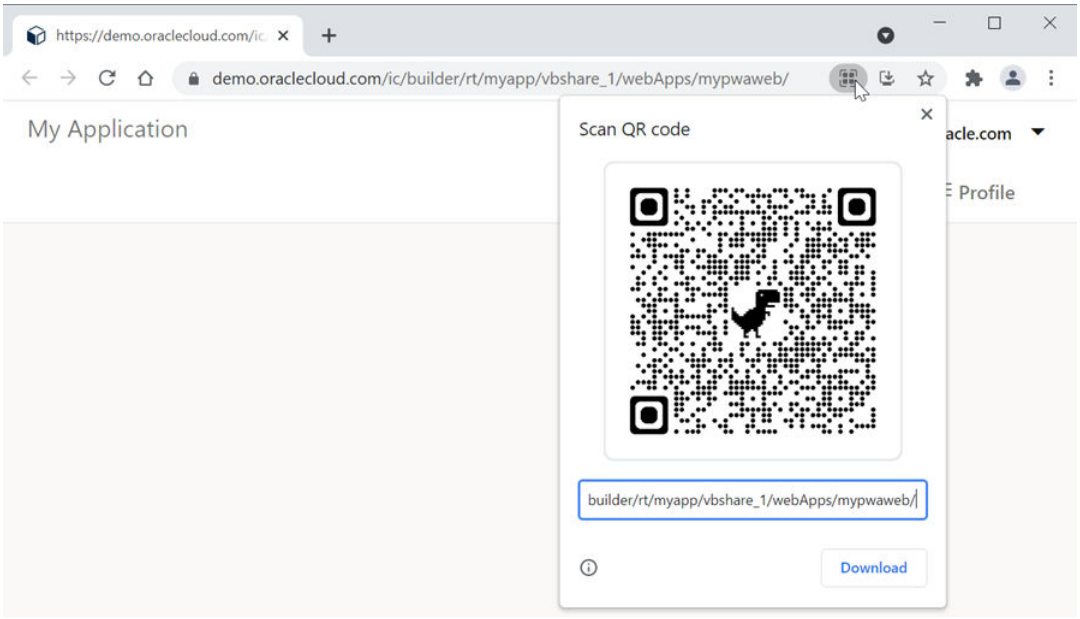
9. For mobile applications, specify the files to cache on the user's device in the Advanced File Caching section.

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

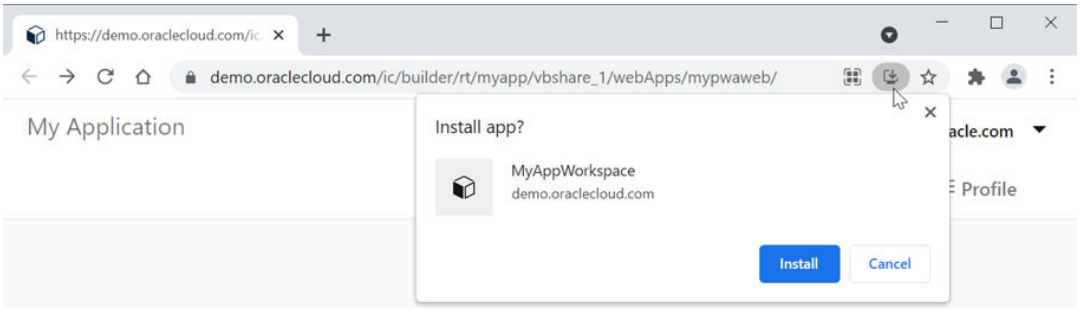
After you enable PWA support, you can stage and publish a *web* app to distribute it as you would any web app. See Stage and Publish Visual Applications. To stage and publish a PWA-enabled *mobile* app, you must first build the app to generate a QR code. See Build a Mobile Application as a PWA.

When you open the link for a PWA in your browser, you can use the options in the address bar to install your app as follows:

- Click the address bar and use the QR Code icon to display the app's QR code which you can then scan to install the app:



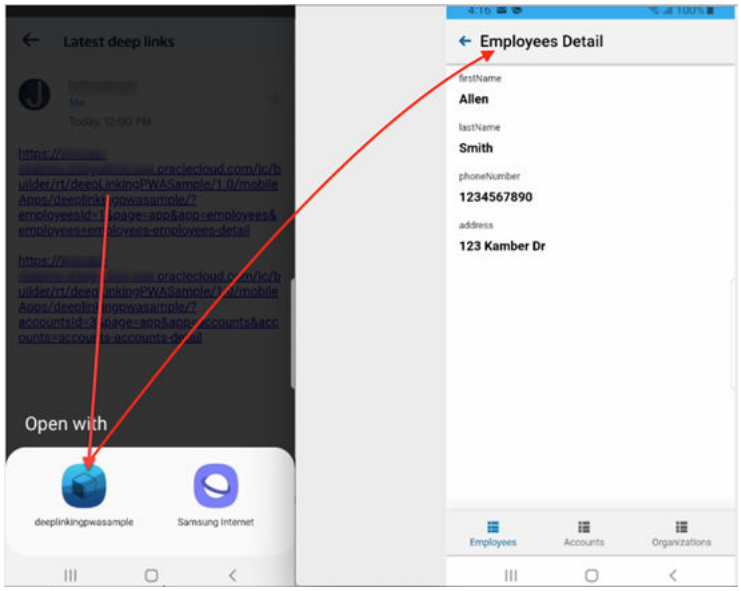
- Or use the install icon in the address bar to install the app:



Deep Linking on Android

Deep linking allows users to open a specific page or content in the PWA directly, instead of requiring them to search or navigate to the content from the Home page. For PWAs on Android devices, deep linking is enabled without the need for any additional configuration.

Clicking a deep link URL shared in an email or any other application for the first time after the PWA is installed prompts the user to either open the link in the installed PWA or in the browser. Once the user chooses to open the link in the PWA, the device remembers this choice. From then on, it will open other deep link URLs only in the PWA. For secure applications, the user will need to log in to view data.



Here are examples of deep link URLs as they appear on Visual Builder PWAs, specifically those that point to an employee's details page:

- When using a query, the URL could be similar to `https://demo.oraclecloud.com/ic/builder/rt/hrapplication/1.0/webApps/hrwebpwa/?employeesId=1&page=app&app=employees&employees=employees-employees-detail`
- When using a path, the URL could be similar to `https://demo.oraclecloud.com/ic/builder/rt/hrapplication/1.0/webApps/hrwebpwa/vp/app/employees/employees-employees-detail?employeesId=1`

> **Note:**
>
> If the deep link URL takes the user to a detailed page with a Back button, it falls to the app developer to implement the logic for the Back button.

Run Mobile Applications as PWAs

With enhanced capabilities for PWAs and the responsive template for web applications, mobile applications have been deprecated in favor of PWAs. You can no longer create a new mobile application in Visual Builder, but you can continue to use an existing mobile app by deploying it as a PWA.

To deploy your existing mobile app as a PWA, first import it, then enable the PWA option in the application's Settings page *before* you stage or publish the mobile app. Once you do that, you can build your PWA-enabled mobile application to generate a QR code, which users can scan to install the app on their devices. This way, you can have a single application that is installable and can scale from desktop to tablet to mobile. It can also work offline if configured for offline support.

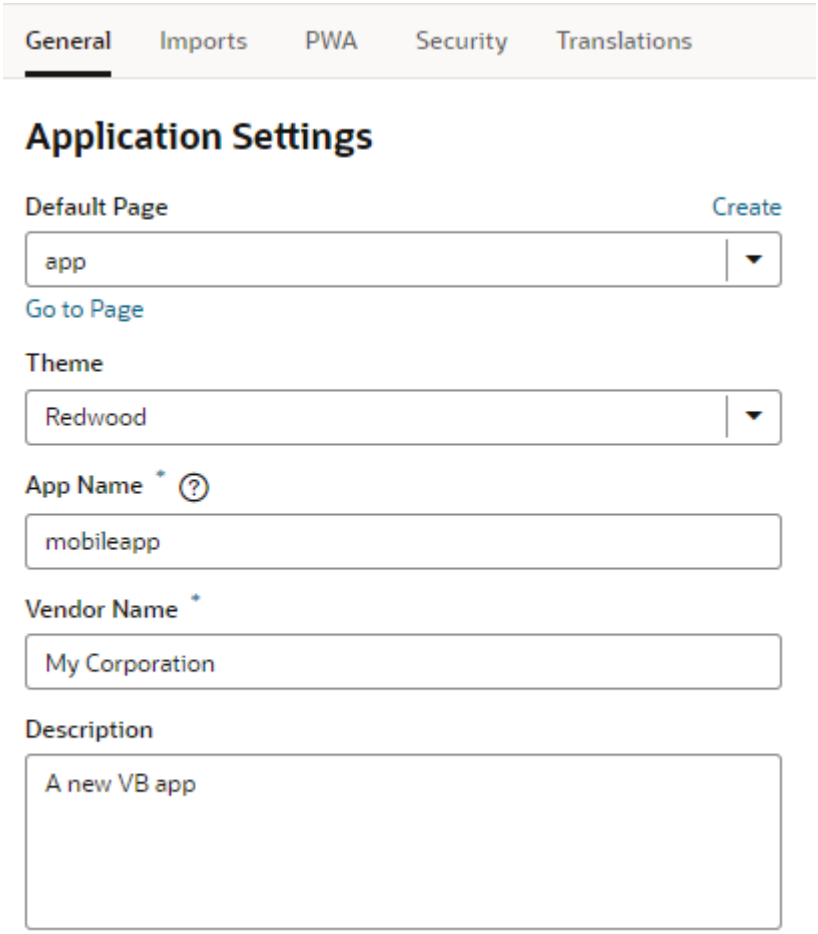
> **Note:**
>
> You can continue to use your existing mobile apps until July 2024 when mobile apps, including PWA-enabled ones, reach End of Life (EOL). To use your mobile PWAs beyond July 2024, we strongly urge you to transition your mobile app as a web app and deploy it as a PWA.

Configure Mobile Application Settings

Review the settings for an imported mobile application to verify that it has the appropriate values.

1. Click the **Mobile Applications** tab.
2. Click the *app_name* node and click **Settings**.

   The General tab is displayed.



3. In the Application Settings page, 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.

> **Note:**
>
> Your mobile application's theme defaults to the latest JET theme at the time of creation. If your app was created with Visual Builder 20.07 or earlier, the theme is likely to be Alta (which was the default at that time). If your application was created with Visual Builder 20.10 or later, the Redwood theme is likely the default for your application. You can change the theme in the mobile app's Settings editor, but you can't assume that changing the theme from Alta to Redwood will automatically work, as components' dimensions and styling are different between the two themes. See Customize a Web App's Appearance.

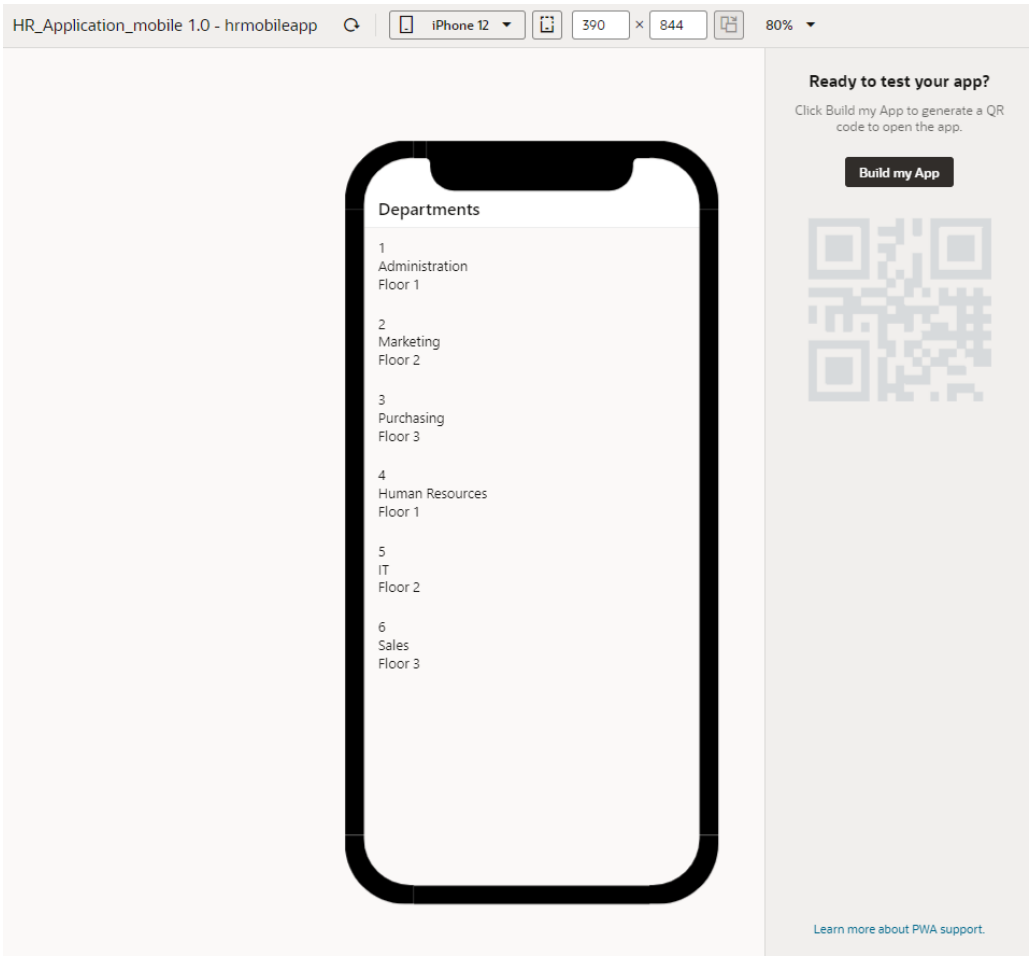
5. In the App Name field, enter the app name that is to be displayed when the app is installed on a mobile device.
6. Enter the name of the vendor who originated the application in the Vendor Name field.
7. Enter text that describes the application in the Description field.

Build a Mobile Application as a PWA

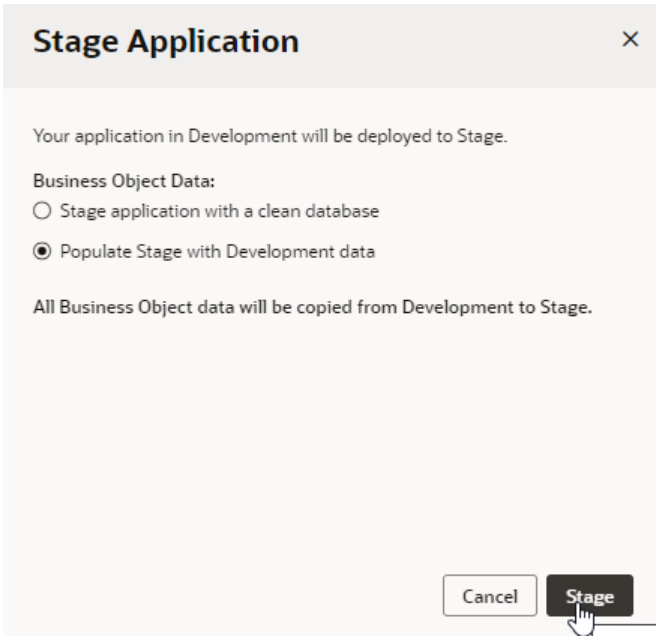
Build your PWA-enabled mobile application to generate a QR code for the application. Users can scan the QR code to install the mobile application as a PWA for testing as well as for production.

Staging and publishing a mobile application is similar to what you'd do to stage and publish a web application, except that you first build your mobile app with the information that lets users install the app on supported platforms.

- Follow these steps to stage a PWA-enabled mobile application:
  1. Open the PWA-enabled mobile application that you want to build.
  2. Click the **Preview** icon ( ) to run the app on a new tab in the browser.
  3. When the mobile app opens in the browser, click the **Build my App** button.



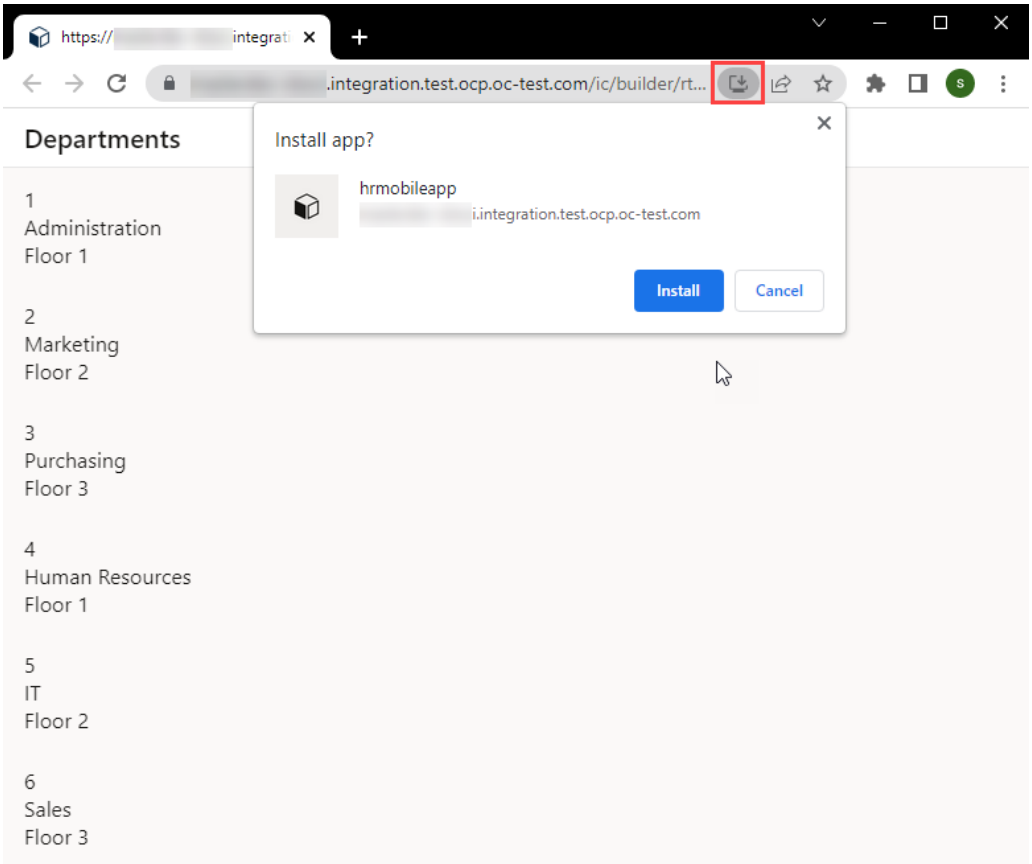
4. In the Stage Application dialog box, select the appropriate option to determine what to do with your business objects, then click **Stage**.



When the build is complete, the generated QR code is displayed.

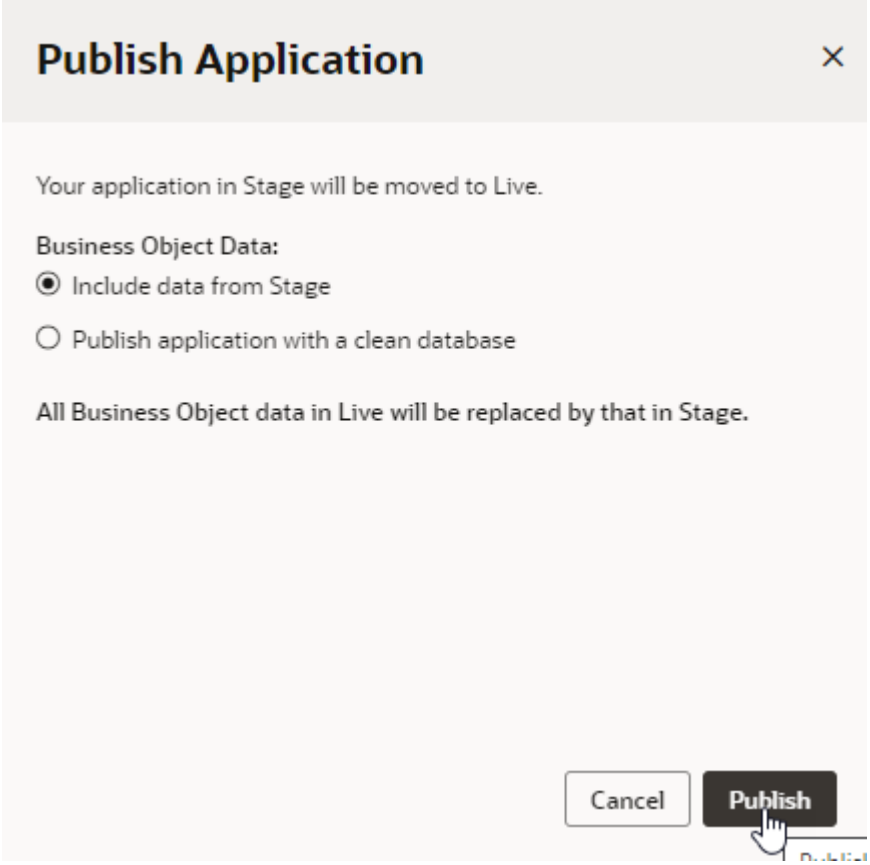
5. Scan the QR code to install the mobile application on a supported device. Alternatively, click the **Launch in Browser** link to view the app in your laptop or device browser.

When you open the link, you can use the Install icon in the address bar (as highlighted here) to install your app on your laptop or device:



- Follow these steps to publish your staged PWA-enabled mobile application and make it live:

  1. Open the PWA-enabled mobile application you want to publish.
  2. Select **Publish** from the application's Menu option in the upper right corner.
  3. In the Publish Application dialog box, specify what to do with your business object data, then click **Publish**:



     The schema and data from the staging database are copied to the live database.

  4. To view the published app, go to the Visual Applications home page and locate your application. Click **Live** in the Status column, then select your mobile application.

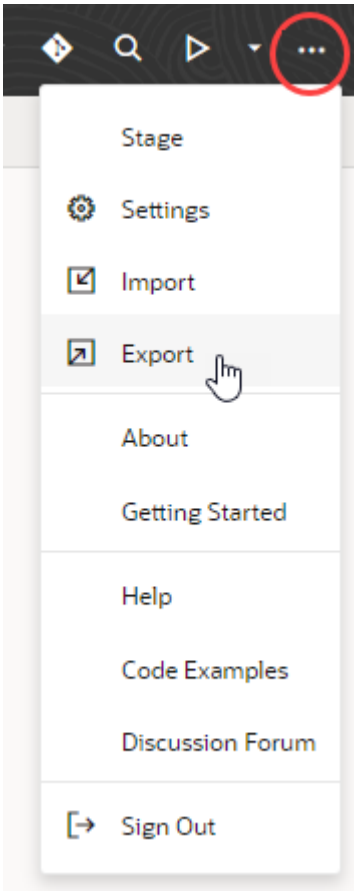
     The application opens in a new browser tab on the application's start page. On the right is the QR code that users can use to install the mobile application on a supported device. The option to launch the app in a browser is also available. You can now share this application URL with your users.

Convert a Mobile PWA to a Web PWA

To be able to use your mobile apps beyond 2024 (when they reach End of Life), we strongly urge you to convert your existing mobile app to a web app and deploy it as a PWA.

To convert your PWA-enabled mobile app to a web app:

1. Open an existing mobile application, then export the application to download it as a ZIP archive to your local file system.



> **Note:**
>
> Create a backup of the application's ZIP archive, so you have a copy you can use in case you run into issues.

2. Extract the contents of the archive. You should see something similar to this directory structure:

```
businessObjects/
mobileApps/
settings/
Gruntfile.js
package.json
visual-application.json
```

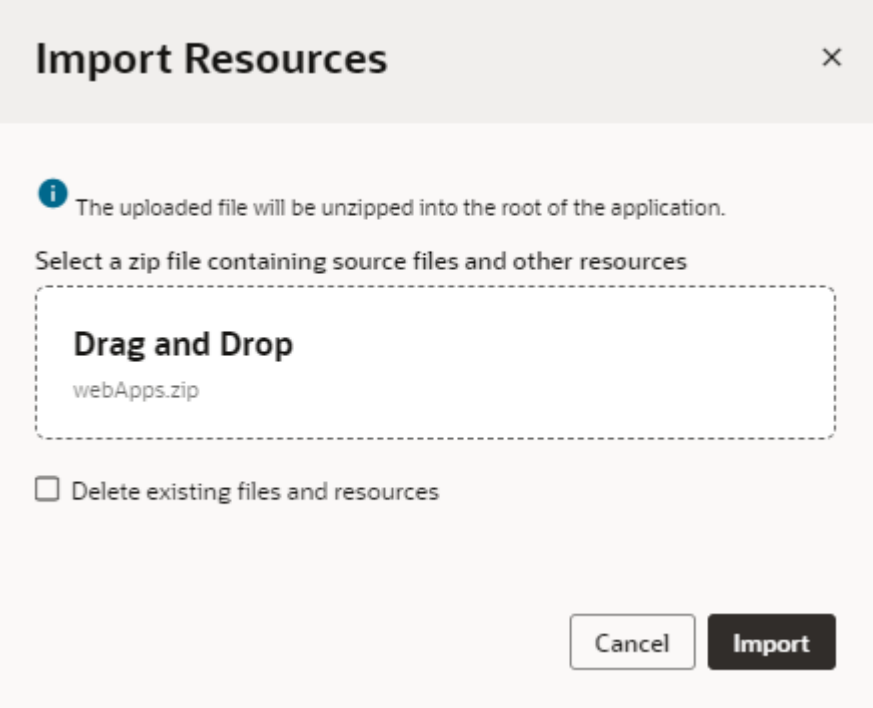
3. Create a new `webApps` directory (take note of the capitalization) in the extracted folder.
4. Copy the contents of your `mobileApps` directory into the `webApps` directory. For example, if your `mobileApps` directory contains a `hrmobileapp` folder, copy the `hrmobileapp` folder to the `webApps` directory.
5. Go to the mobile app directory you copied into the `webApps` directory (`hrmobileapp` for example) and remove these files:
   - `manifest.json`

- `settings/appShellCache.json`
- `settings/build-configurations.json`
- `settings/mobile-build.json`
- `mobile-build-templates/*` (if it exists)

6. Compress the `webApps` directory into a new archive (for example, `webApps.zip`).

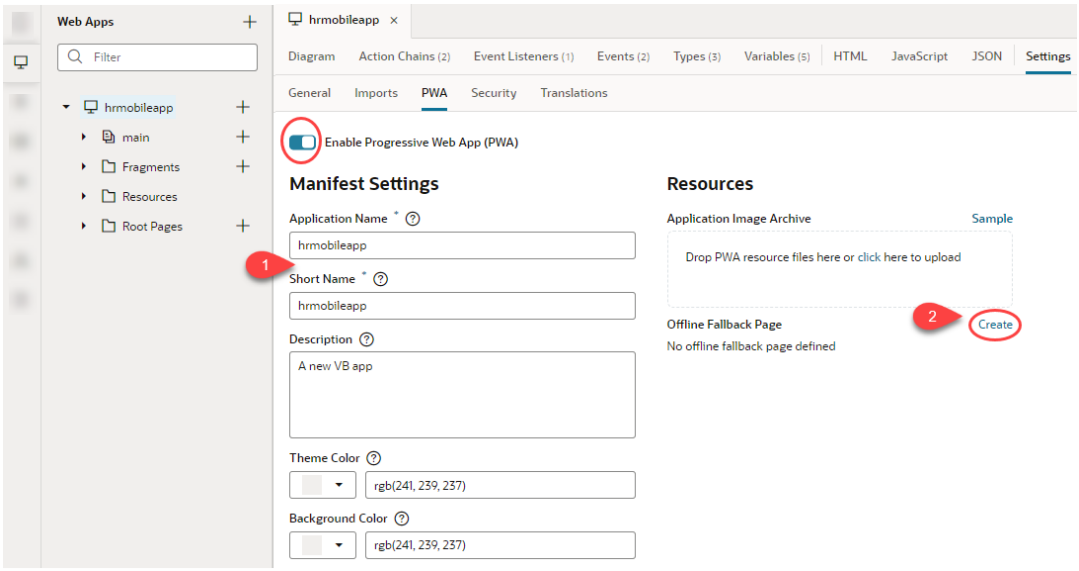
> **Tip:**
>
> If you're working on a Mac, use these steps to create an archive:
>
> ```
> % cd <Visual Application Directory>
> % find . -name '.DS_Store' -type f -delete
> % zip -r webApps.zip webApps
> ```

7. With your mobile application openin Visual Builder, import the `webApps` archive you created. Don't select the **Delete existing files and resources** option:



Your new web app should now show up in the Web Apps pane.

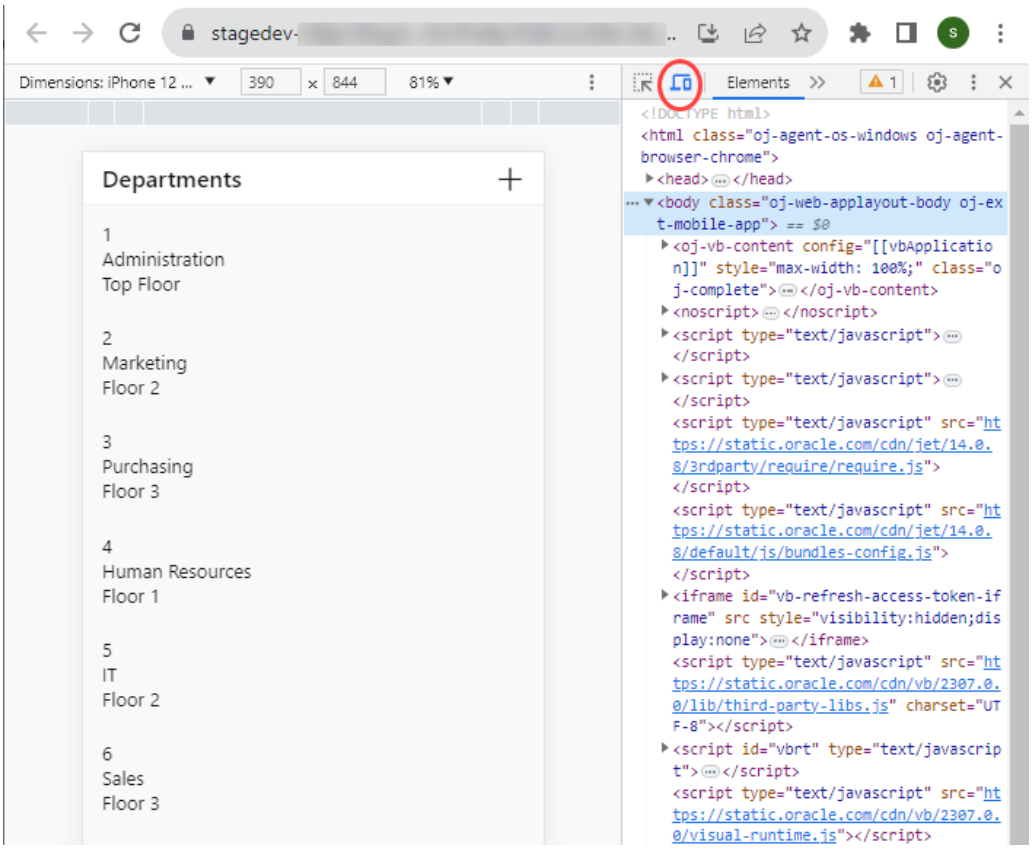
8. Enable PWA support for your new web app:
   a. Select the app's <*app name*> node in the Web Apps pane, then click **Settings** and **PWA**.
   b. Click the **Enable Progressive Web App (PWA)** toggle to disable and re-enable PWA support.
   c. Review PWA settings for your web app. You might want to change the Manifest Settings and create an Offline Fallback Page.



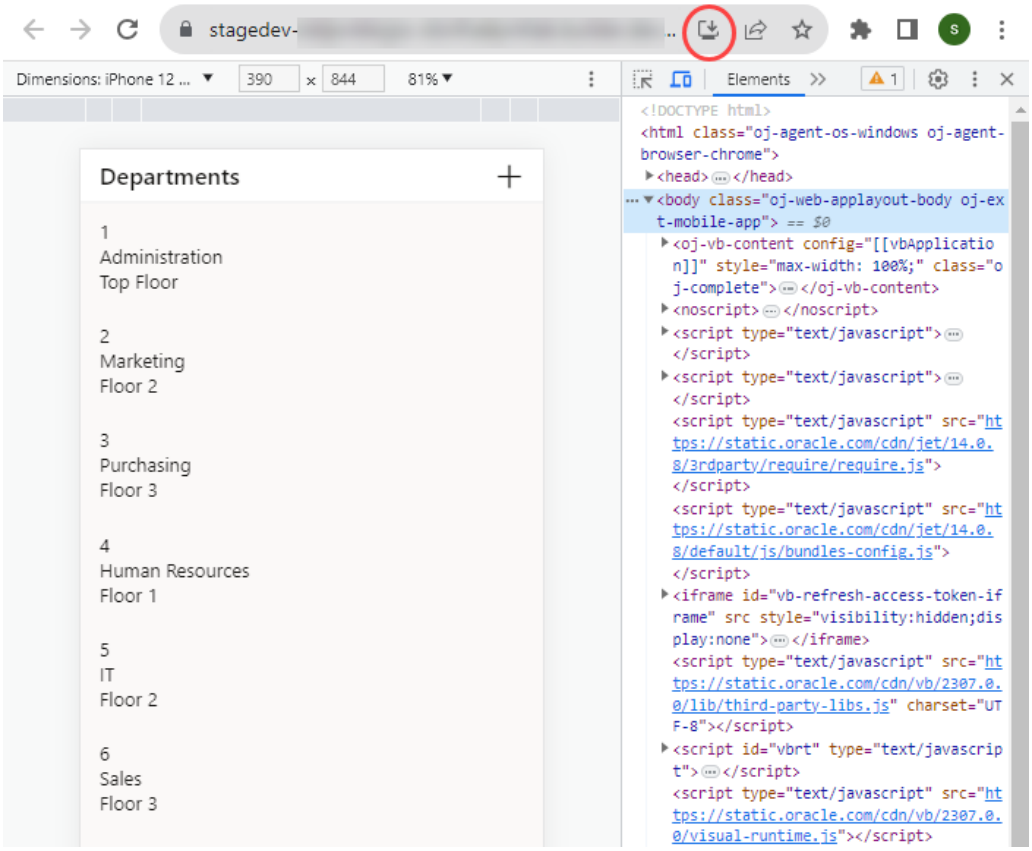
9. Now stage your application to test and make sure everything works as expected.

   Your new web PWA should work the same way as your original mobile PWA. When your PWA-enabled web app opens in a browser:

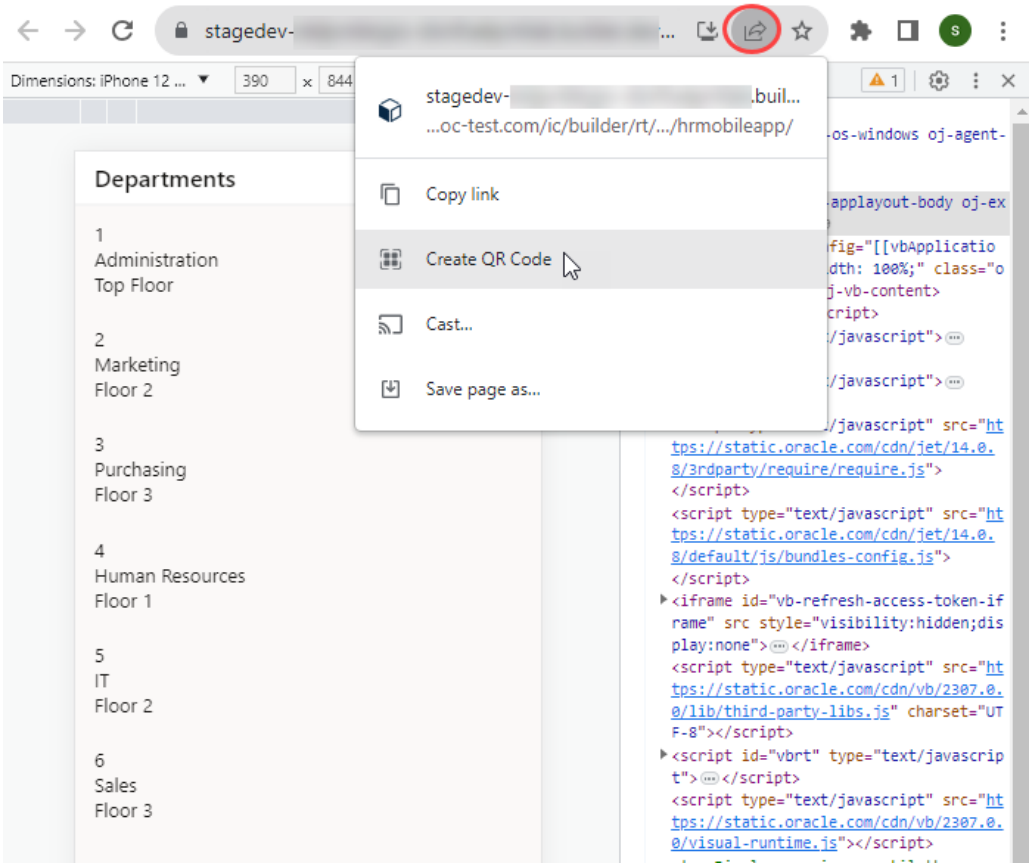
   a. Use the device toolbar in the browser's Developer tools to debug your app in a simulated mobile environment:



   b. Use the install icon in the app's address bar to install your web app:



c. Use the share icon in the app's address bar to generate a QR code that will allow users to quickly open the web app on a mobile device via its camera:



After you've confirmed that your app works as expected, you can choose to delete the mobile app from your visual application.

> **Note:**
>
> If your mobile app leveraged the Oracle JET Offline Persistence Toolkit to provide offline capabilities, you need to publish your application using Grunt commands, or better yet, use the package and deploy pipeline in VB Studio.

18

Secure the Application

You can secure access to your application with user credentials and create user roles to secure data stored in your application's business objects.

Security for Web Apps

Visual Builder apps can use Oracle Identity Cloud Service (IDCS) for token-based authentication. Token-based authentication protects your business data from unauthorized access, while allowing your app's users to access the app now and again without having to log in each time.

When a user logs in to your deployed app, the app authenticates with IDCS, which sends a token to the app. Once authenticated, the user can continue to use the app without having to log back in until the token expires, typically after 8 hours.

Whenever the app makes a call to the REST service, it retrieves the token and attaches it to the request. As long as the token is still valid, the REST service sends the appropriate response. If the token has expired, the service rejects the request (returns a '401') and the user is redirected to the log-in screen.

For web and PWAs (including PWA-enabled mobile apps), the token is stored in the browser session and is discarded when the user closes the browser window, exits the PWA, or reboots the device. When the user relaunches the app following one of these events, they are prompted to log back in.

The following table describes the authentication behavior after some common user events such as restarting, rebooting, and going online:

What happens if ...	Web	PWA
...I quit my app or it crashes and I relaunch it? ...I reboot my device and relaunch the app?	I am prompted to log back in. For web apps, the token is stored in the browser session and is discarded when the browser window or the app is closed.	If the device is online, I am prompted to log back in. For PWAs, the token is stored in the browser session and is discarded when the browser window or the app is closed. If the device is offline, the PWA uses a cached user object to allow me to continue working with cached data. I am only prompted to log back in when I reestablish an Internet connection.
...I switch from a data network to a WiFi network or vice versa?	I am not prompted to log in. Changing networks does not affect token behavior or duration.	I am not prompted to log in. Changing networks does not affect token behavior or duration.

What happens if ...	Web	PWA
... I lose my network connection or switch to airplane mode?	I receive a browser error, such as a "No internet" error (Google Chrome). If my web app uses the cache control HTTP header to manage cached data, I can continue to work in offline mode. See Add Offline Support Using the Oracle Offline Persistence Kit.	I can continue to work in offline mode with cached data even after the token expires since I am not connecting to the server. The app uses the Oracle Offline Persistence Toolkit (OPT) to manage cached data. See Add Offline Support Using the Oracle Offline Persistence Kit.
...My device comes back online?	If the token is still valid, I can continue working as before without having to log in. If the token has expired, I am prompted to log in again.	If the token is still valid, I can continue working as before without having to log in. If the token has expired, I am prompted to log in again.

Authentication Roles Versus User Roles

You use *authentication roles* 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's resources by creating *user roles* and assigning authenticated users to them.

All app users are automatically assigned either the *Anonymous User* or *Authenticated User* authentication role. When access to the app requires authentication (default), all users are granted the Authenticated User role when they sign in. If anonymous access to the app is allowed, users are granted the Anonymous User role. You can use these roles when granting permissions to operations on business objects when role-based security is enabled. Here's a table that describes the two authentication roles:

Authentication Role	Description
Authenticated User	All users who access 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.
Anonymous User	All users who access Visual Builder applications are assigned this role when anonymous access to the application is enabled. An anonymous user cannot access data stored in the application's business objects or retrieved from services, unless anonymous access is explicitly enabled for the Anonymous User role.

When your app requires authentication, you can further control access to business objects and data in your application through user roles. The application’s user roles ensure that users assigned the same role or group in the identity provider are granted equal access in your application. You define user roles in the **User Roles** tab of your application’s Settings editor. See Manage User Roles and Access.

As a developer, you can 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. 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.

> **Note:**
>
> If you want to federate IDCS with your existing identity provider, see Federating with Identity Providers.
>
> You can also choose to override the default security provider that an app is using by creating your own security provider that maps to a third-party provider. Note that this might affect functionality such as identity propagation to REST service calls. See Security 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 in the Business Objects 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.

Manage User Roles and Access

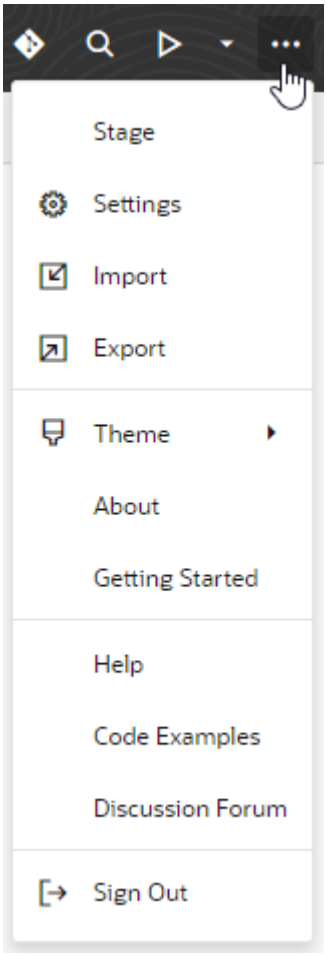
You can create, edit, and remove user roles to secure access to your application's business objects.

In addition to the Authenticated User role granted to users who sign in to your application, users can be assigned a user role based on their credentials and the groups they've been assigned to in Oracle Identity Cloud Service (IDCS). When a user tries to access data in a business object secured by this user role, the roles assigned to the user are authenticated in IDCS. Access is granted if one of the user roles securing the business object is mapped to one of the groups the user has been assigned to in IDCS or if the user was mapped to that user role directly.

Use the **User Roles** tab in a visual application's Settings editor to create a user role and assign users and groups in your IDCS account to the user role. Assigning groups to your user role maps the role to IDCS groups and is known as "role mapping". Once you create a user role, the role and any users or groups assigned to it are automatically added to the client application in IDCS.

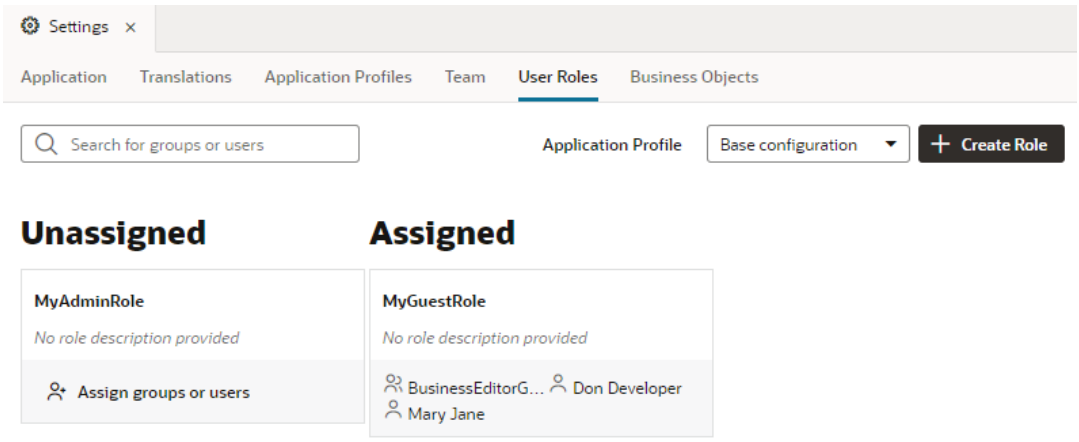
To create a user role in your visual application:

1. Click **Menu** in the upper right corner and select **Settings**.



2. Click the **User Roles** tab in the Settings editor.

   If user roles have been defined, you'll see a tile for each user role in your application (along with the groups and users assigned to it).



3. Click **Create Role**.
4. Enter a name for the role in the Create Role dialog box. Click **Create**.

   This role name is displayed when designing your application, but is not exposed to users.

5. Before you proceed to assign groups or users, or edit a user role, verify that the application profile selected in the Application Profile drop-down list is the one where you want to make changes.

   Visual Builder provides a ready-to-use application profile (Base configuration) that is used by default for your app's development, stage, and live phases. If you want to use another profile, duplicate the default one to create another profile, then associate it with the user role you want to use.

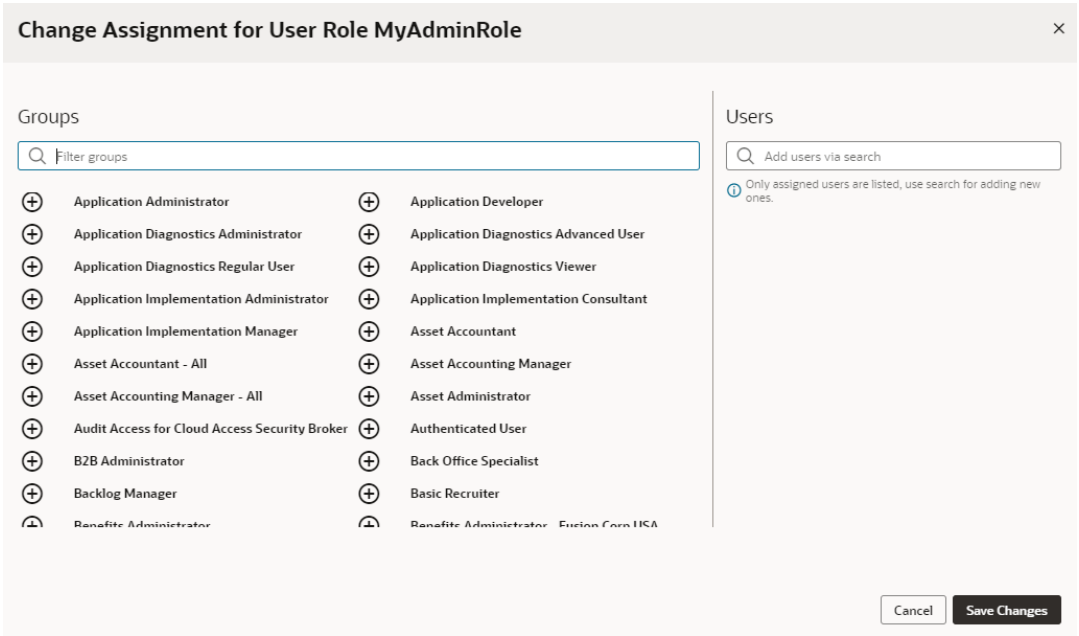
   a. Switch to the **Application Profiles** tab in the Settings editor.

   b. From the Base configuration's menu, click **Duplicate**, then provide a new name (for example, Prod configuration) and description in the **Duplicate Application Profile** dialog. Click **Duplicate**.

   c. Switch to the **User Roles** tab and select the profile in the Application Profile drop-down list.

6. 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 that appears when you hover your cursor over the tile.

7. In the Change Assignment... dialog box, click for each group that you want to assign to the role. In the Users field, enter the name of the user that you want to add, or enter a character to retrieve a list of users. For example, enter `a` to retrieve all user names that include the character `a`. Click to add the user to the role.



   You can assign multiple groups and users to your user role. Keep in mind that the list of groups and users is defined in the identity provider and managed by the identity domain administrator. Click **Save Changes** when you are done. Saving your changes automatically updates the user roles for your application in IDCS.

After you create a role, you'll need to enable role-based security for the application's business objects by specifying the user roles that can access the object and setting access privileges for the role in the business object's **Security** tab.

Besides securing access to the data in your business objects, user roles can help control what a user sees in your application. For example, you can use role-based permissions to limit access to the app, entire pages or flows, even set restrictions on certain components in a page, so only users with certain roles can view that information.

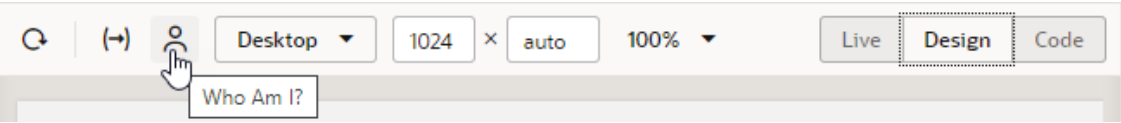
> **Note:**
>
> An application's user role definitions are preserved whenever it is exported and imported—as long as the app is imported to the same IDCS domain it was exported from. When you export an app, its user roles (as defined in `user-roles.json`) are included in the exported application archive (`role-mapping.json`), then re-created when you import the application. Once this is done, the `role-mapping.json` file is deleted from the application's sources. But if you run into errors and this doesn't happen (say, because you're importing an older app whose users and groups no longer exist in IDCS), you'll need to manually set up the user roles again.

Test Role-Based Access

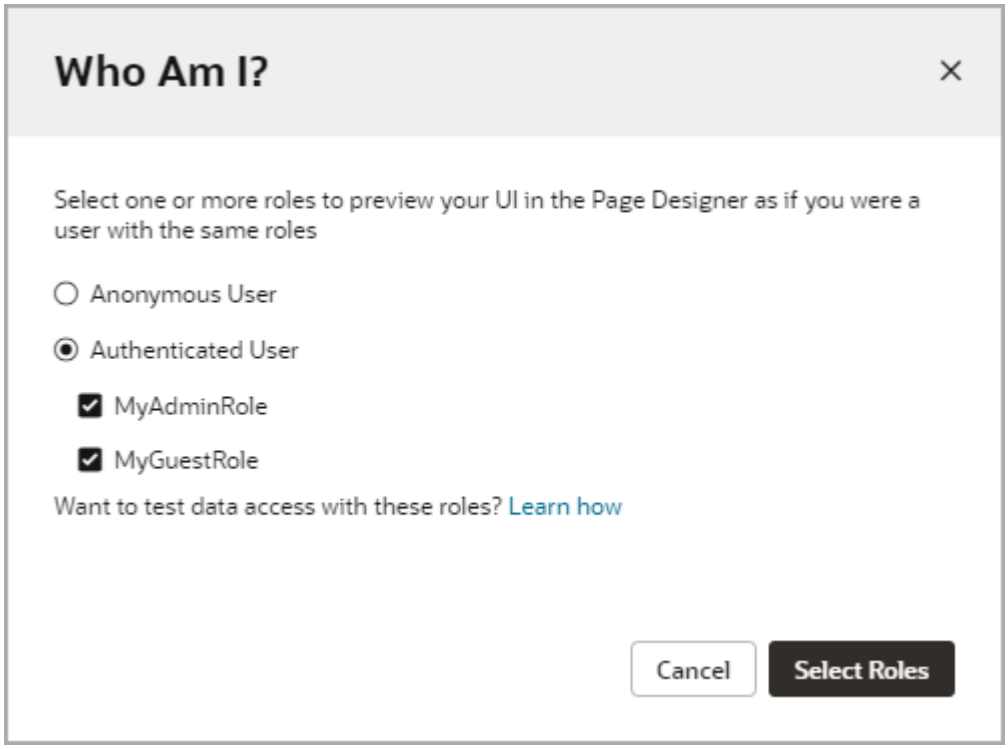
When role-based security settings are defined, you can preview how your application will look and behave for users with different roles.

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.

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.



The **Who am I?** dialog will list all your application's roles. You will always see the Authenticated User and Anonymous User in this dialog. If you've defined custom user roles, you'll see those as well:



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.

Access and Secure Business Objects

Enable role-based security to control access to your business objects through REST endpoints, both for apps in your visual application and external clients. You can configure each business object's security settings to control the user roles that can access the endpoints and the types of operations they can perform.

Secure Business Objects

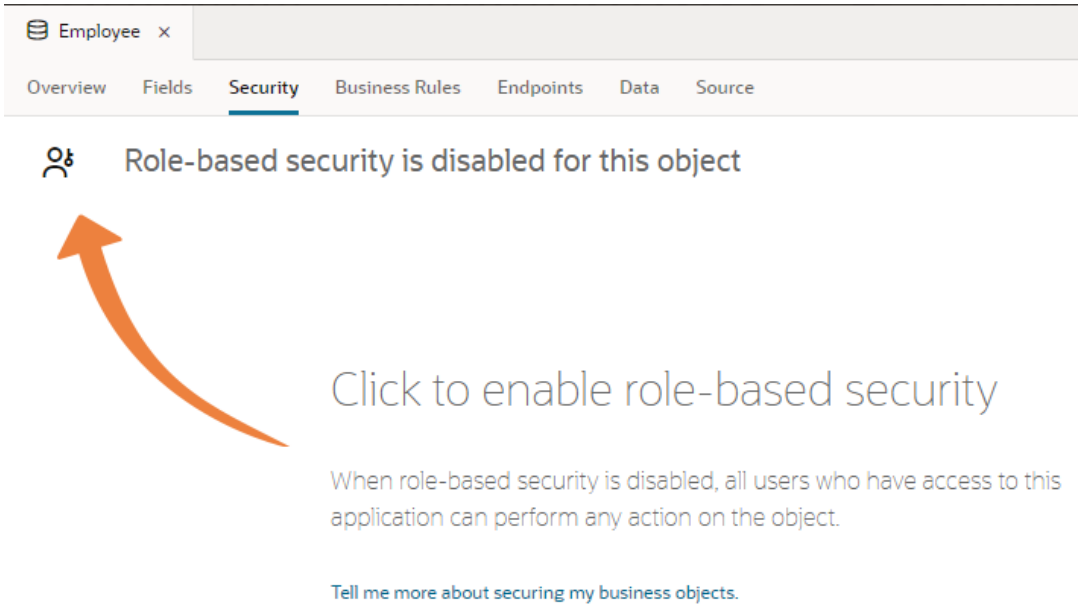
User roles can be used to secure the data stored in business objects.

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 user roles to restrict a user’s access to view, create, update, and delete operations by configuring role-based access for each operation. 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.

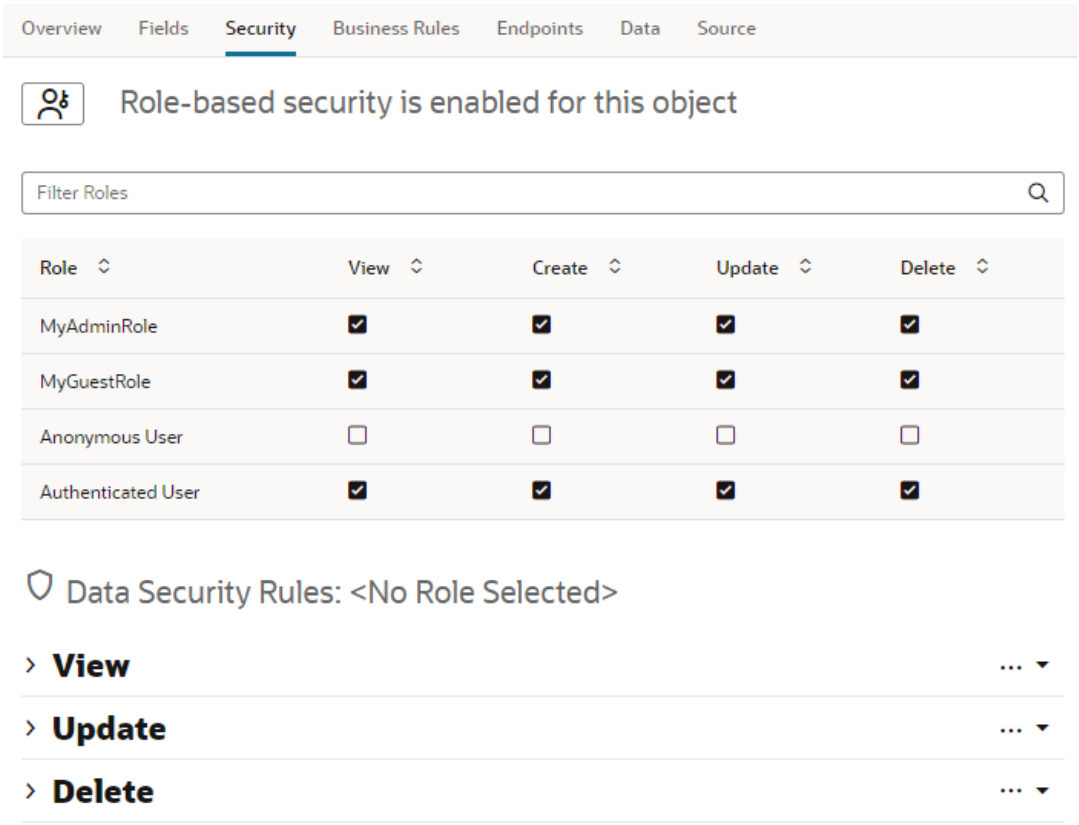
To enable role-based security for a business object:

1. Select the business object you want to secure.
2. Open the **Security** tab of the business object.

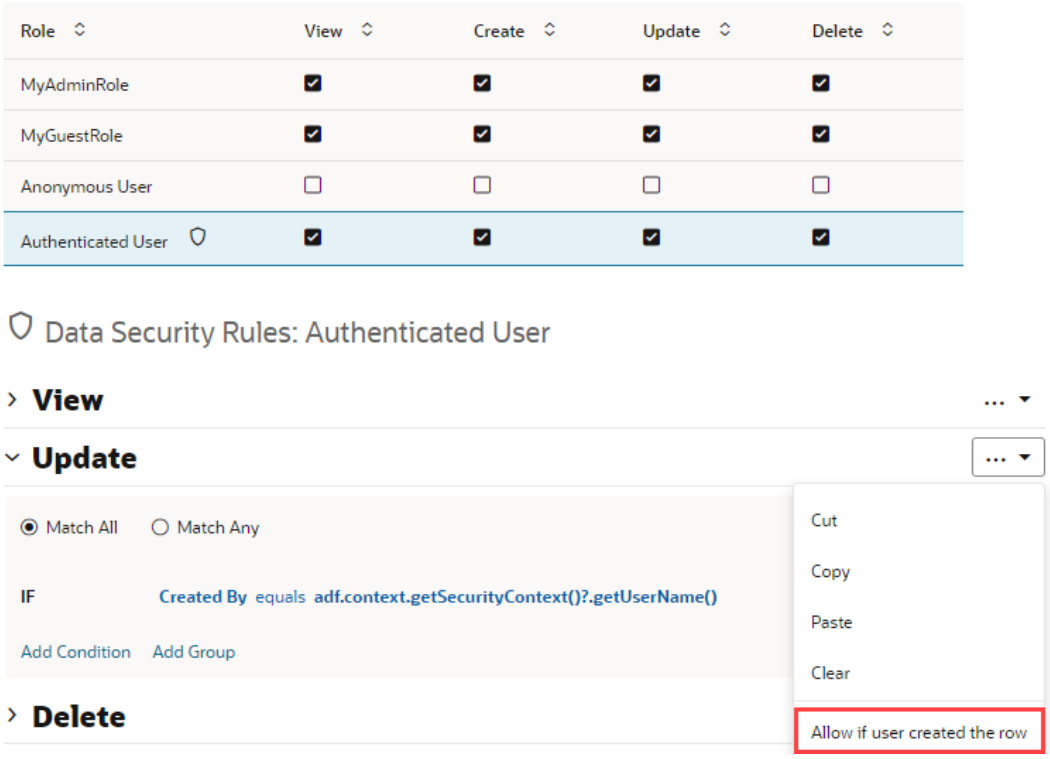


3. Click the **Role-based security** icon to enable security for the object.

   When you enable role-based security for a business object, you see a matrix of the existing user roles and the business operations that can be performed. By default, when you enable security, all existing user roles are permitted to perform all operations. If you create a new user role (see Manage User Roles and Access), permissions to perform operations are disabled for the new role and must be enabled manually.
4. Select the operations that can be performed by each authentication and user role. You can enable or disable permission for each operation.



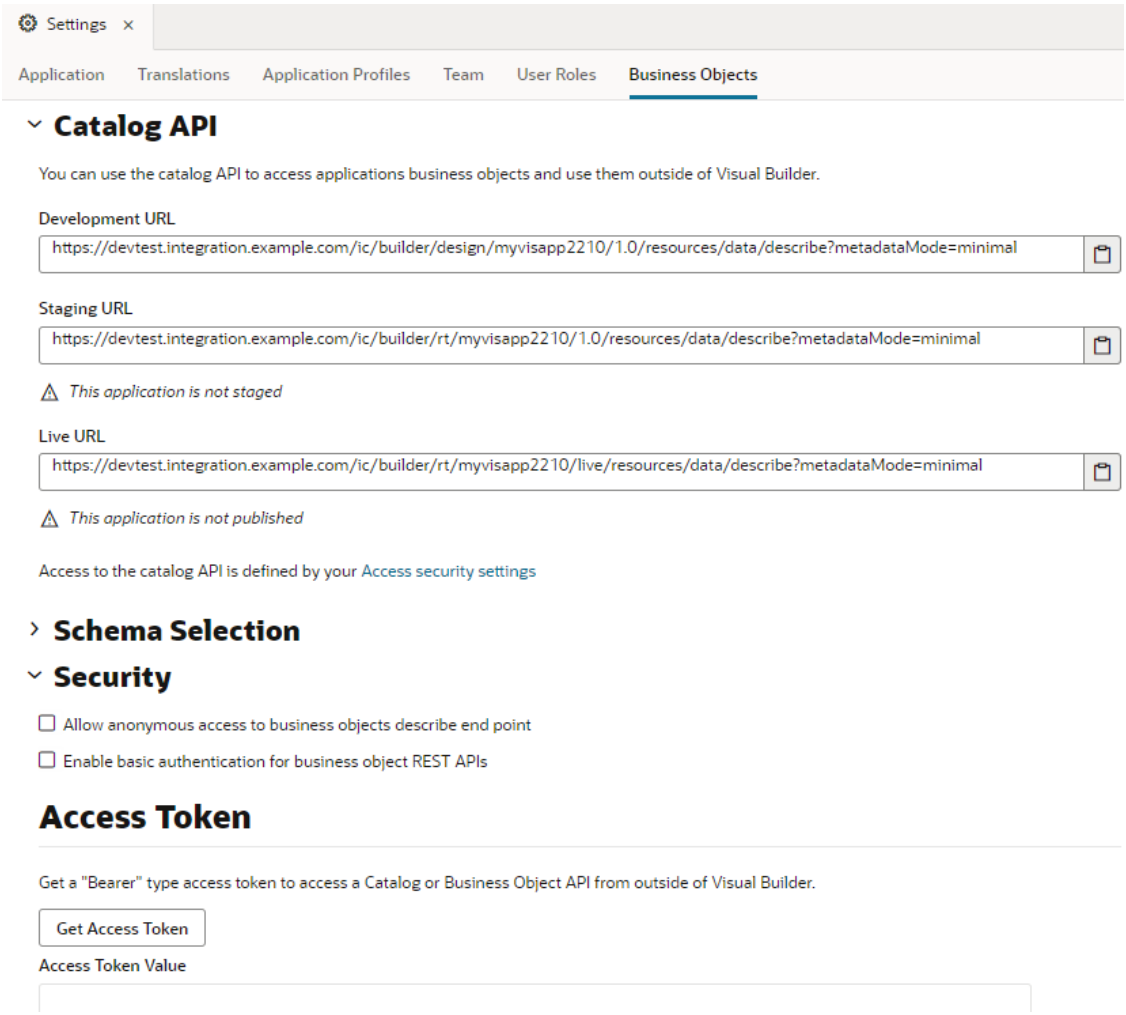
You can further define security at the row level for View, Update, and Delete operations by using a query builder to define conditions. To specify which users the conditions apply to, select the user role in the table. You can select **Allow if user created the row** from the action menu to limit an operation to the user who created the row. The menu also has **Cut** and **Copy** options for you to move conditions from one role or operation to another.



Allow External Access to Your Business Objects

When configuring security, you can allow external clients to access the business objects in your application through their REST endpoints. For example, you might want an external service like Process Automation to update a business object after a process is complete, say change a status field from "requested" to "approved".

To do this, you need to retrieve the API for the catalog of endpoints exposed by your application's business objects, found in the **Catalog API** panel in the Business Objects tab of your application's Settings editor:



The Development, Staging, and Live versions of your visual application each have their own catalog APIs. 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.

> **Tip:**
>
> For each URL, click the Clipboard icon to quickly copy the URL to your clipboard.

Accessing the catalog APIs requires authentication. To set up security options for allowing access to the business object APIs:

1. Open the Business Objects tab in the visual application's Settings editor.
2. Under **Security**, select an authentication option:
   - **Allow anonymous access to business objects describe end point**
   - **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. Here's how you can add the header for requests from external clients:

- Include `auth` in the Describe endpoint URL, for example:
  `https://servicename-cloudaccount.test.oraclecloud.com/ic/builder/rt/myapp/1.0/resources/auth/data/describe?metadataMode=minimal`
- Add the “Authorization: Public” header to the request, for example, from the cURL command line:
  `curl -v https://servicename-cloudaccount.test.oraclecloud.com/ic/builder/rt/myapp/1.0/resources/data/describe?metadataMode=minimal -H 'Authorization: Public'`

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 allowlist 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 allowlist. 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 business objects from outside 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 to read the data in the app's business object.

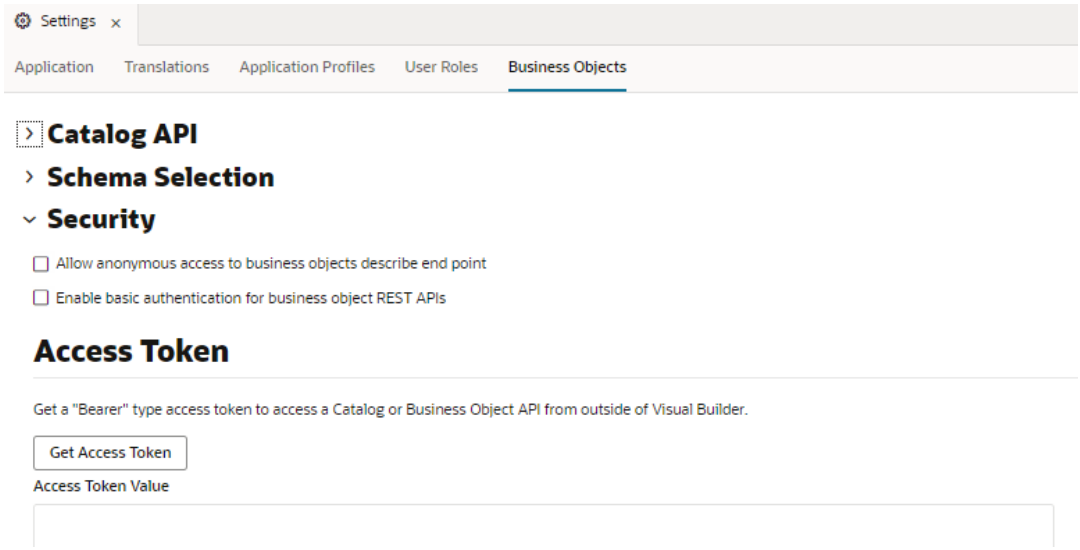
When authentication is handled by IDCS, you can use the token with connections that are authenticated with OAuth using the following authentication methods:

- Oracle Cloud Account
- User Assertion OAuth 2.0
- Client Credentials OAuth 2.0
- Resource Owner OAuth 2.0

You cannot use the token with connections to Oracle Cloud Applications.

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.

Allow Anonymous Access

Visual Builder applications, by default, require authentication; all users must sign in with their Oracle Cloud credentials to access your app. If you want users to access your app without signing in, you can enable anonymous access from the app-level Settings editor.

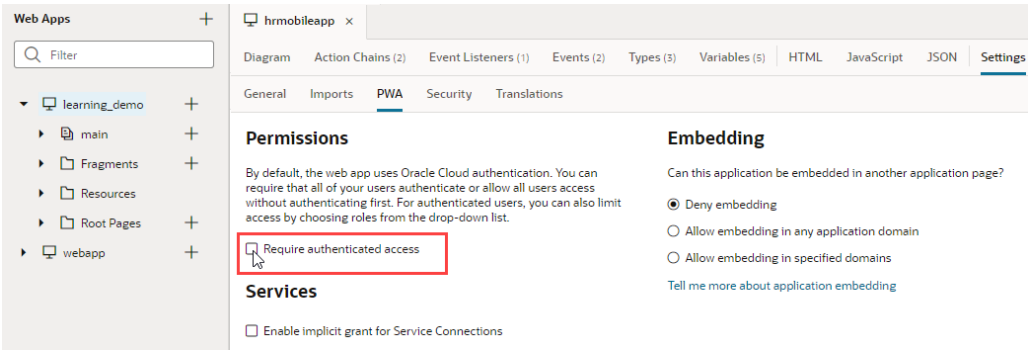
> **Note:**
>
> The service administrator must enable anonymous access in the instance's Tenant settings. You will not be able to enable anonymous access for your visual applications if anonymous access to applications is not permitted for the instance.

When anonymous access is enabled, users are not required to sign in and are automatically assigned the **Anonymous User** authentication role. By default, users assigned this role cannot access the data stored in your visual application's business objects or retrieved from services. You must explicitly allow anonymous users access to this data by configuring the security settings of business objects, backends, and service connections. You also need to allow anonymous access to the Describe endpoint for your business objects.

> **Note:**
>
> If you allow anonymous access to a backend and its service connections by specifying credentials for anonymous access, be aware that those backend systems will be accessible through a public URL without any authentication. If you need anonymous access for a backend or service connection, consider:
>
> - Creating a backend specifically for the service connections to which you want to allow anonymous access (allowing anonymous access is inherited by a backend's service connections).
> - Adding only the minimal credentials required to access just those service connections (for example, read-only credentials).

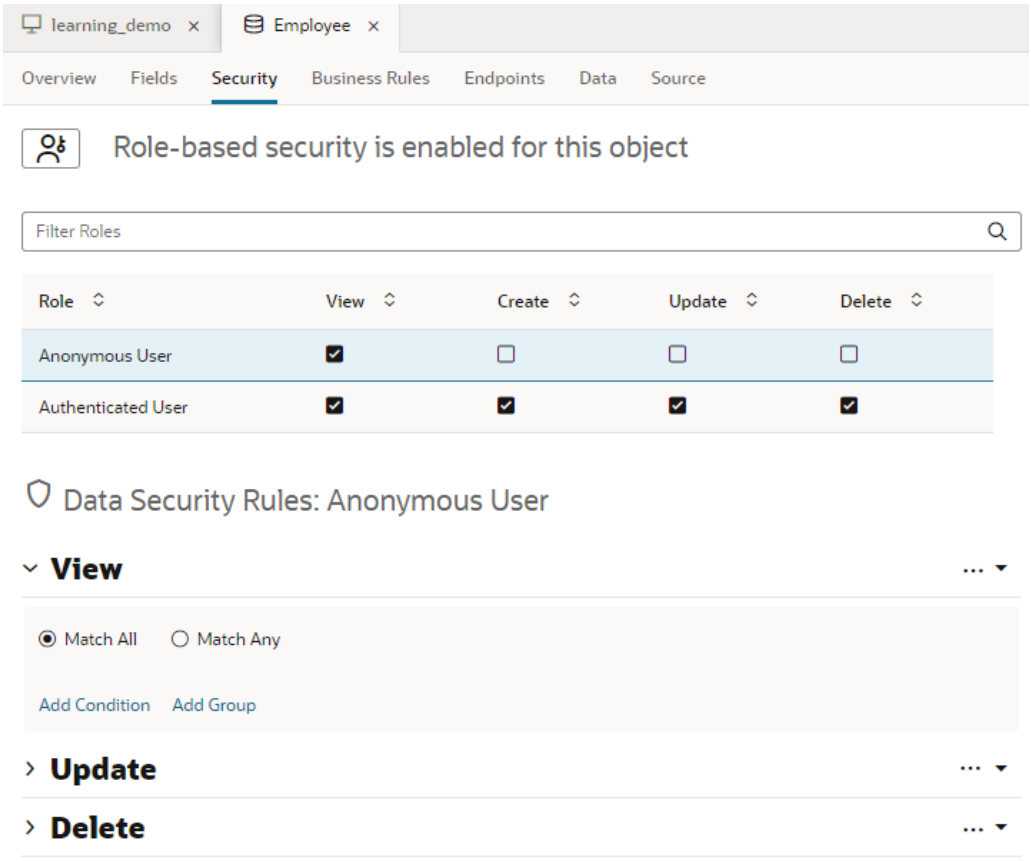
Changes that you make to authentication and security settings are applied only when you stage or publish the application. The versions of your application that are currently staged or published are unaffected. For example, if your application is already published, you must create a new version of the application, change its settings to allow anonymous access, then stage or publish the application again for the new security settings to take effect.

1. To enable users to access your visual app without signing in, enable anonymous access in the app's Security tab:
   a. Open your web (or mobile) application in the Navigator.
   b. Open the application artifact and click **Settings**, then **Security**.
   c. Deselect **Require authenticated access** under Permissions.



   With anonymous access enabled, users don't need to sign in to access the app.

2. To allow anonymous users access to the visual application's data stored in business objects, enable role-based security in the business object's Security tab and specify the operations that the Anonymous User authentication role can perform:
   a. Open your business object's **Security** tab.
   b. Click the **Role-based security** icon (if not enabled).
   c. Configure the rights granted to users assigned the Anonymous User role.



With anonymous access enabled, anonymous users can perform operations on business objects based on the permissions granted to the Anonymous User authentication role.

3. To allow anonymous access to service connection data, enable and specify the authentication mechanism for anonymous access in the service connection's server details:

   a. Open your service connection's **Servers** tab and edit the server details.

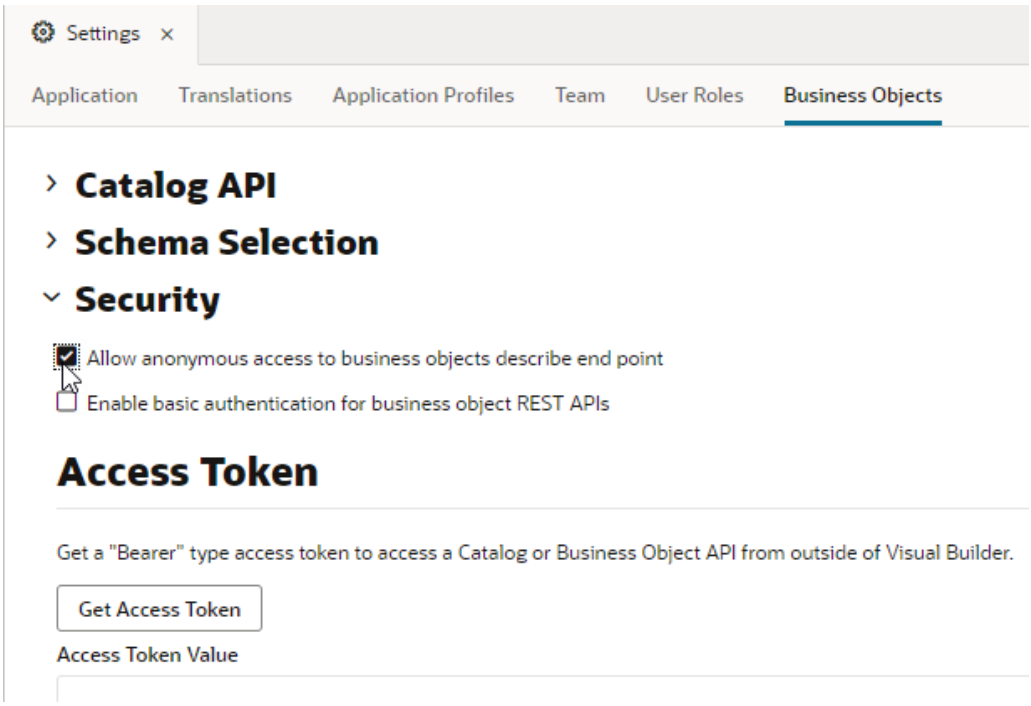
   b. Select **Allow anonymous access to the service connection infrastructure** under Security.

      If the option is grayed out, click **Override Security** to override security inherited from the backend, then select **Allow anonymous access to the service connection infrastructure**.

   c. From the **Authentication for Anonymous Users** drop-down list, select the authentication mechanism you want to use.

With anonymous access enabled, anonymous users can access data from the service connections that are configured to allow anonymous access.

4. Applications that allow anonymous access and have business objects with anonymous access must explicitly allow anonymous access to the business object's Describe endpoint:
   a. Open the **Business Objects** tab of the visual application's Settings editor.
   b. Select **Allow anonymous access to business objects describe end point**.



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. Here's how you can add the header to the request from external clients:

- Include `auth` in the Describe endpoint URL, for example:
  `https://servicename-cloudaccount.test.oraclecloud.com/ic/builder/rt/myapp/1.0/resources/auth/data/describe?metadataMode=minimal`
- Add the “Authorization: Public” header to the request, for example, from the cURL command line:
  `curl -v https://servicename-cloudaccount.test.oraclecloud.com/ic/builder/rt/myapp/1.0/resources/data/describe?metadataMode=minimal -H 'Authorization: Public'`

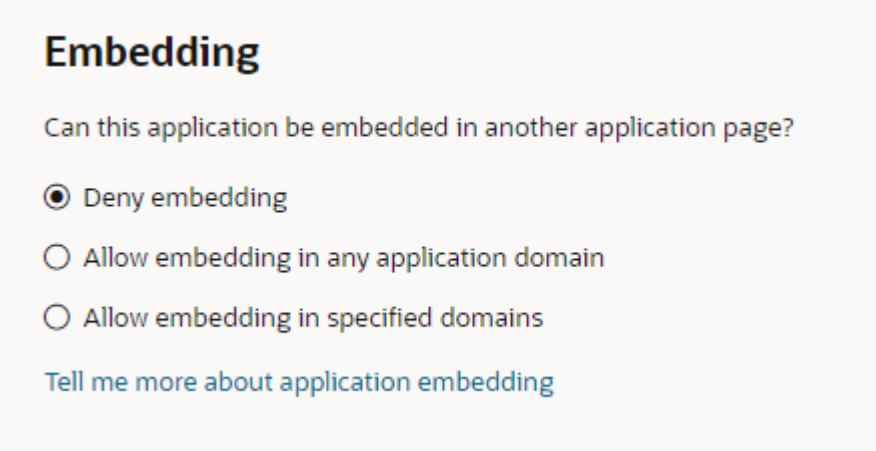
Embed a Web Application

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 app-level Settings editor to change this setting:



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.
3. Click **Settings**, then **Security**.
4. In the Embedding section, select **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.

You can also embed a web app in an Oracle Cloud Application, but there's more to it than just allowing access. See Embed a Web App in an Oracle Cloud Application for details.

19

Add Offline Capabilities to Your Application

Applications that you create in Visual Builder can function even when your device is disconnected from the network. To do this, you can use the Oracle Offline Persistence Toolkit which enables your application to cache data on the client for offline support.

If you use business objects to shape your application's data, you can leverage the object's caching settings to cache data read from a business object on the client. For more information, see Control Data Caching for Business Objects.

Add Offline Support Using the Oracle Offline Persistence Kit

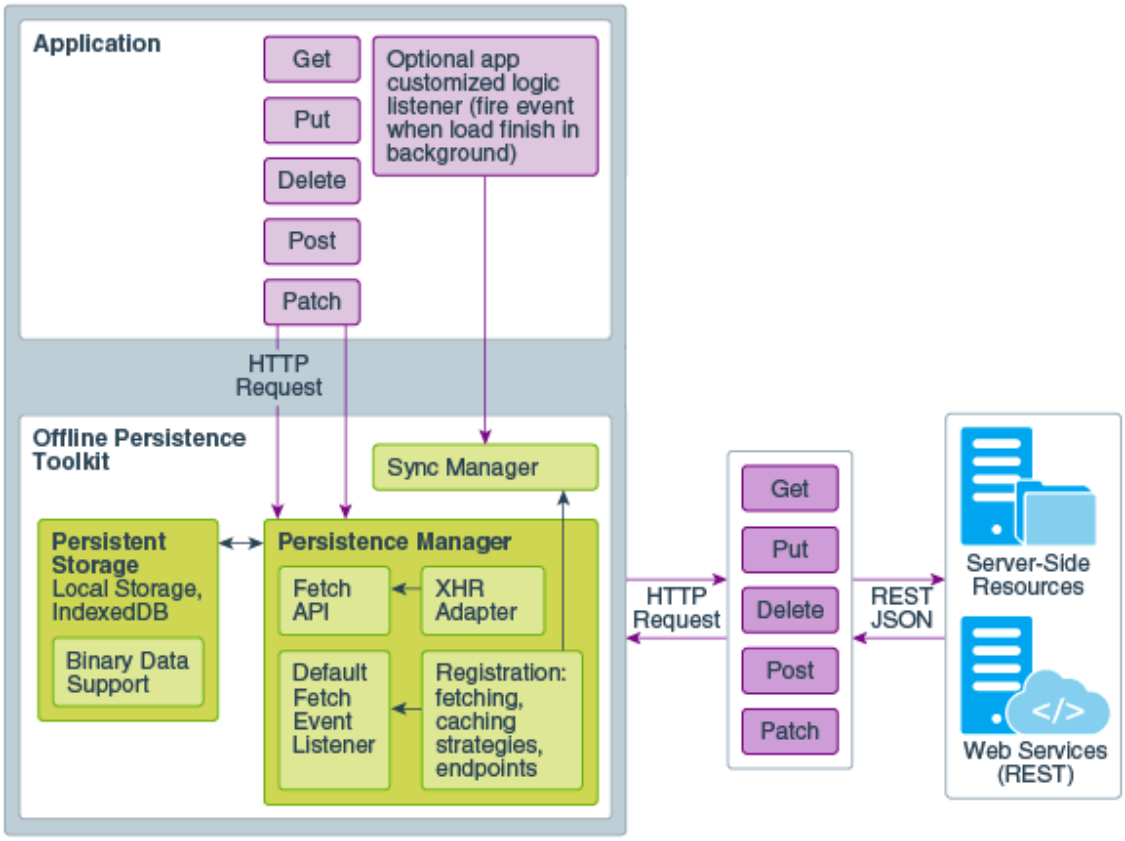
The Oracle Offline Persistence Toolkit is a client-side JavaScript library that helps to provide offline support for your application.

It enables caching for 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 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:



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.

> **Note:**
>
> Data caching for business objects is disabled by default, with each object's Resource Cache Control setting defined as `Sensitive`. This default option combines the `no-cache` and `no-store` values to disable data caching. Before you use the Oracle Offline Persistence Toolkit to enable data caching for offline support, check the data caching strategy used by your application's business objects. See Define a Data Caching Strategy.

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 for offline capabilities in your application. The file also demonstrates how you enable toolkit's logging functionality 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.

```
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.
     *
     */
    'persist/persistenceManager',
    'persist/defaultResponseProxy',
    'persist/fetchStrategies',
    /**
     * Add the following entry to enable console logging while you develop
your app with the toolkit.
     */
    'persist/impl/logger'
],
    (ServiceWorkerHelpers, PersistenceManager, DefaultResponseProxy,
FetchStrategies, Logger) => {
        'use strict';

        class AppModule {

        }

        var OfflineHandler = function () {

            /**
             * 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 +
                ' 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/ifixitfaster/api')) {

                return this._responseProxy.processRequest(request);
            }
            return PersistenceManager.browserFetch(request);
        };

        OfflineHandler.prototype.beforeSyncRequestListener = (event) => {
            return Promise.resolve();
        };
        OfflineHandler.prototype.afterSyncRequestListener = (event) => {
            return Promise.resolve();
        };
        AppModule.prototype.createOfflineHandler = () => {
            /** Create the OfflineHandler that makes the toolkit cache data
URLs */
            return Promise.resolve(new OfflineHandler());
        };
        AppModule.prototype.isOnline = () => {
            return ServiceWorkerHelpers.isOnline();
        };
        AppModule.prototype.forceOffline = (flag) => {
            return ServiceWorkerHelpers.forceOffline(flag).then(function () {
                /** if online, perform a data sync */
                if (!flag) {
                    return ServiceWorkerHelpers.syncOfflineData();
                }
                return Promise.resolve();

            }).catch(function (error) {
                console.error(error);
            });
        };
        return AppModule;
    });
```

Oracle maintains the persistence 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.

20

Optimize Your App for Search Engines

Search Engine Optimization (SEO) covers a variety of techniques that make your application's pages more accessible to web crawlers, the scripts used by search engines to crawl the web and gather pages for indexing. SEO looks to increase the ranking of indexed pages in search results to give your content more visibility.

You can often improve the ranking of your application's pages using sitemaps. Sitemaps list the URLs of your pages, enabling web crawlers to identify your content without relying solely on their ability to crawl and parse the pages. But while sitemaps can help web crawlers find pages for indexing, the indexing process itself might be less than ideal for single-page applications. This is especially true when content is dynamically added to pages based on the results of REST calls. In such scenarios, web crawlers may not wait long enough for the REST calls to complete and the page to be fully rendered before indexing the page. Pages that are indexed before their content can be fully loaded don't feature well in search results.

To address this issue, Visual Builder "prerenders" pages that web crawlers access. When a request for a page is received from Googlebot (or any other search engine), the request is routed through a prerender server, which loads the page, runs any JavaScript required to fully render that page, then strips it out before returning the page to Googlebot. This way, Googlebot always receives a fully rendered page for indexing—just one without any JavaScript in the source.

A page, once prerendered, is always cached, so page markup can be returned immediately when subsequent requests are made for the same URL. This helps reduce page load times that may adversely impact page ranking. If markup is yet to be cached, the process of rendering the page can take some time. To optimize these response times, it's recommended that you warm the prerender server's cache for URLs listed in your application's sitemap, so web crawlers always get the fastest possible response when requesting those URLs.

Here's a recommended list of actions to optimize your app for search engines:

1. Create a sitemap for your web application.
2. Add the sitemap to your web application's resources.
3. Once your web application is deployed (or redeployed), warm the cache for URLs listed in your app's sitemap (using the `vb-prerender-cache-warm` Grunt task).

Create a Sitemap for a Web App

A sitemap is a document that contains URLs to pages representative of your application. It ensures that important locations in your application are possible for web crawlers to locate and are properly ranked.

A simple sitemap may look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
   <url>
      <loc>https://eta.myexample.org/?page=shell</loc>
      <lastmod>2023-08-21T16:12:20+03:00</lastmod>
   </url>
```

```
   <url>
      <loc>https://eta.myexample.org/?page=shell&shell=bugs&bugs=main-bug</
loc>
      <lastmod>2023-08-21T16:12:20+03:00</lastmod>
   </url>
   <url>
      <loc>https://eta.myexample.org/?page=shell&shell=tags</loc>
      <lastmod>2023-08-21T16:12:20+03:00</lastmod>
   </url>
</urlset>
```

Each `<url>` tag is used to specify the URL of a page. This tag has several child tags, but only the `<loc>` and `<lastmod>` tags are used to populate the prerender server's cache:

- `<loc>` is a required tag that specifies the actual URL of a page. The value of a `<loc>` tag must begin with the protocol (such as https) and end with a trailing slash, if your web server requires it. This value must be less than 2,048 characters long.
- `<lastmod>` is a required tag that specifies the date the contents of the URL changed between application deployments.

Make sure the sitemap contains each individual URL you want indexed. It should also include up-to-date `lastmod` entries if the contents of the URL can change between deployments (for example, when pages contain data from other sources).

For general information on sitemaps, see https://www.sitemaps.org/protocol.html.

Add a Sitemap to a Web App's Resources

You can add a sitemap to your web application's resources to provide web crawlers information about the content in the application.

Before you add a sitemap file to your web app's resources, make sure it contains each URL you want indexed as well as up-to-date information for each URL. See Create a Sitemap for a Web App.
To add a sitemap to your web application:

1. Upload the `sitemap` file (for example, `sitemap.xml` or `sitemap.txt`) to your application in Visual Builder.

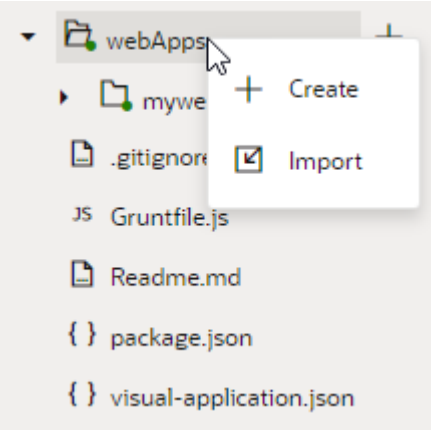
   You can upload the `sitemap` file to a web application's directory (for example, `/webApps/mywebapp/sitemap.xml`) or directly to the root directory of your visual application (at the same level as `Readme.md` or `visual-application.json`). Note that the root `/webApps` directory has higher priority and uploading a sitemap file here should work for all your web applications.

   To upload a sitemap file at the root of your visual application:

   a. Open your web application in the Navigator.

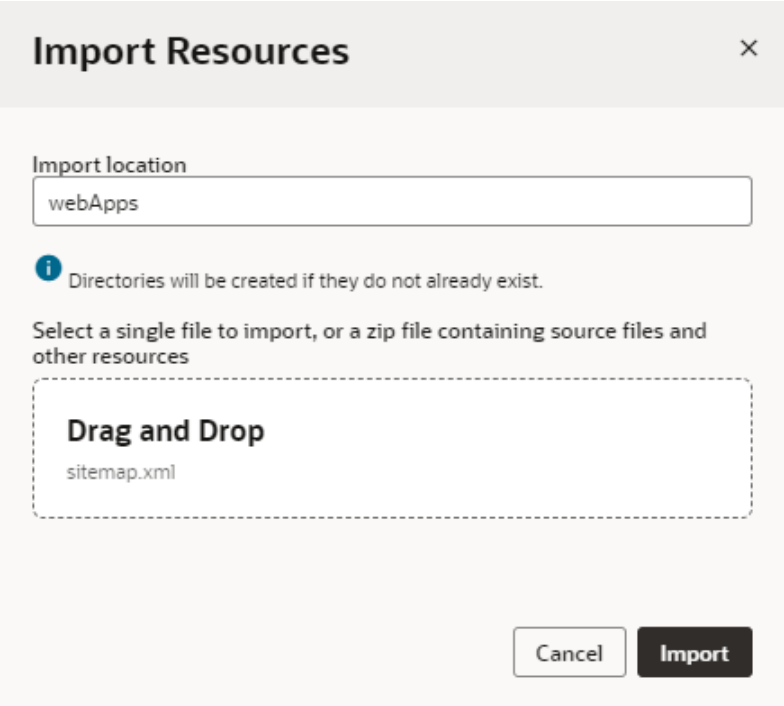
   b. Click **Source View**.

   c. Right-click the **webApps** directory and choose **Import** in the pop-up menu:



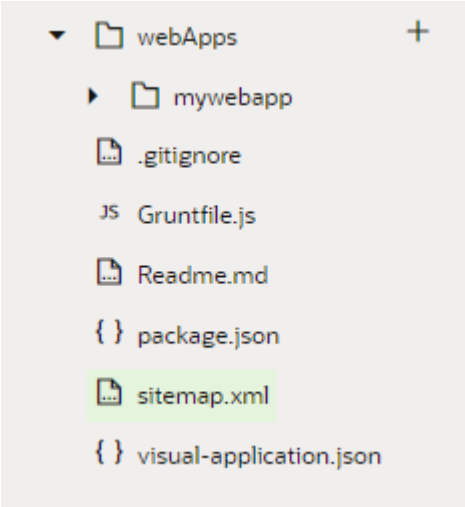
Alternatively, drag a file from your local file system onto the **webApps** directory in the Navigator.

d. In the Import Resources dialog box, remove `webApps` in the **Import location** field, then click the drop target area and navigate to the file on your local system:

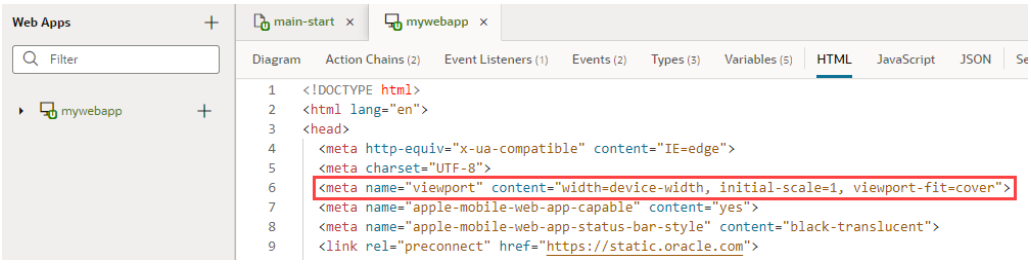


e. Click **Import**.

The sitemap is added to the root of your visual application:



2. Add the description/snippet for your application:
   a. Click **Web Applications** in the Navigator and open your application.
   b. Click **HTML** to open the application's `index.html` file.
   c. Update the `<meta name="description" content="description">` tag in the `<head>` section to add a description and a snippet that a web crawler will extract.



   Ensure that your description and snippet meet recommended guidelines, as described in Google documentation for snippets and meta-tags.

3. Stage and publish your application (see Stage and Publish Visual Applications).

   After your application is successfully staged and published, you can access the sitemap at the deployed location, for example, at:

   ```
   https://visualbuilder-dev.integration.test.ocp.oraclecloud.com/ic/
   builder/rt/myproject/live/webApps/mywebapp/
   version_324616315386523389/sitemap.xml
   ```

> **Note:**
>
> Each time you stage or publish your web app, the location of the resources changes to use a new version, and the sitemap file is moved to the newer `/webApps/webAppName/version_id` directory. To access the sitemap afterward, remember to use the new `version_id` in the resource path.
> If you want to automatically copy the sitemap file to the visual application's root directory each time the web app is staged or published, see Move Your Sitemap to a Visual Application's Root Directory.

Warm the Cache for URLs in a Sitemap

When your web app includes a sitemap that lists URLs to pages representative of your application, you can use the `vb-prerender-cache-warm` Grunt task to warm the prerender server's cache for these URLs, so web crawlers get the fastest possible response when requesting those URLs.

To use the `vb-prerender-cache-warm` Grunt task, you must set up your file system to build the visual application by installing `Node.js` and its package manager (`npm`). Once you have installed the necessary tools, you need to save the visual application's sources to your local system. See Build Your Application Locally.

To warm the cache for URLs in a sitemap using the `vb-prerender-cache-warm` Grunt task:

1. In the command-line interface, navigate to the folder on your local system containing the `package.json` and `Gruntfile.js` files.
2. Enter `npm install` to retrieve the node dependencies required to build the application.
3. Enter `vb-prerender-cache-warm` in the command-line interface to warm the cache. Here are some examples:
   - To initially warm the cache after a redeployment:

     ```
     grunt --id=MyApplication --ver=1.0 --sitemap="./webApps/myWebApp/
     sitemap.xml" --url=http://my.vbinstance.com/ic/builder --
     clearCache=true vb-prerender-cache-warm
     ```

   - To update the cache after minor modifications to the sitemap:

     ```
     grunt --id=MyApplication --ver=1.0 --sitemap="./webApps/myWebApp/
     sitemap.xml" --url=http://my.vbinstance.com/ic/builder vb-prerender-
     cache-warm
     ```

   The `vb-prerender-cache-warm` task reads the `manifest.json` file on startup. It will compare the timestamps of the pages in the manifest with the application's sitemap, and warm the cache with any files that have changed since the last run (or which failed on the last run) and write details of the run back to the same file on completion. If the `clearCache` flag is set, any existing manifest is ignored and overwritten when the task completes.

In the preceding examples, metadata about the last operation is saved to `manifest.json` in the current directory. To save or load from a different manifest file, you can specify the `--manifest` option:

```
grunt --id=MyApplication --ver=1.0 --sitemap="./webApps/myWebApp/
sitemap.xml" --url=http://my.vbinstance.com/ic/builder --manifest="/tmp/
manifest.json" vb-prerender-cache-warm
```

For more information on the supported command-line options, see vb-prerender-cache-warm.

Move Your Sitemap to a Visual Application's Root Directory

When you manually add a sitemap to your web application's resources, you can set up the sitemap file to be copied to the visual application's root directory after the app is staged or published, so the sitemap path does not change.

Each time a web application is staged or published, the location of the resources changes to use a new version, and the sitemap file is automatically moved to the new `/webApps/`*`webAppName`*`/version_id` directory. As a result, you'll need to include the newest `version_id` in the path to access the application's sitemap.
To automatically copy the sitemap file from the `/webApps/`*`webAppName`*`/version_id` directory to the application's root directory each time the app is staged or published, you can edit your application's `Gruntfile.js` file and add the `vb-post-package` task:

1. Open `Gruntfile.js`.
2. Edit the file to define the tasks performed for the `vb-post-package` task:

```
{code:java}
    /**
    Moves sitemap.xml resource back to Visual Application Root.
    */
    grunt.registerTask('vb-post-package', () => {
        const fileName = 'sitemap.txt'; // REPLACE WITH YOUR FILENAME
        const webAppName = 'mywebapp';  // REPLACE WITH YOUR WEB APP NAME
        const webAppDirectory = `build/optimized/webApps/${webAppName}`;
        // find out version dir
        const files = fs.readdirSync(webAppDirectory);
        const versionDir = files.find(file => /^version_\d+/.test(file));
        const source =
`${webAppDirectory}/${versionDir}/resources/${fileName}`;
        const target = `build/optimized/${fileName}`;
        // move the file back to assets root
        fs.renameSync(source, target);
        console.log(`${source} moved to ${target}`);
        });
{code}
```

Remember to update the task to use your sitemap resource and your web app.

See Customize Your Grunt Build Process for more information on adding custom functionality to existing Grunt tasks.

21

Work With Business Processes

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 used to complete user tasks.

> **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 Creating and Editing Instances in *Provisioning and Administering Oracle Integration Generation 2*.

> **Note:**
>
> - This chapter describes working with Oracle Process Cloud Service, which is included in the Enterprise edition of Oracle Integration Gen 2. This service is now deprecated; for details, see Process Features.
> - Oracle Process Cloud Service is **not** the same as OCI Process Automation. OCI Process Automation is a separate product.

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) application to initiate the process. You can also add process actions to action chains that can be triggered by events in your application.

Add a Business Process to an Application

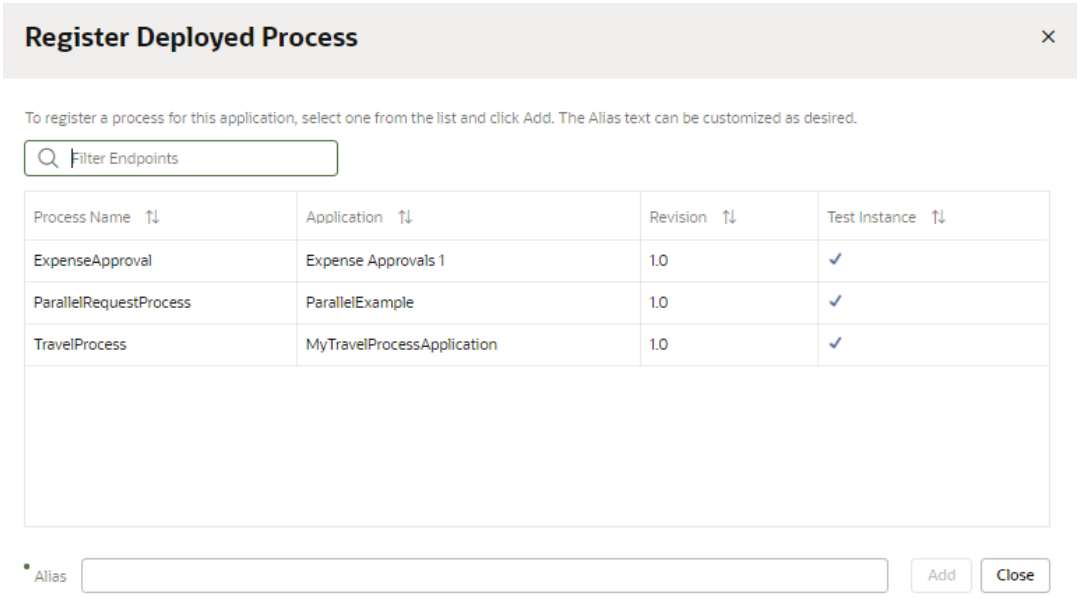
When adding a business process to an application, you can use business process aliases that are registered on the Process Server.

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.

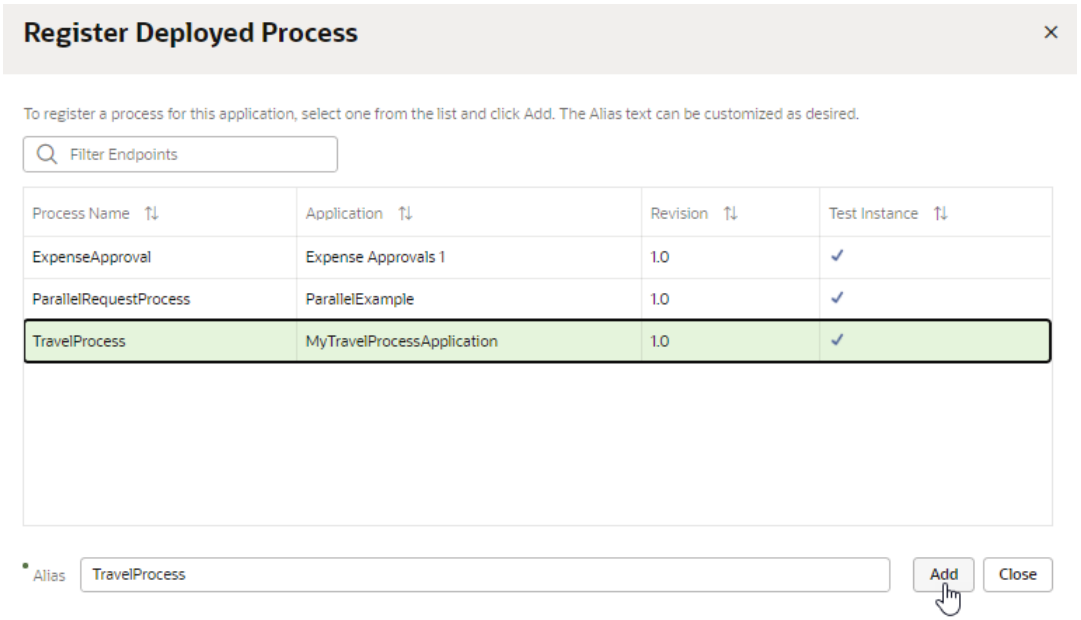
The process must be one that exists on the Processes in Oracle Integration instance.

To add a business process to an application:

1. Open **Processes** in the Navigator. As soon as you open Processes for the first time, Visual Builder configures a Process Server.
2. Click **+ Register Deployed Process** to open the Register Deployed Process dialog box.



3. Select the process you want to add.
4. Type the alias you want to use for the process, if you want to change it. Click **Add**.



5. 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.

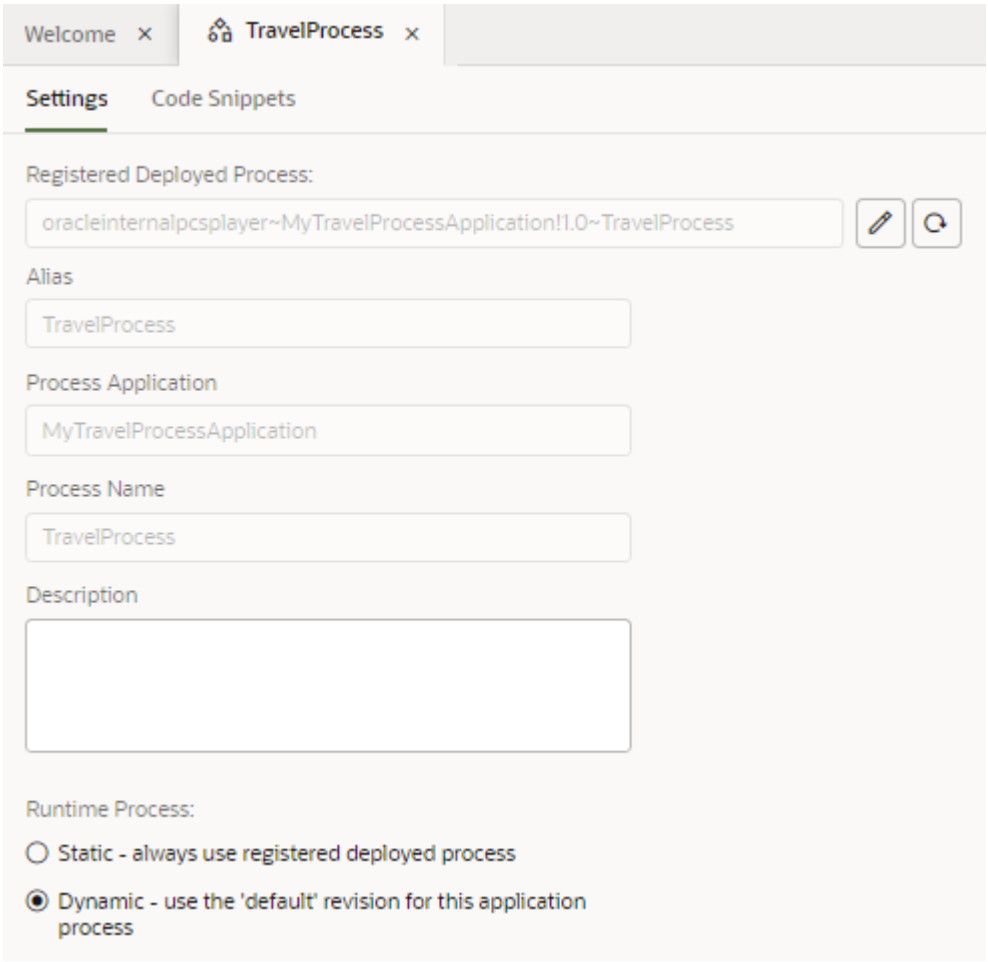
You can use the process editor to modify the target of your process alias to point to a different process and to automatically point to a default or more recent revision instead of a specific revision.

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:

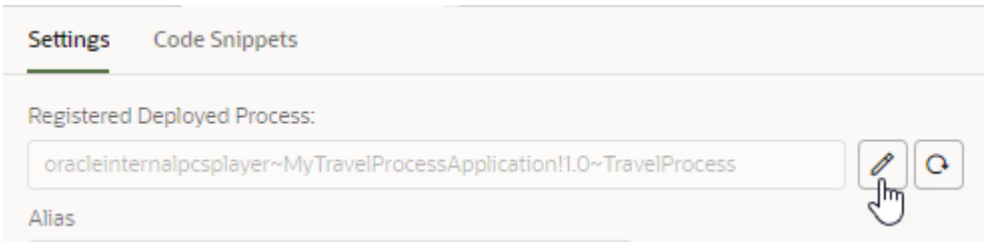
1. In the Processes pane, open the alias in the Process Alias editor.

   The Process Alias editor's Settings tab 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's Code Snippets tab shows the code that you will copy into pages in your application.



2. Click the Choose a different process icon ( ) 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.



   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 **Test Connection** to test the connection to your Process Server.

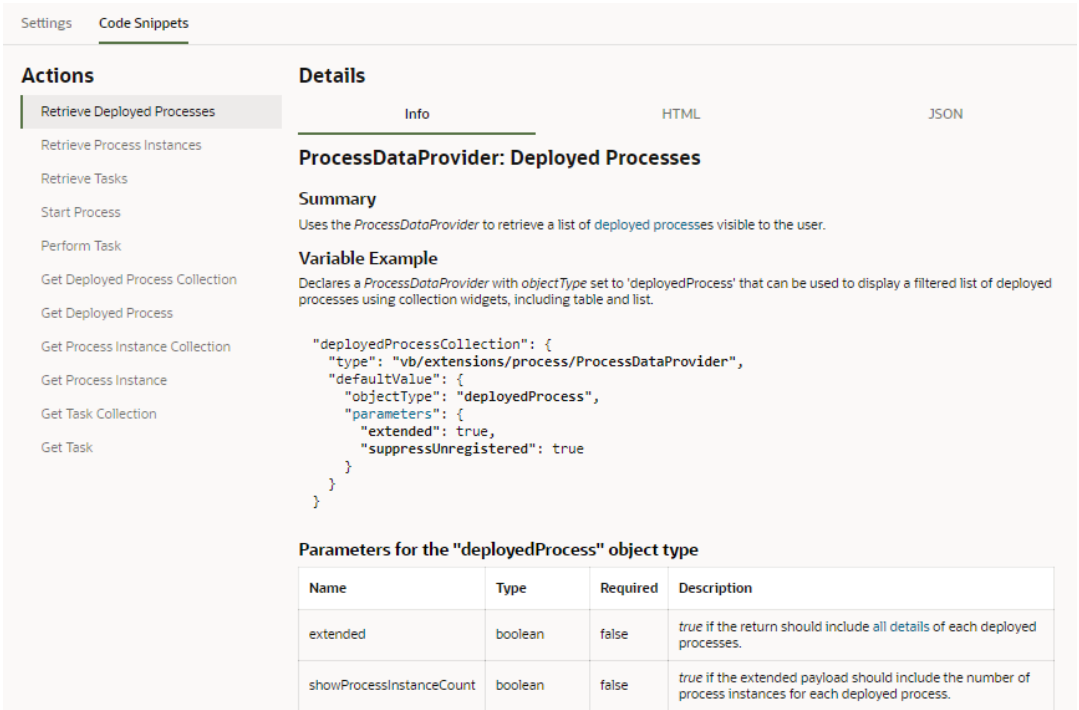
Add a Process to a Page

After you create 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 tab in the Process Alias editor contains a description and code used for each of the process's available operations.

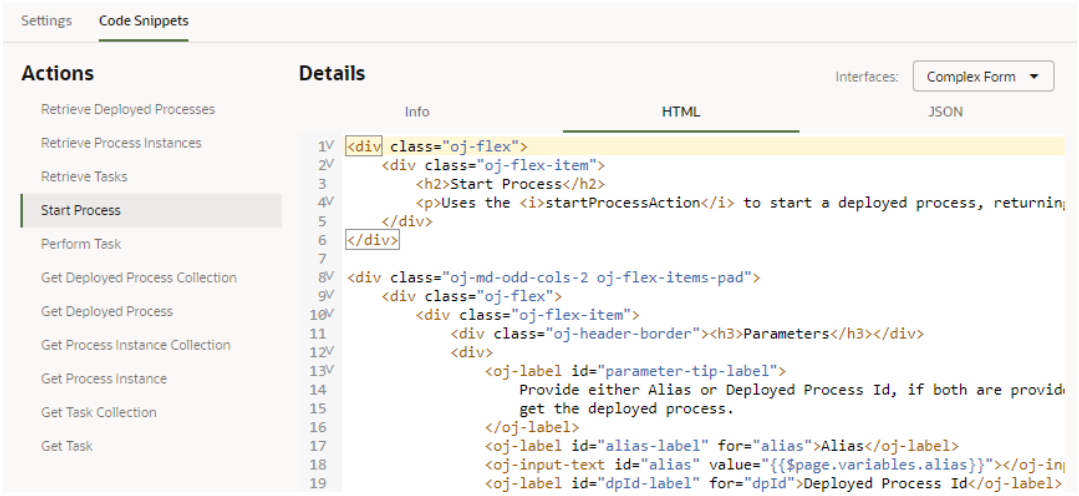
To use code snippets to add a process to a page:

1. Open the process you want to add in the Process Alias editor.
2. Click the **Code Snippets** tab and select the action you want to add.

   When you select an action, the Info tab contains details about the action, including descriptions of objects, and the properties, parameters, and outcomes of actions.

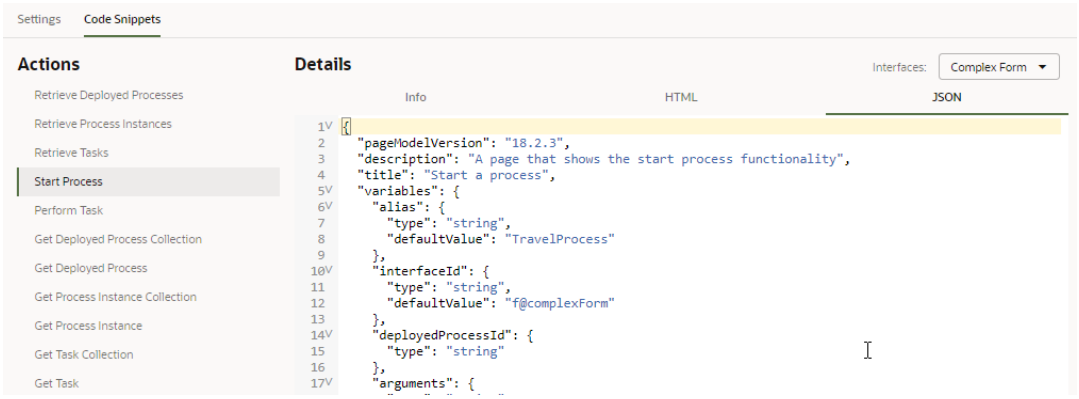


3. Click the **HTML** tab to copy the HTML source for the action.



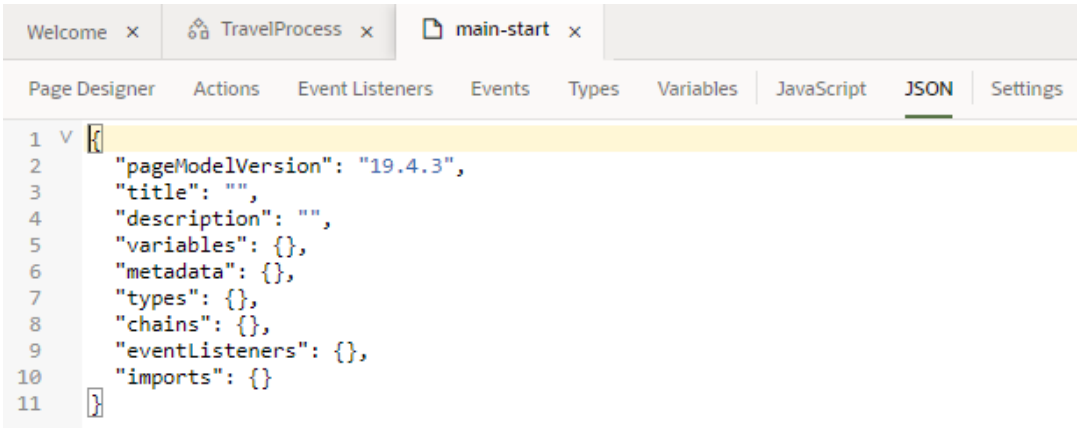
4. Open the page where you want to add the process in the Page Designer.
5. Click **Code** in the toolbar to open the page's HTML code in the editor.
6. Paste the HTML code you copied into the page's HTML editor.

   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, click the alias' **JSON** tab to copy the content.



8. Click the page's **JSON** tab 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 tab of the Process Alias editor. The Info tab in the Code Snippets tab 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 Properties pane. See Visually Create an Action Chain.

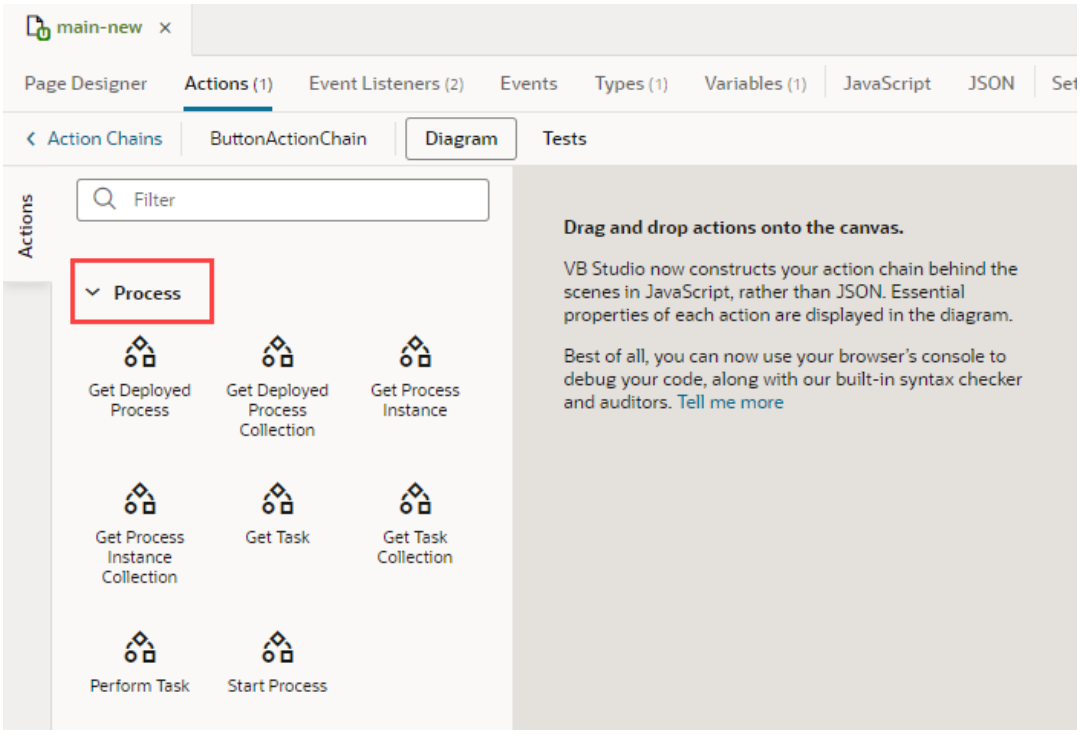
To add a process operation to an action chain:

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.

   The Process category in the Action Chain editor contains process actions. Each action is described in the Info tab in the Process Alias editor. See Add a Process to a Page.



3. Drag the action you want to add onto the Add icon ( ) in the action chain where you want to add the action.

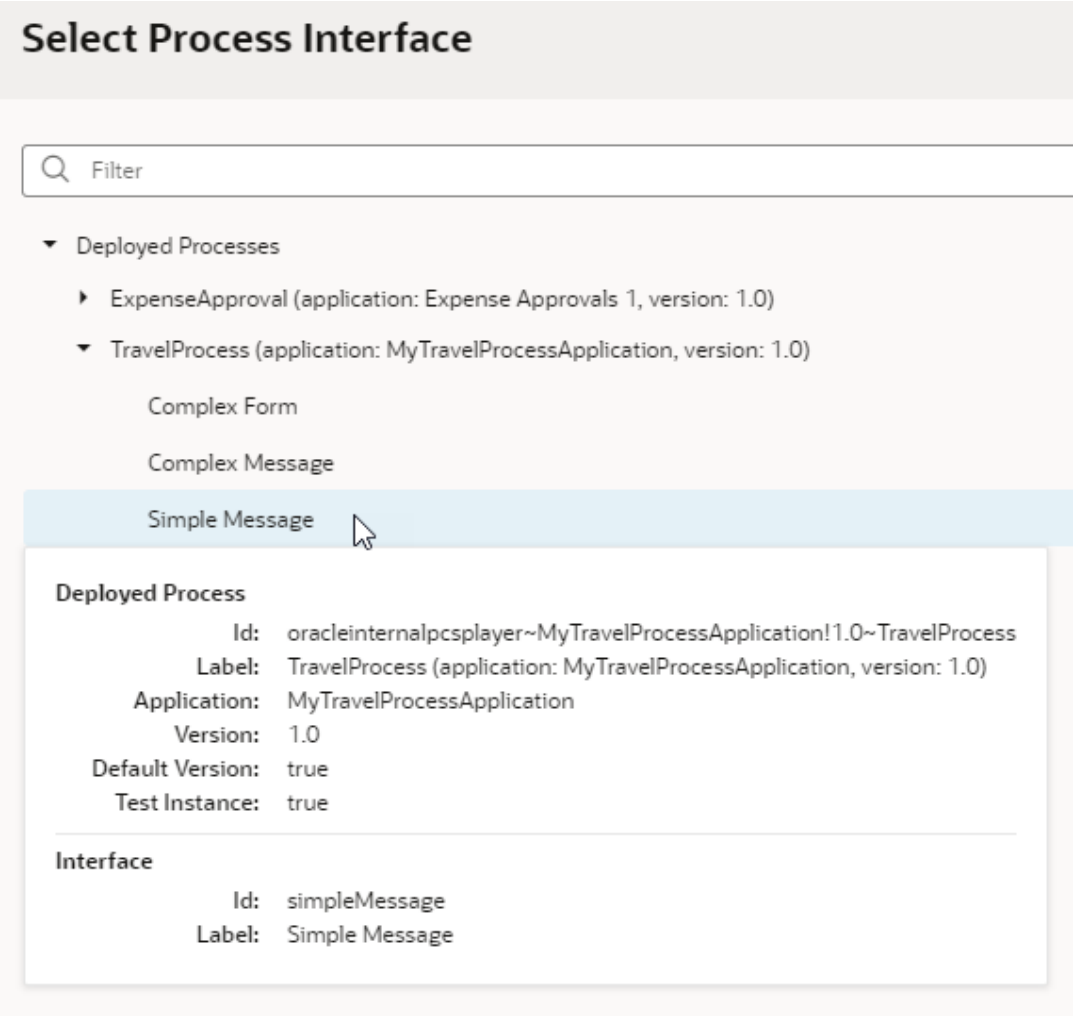
   The following steps apply if you select the Start Process action.

4. Click **Select** next to Process Interface in the Properties pane.

5. Select the process interface for the business process in the Select Process Interface dialog box. Click **Select**.

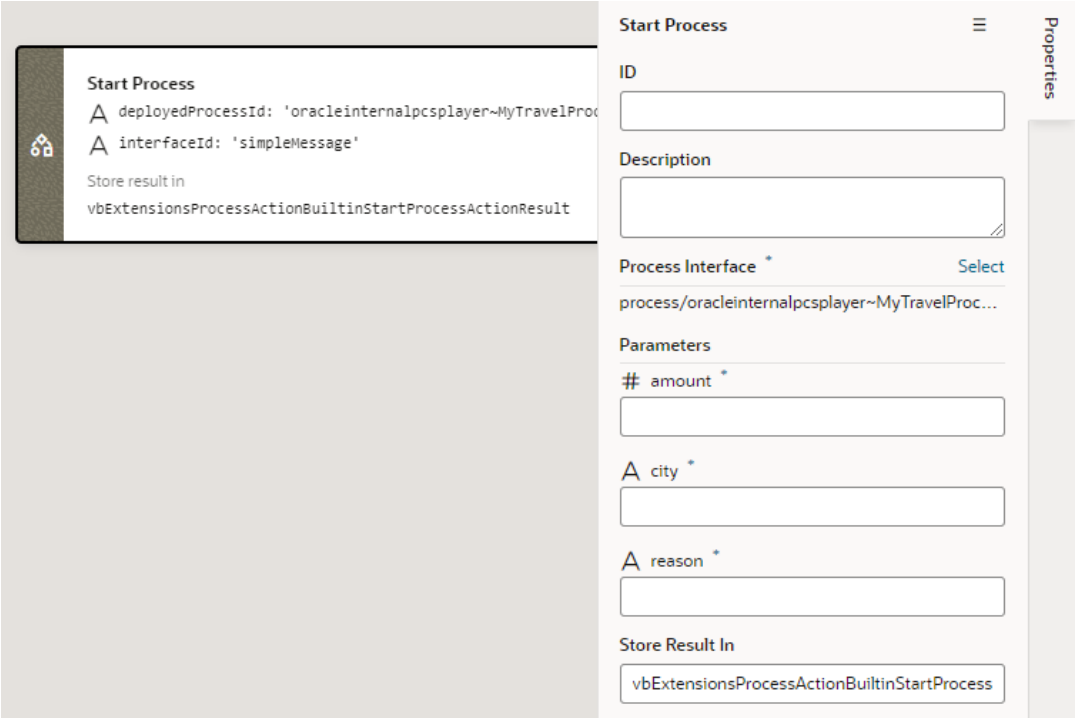
   The Select Process Interface dialog box lets you choose a process alias that you have created or a process definition available on the Process server.

   The business process that you select may have one or more process interfaces. Expand the process node in the dialog box to see the available interfaces.



6. Specify the interface's required properties and parameters in the Properties pane.

   Depending on the process interface for the process, you might need to map the interface's required input parameters to variables.



Start and Complete Processes from Visual Builder

To be able to start a process from 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

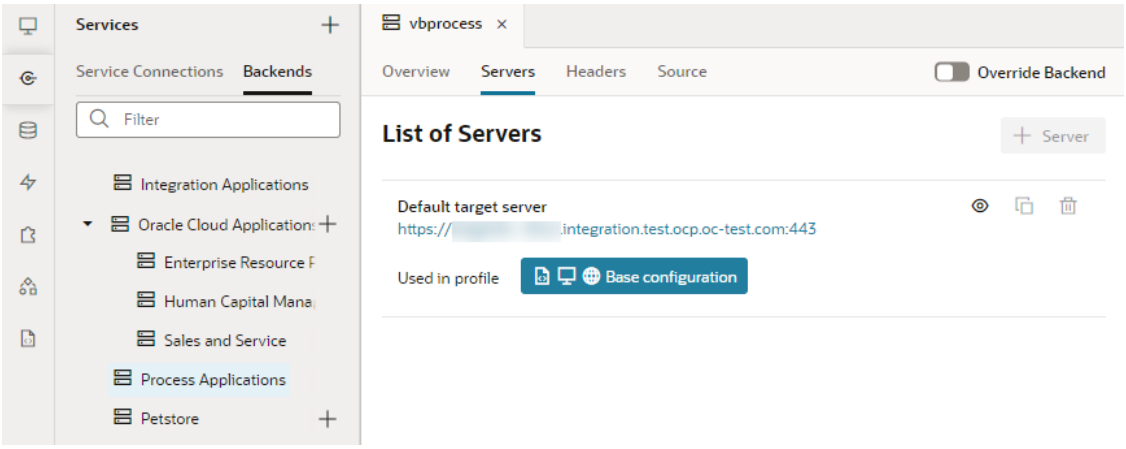
In Visual Builder, you can map the message parameters to page variables, business objects, or service connection endpoints. You can then set up forms and action chains that execute your processes, and tables or lists that display processes and tasks. An Add Task Actions Quick Start for lists and tables 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 Visual Builder.

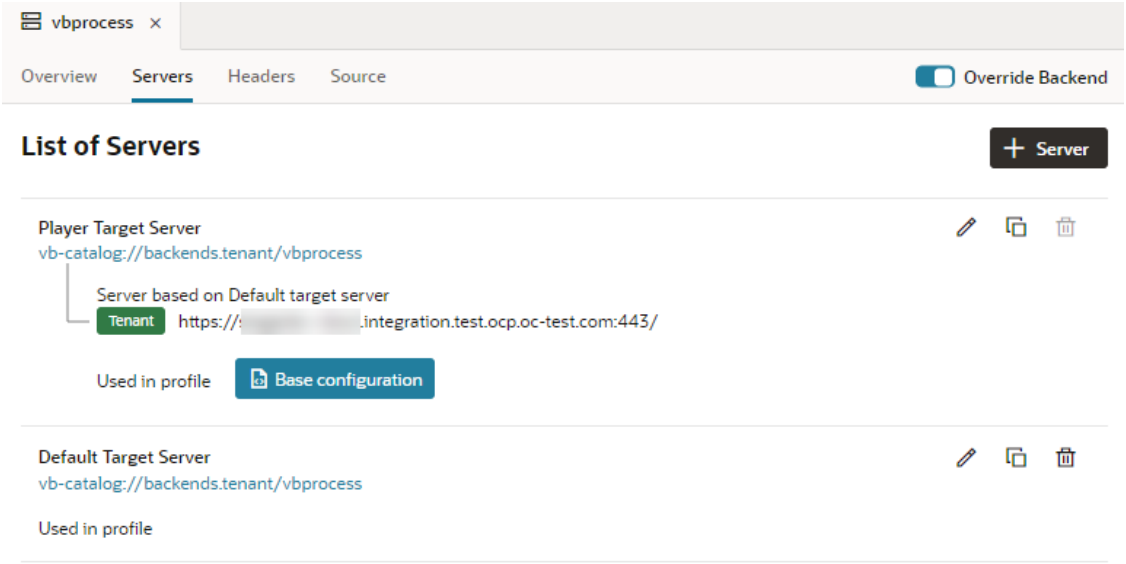
Configure the Connection to a Process Server

When developing applications that use business processes, you have considerable flexibility in configuring the settings used to connect to Process Servers and the credentials that are used when communicating with the processes on a Process Server.

Visual Builder automatically sets up a default Process Server configuration when you create an application in Visual Builder. To see the configuration, click the **Backends** tab in the Navigator's Services tab. In the **Backends** tab, click **Process Applications**, then **Servers**. You'll see a Default Target Server, which in development mode, is configured to point to a test environment on Processes in Oracle Integration instance.



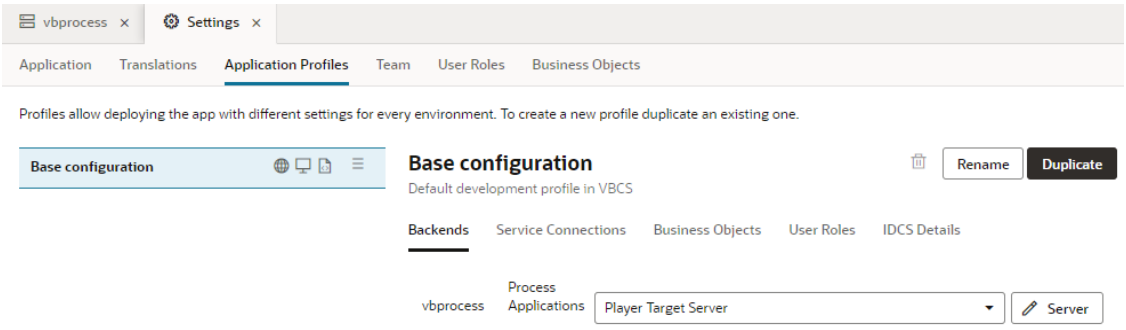
When a Process-related interaction occurs (say, you clicked **Processes** in the Navigator and tried to register a process, or used a Process Quick Start), the Tenant settings are overridden and you'll see another target server, the Player Target Server, as shown here:



The Player Target Server is useful when you want to "play" the process for testing, without making it live to your users.

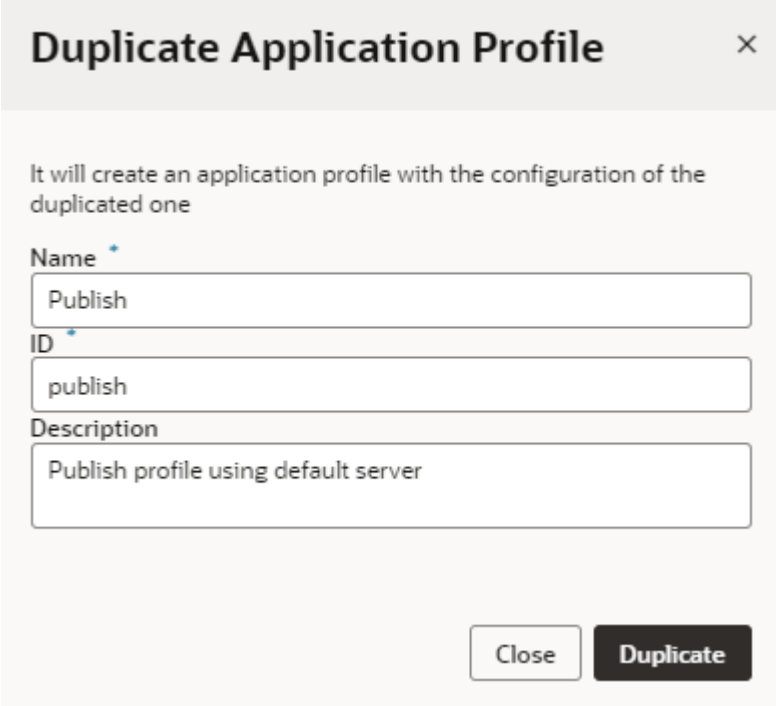
The target server is set up in a default Application Profile. For basic information on Application Profiles, see What Are Application Profiles?

When you click the **Application Profiles** tab under **Settings**, you can see a base configuration that is set up to use the Player Target Server. This configuration is the default for the publishing, staging, and development phases of application development.

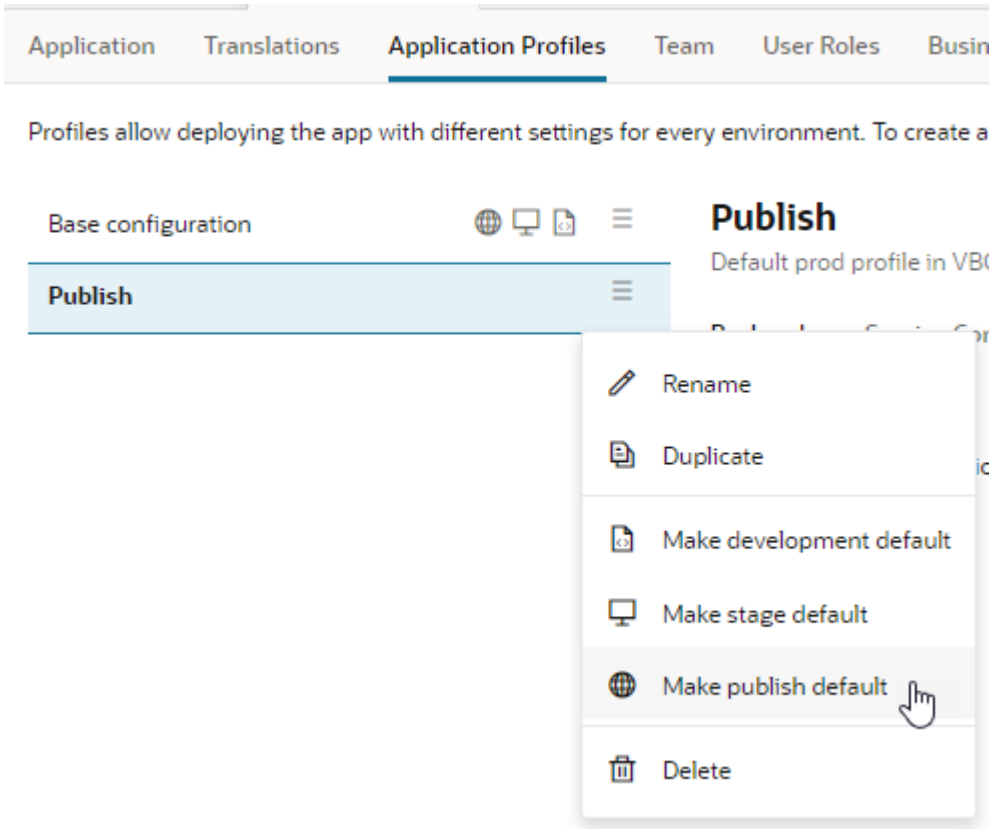


Once you finish developing your application, you'll want to duplicate the base configuration, rename it, and specify the use of the Default Target Server instead of the Player for the staging and/or publishing phases. To do this:

1. Click the **Duplicate** button and specify a new **Name** and **ID** (for example, `Publish`).



2. Click the **Publish** configuration and select **Default Target Server** from the drop-down list. By default, an application profile is always connected to the Player Target Server to avoid accidentally kicking off a live process.
3. From the configuration's Options menu (☰), select **Make stage default** or **Make publish default**.



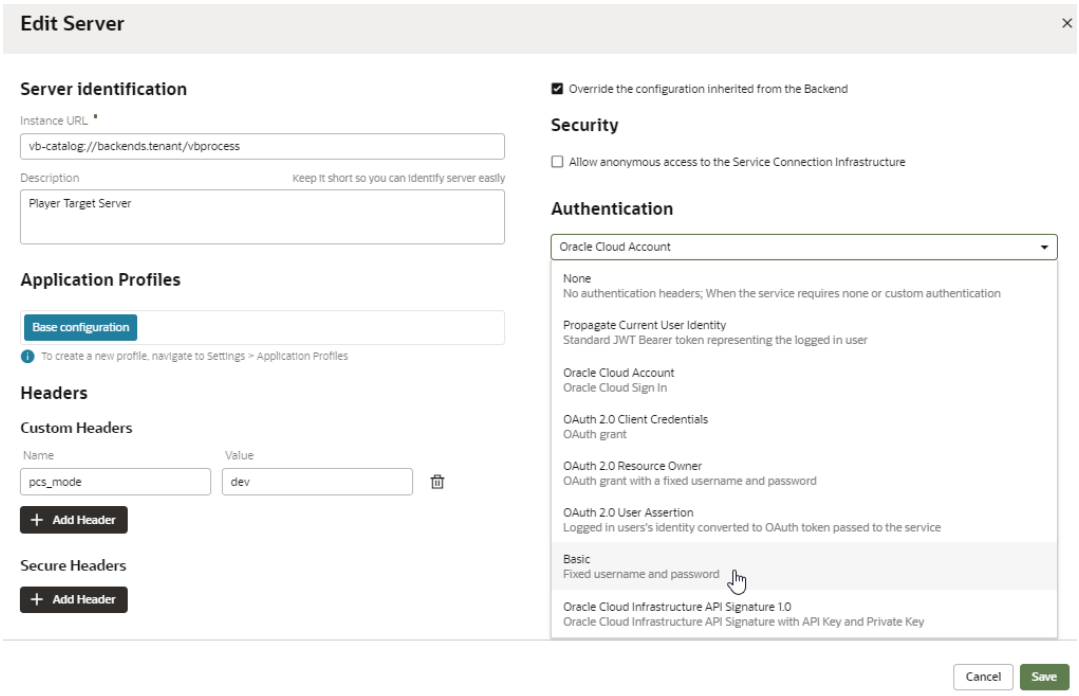
Process Server Authentication Options

For each Process Server configured, like the Player Target Server and the Default Target Server, you can specify an authentication mechanism.

For example, you may want to specify basic authentication for development and testing, and the user's Oracle Cloud Account for the published app.

To specify authentication:

1. Click the **Services** tab under **Settings** (if you aren't already in Settings, select **Settings** from the top right menu).
2. Click the **Edit** icon next to the target server you want to edit.
3. In the **Edit Server** dialog box, select the authentication mechanism you want to use for that server from the **Authentication** drop-down list. The list has several choices, but for processes only Oracle Cloud Account and Basic are supported.



4. Click **Save**.

At any stage of 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.

Selecting **Test Connection** in the options menu for the Processes navigator will by default use Oracle Cloud Account authentication details to test the connection. This means that to test the connection your user credentials must be valid to access the Process Server.

22

Work With Translations

You can use the Properties pane 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.

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

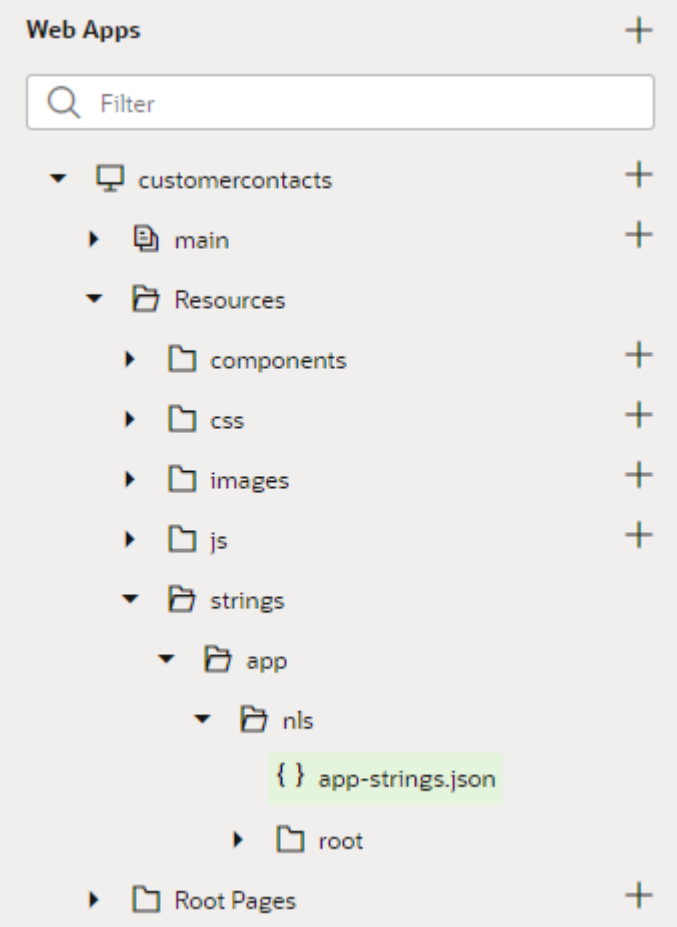
You will want to understand the structure of translation bundles if you want to create additional translation bundles in your application.

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 Properties pane, 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 `<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:

```
[[ $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:

```
"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 Properties pane to add keys and values for the strings to a bundle for translation.

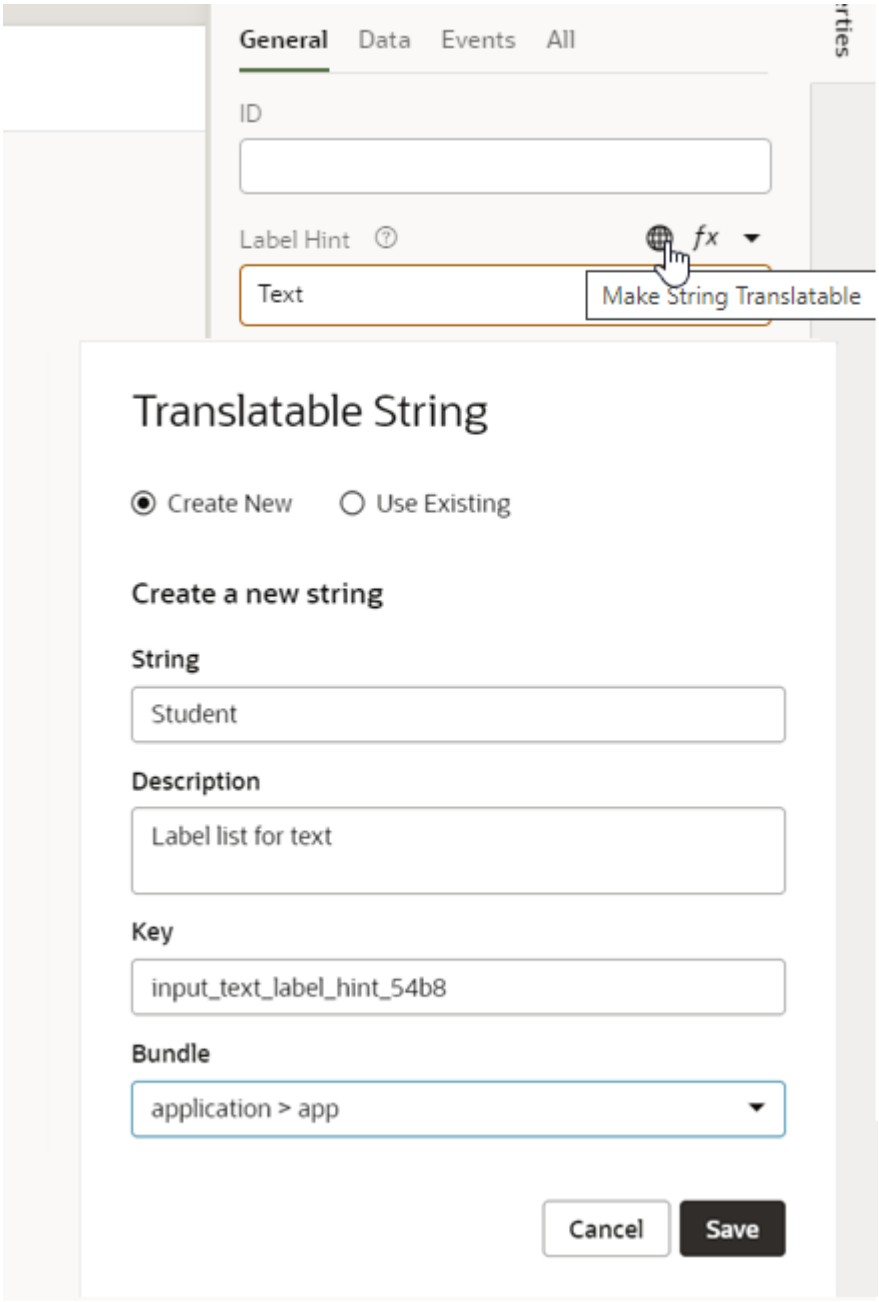
You can use the Properties pane to generate and add keys for UI components to a translation bundle. When you use the Properties pane to define the translatable string, a key is generated automatically, but you can specify your own key in the dialog box. You can also edit the keys, values and metadata in translation files directly using a text editor.

To generate a key for a string using the Properties pane:

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. Select **Create New** to add a new string to a translation bundle.
4. 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.

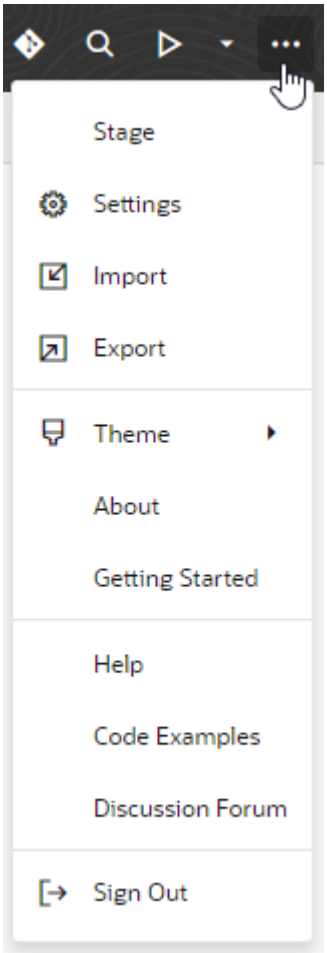
When you download the translation bundles of your visual application, the archive that you download contains the translation files for all of the applications in your visual application. The archive contains a folder for each translation bundle in your application, and each folder contains a file in the `.json` or `.xliff` 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",
```

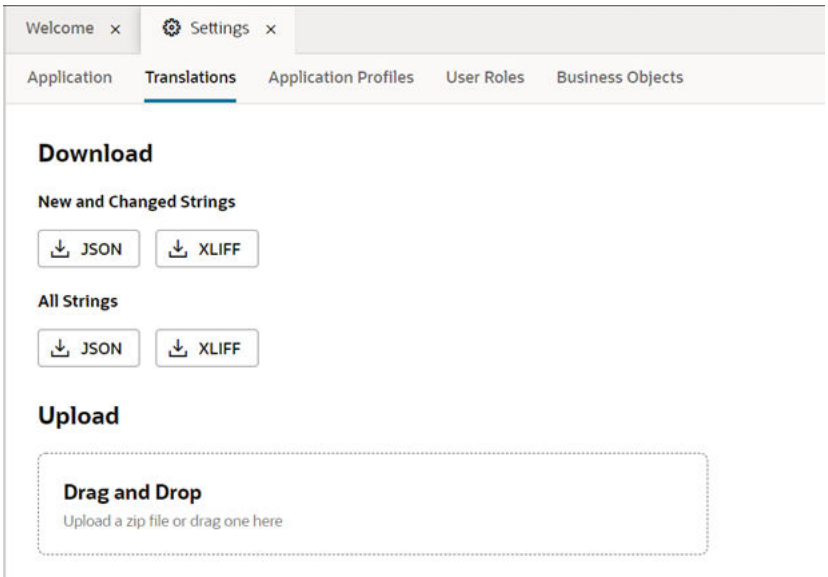
When downloading the bundle, you can choose to download an archive containing all strings that are stored in the translation bundle or an archive containing only those strings in the translation bundle that were added or updated since the last time you downloaded the translation bundle. The first time that you download the translation bundle you can only choose to download an archive containing all strings.

To download a translation resource bundle:

1. Click **Menu** in the upper right corner and select **Settings**.



2. Click the **Translations** tab in the Settings editor.



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 Properties pane 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 Properties pane.

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.

   ```
   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:

   ```
   PageModule.prototype.getMessageFromBundle = function(translations) {
     ...
   ```

```
  return translations.app.h1__text_041a;
};
```

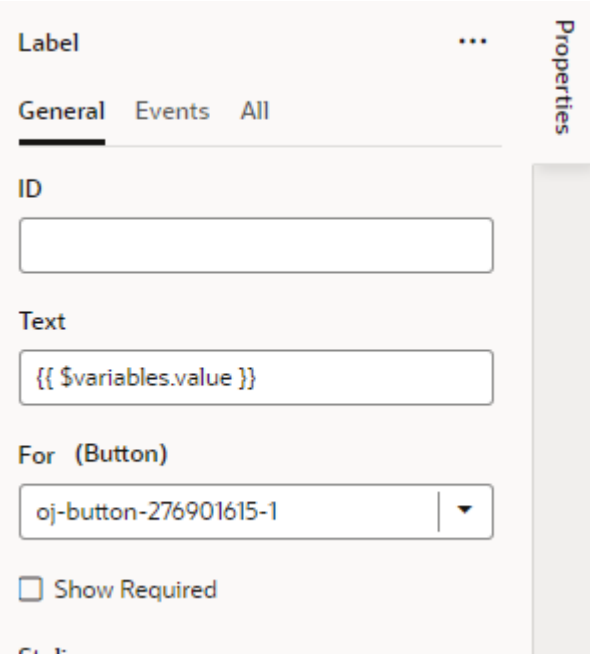
You can also include parameters to generate a formatted message:
`translations.format('app', 'h1__text_041a', param1, param2)`

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`.

```
"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`).



> **Tip:**
>
> Use the Text field's Select Variable menu to select the correct page variable

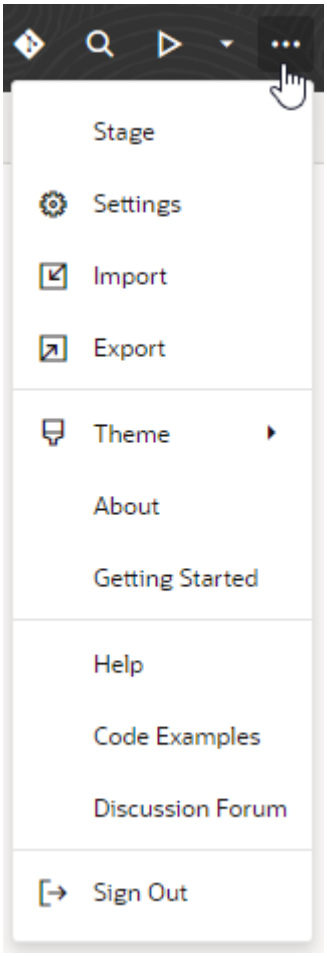
Upload Translated Files

You can upload the resource files containing translated strings in the Translations tab in the Settings editor.

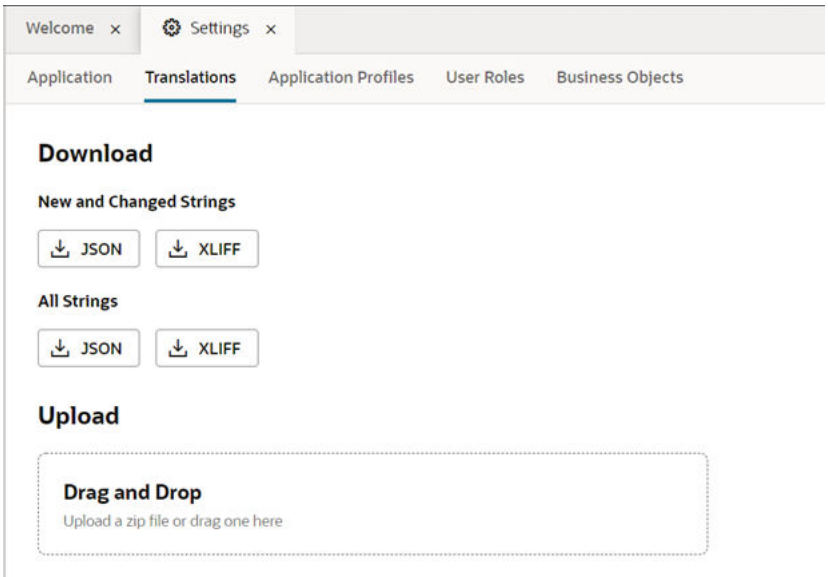
To upload translated files, you need to create a ZIP archive containing the files with the translated strings. Before you create the archive, however, you need to modify the file names to append the locale abbreviation to identify the correct locale. For example, to upload a JSON file that contains translations for the French locale, you will modify the file name to `<bundle-name>-strings-fr.json`.

To upload an archive with translation files:

1. Click **Menu** in the upper right corner and select **Settings**.



2. Click the **Translations** tab in the 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**.

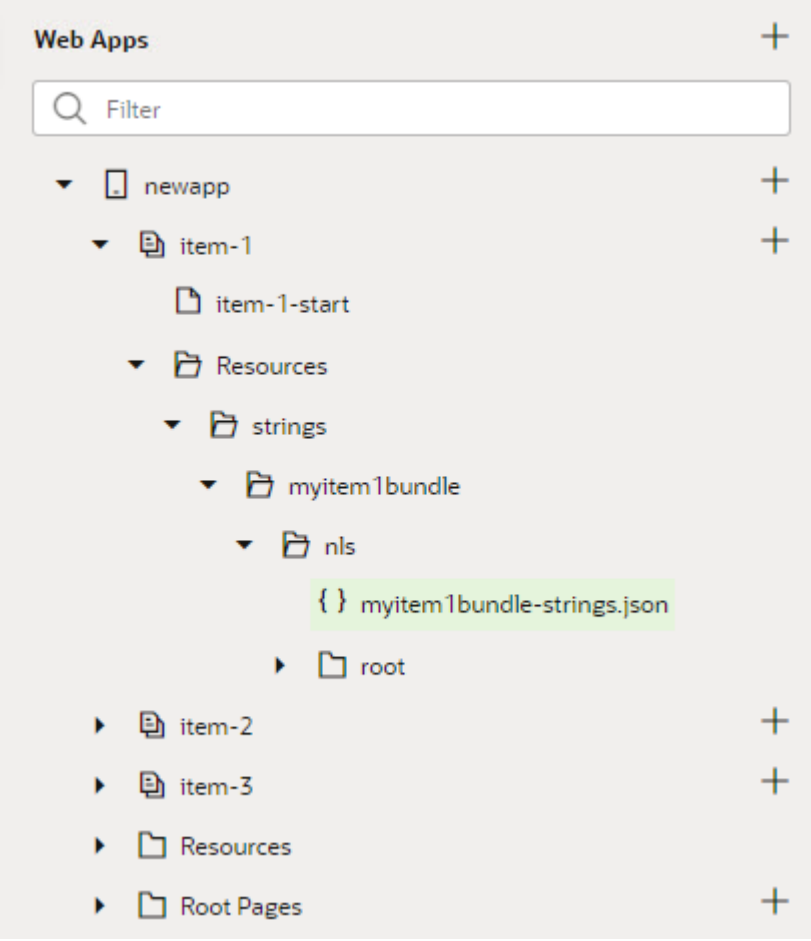
If you added a file with translations for a new locale, a folder for the new locale is created in the bundle. The new locale is also automatically added to the file defining the locales in the bundle.

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 `app` 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.

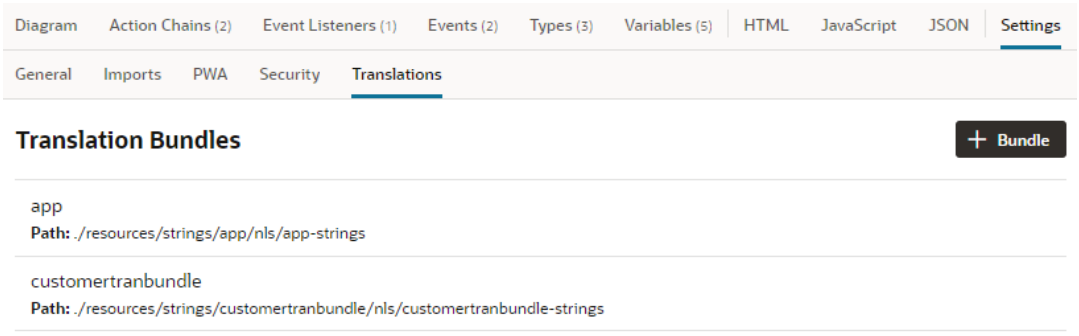


When you externalize a text for translation, the string and key are added to the nearest available bundle by default. For example, if you create a translation bundle for a page, the strings and keys are added to the page's bundle when you externalize strings in the page. Strings in other pages are not 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.

Part V
Manage Applications

As you work through the application development lifecycle, learn how you can view the status of an application and create versions, debug application code, then stage and publish it. Also, take steps to keep your application up-to-date with the latest release of Visual Builder.

Topics:

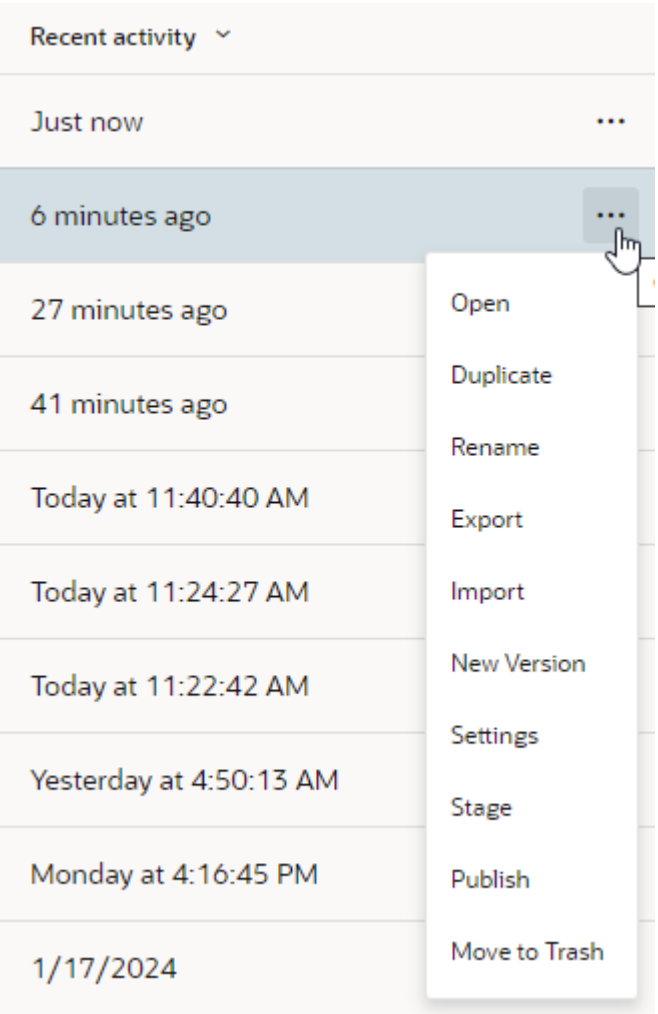
- Manage Your Visual Application
- Integrate Your Visual Application With a Git Repository
- Test and Debug Applications
- Stage and Publish Visual Applications
- Manage Runtime Dependencies for Visual Applications
- Optimize Your Builds and Audit Your Code Using Grunt

23

Manage Your Visual Application

You can manage your visual applications using the tools available on the Visual Applications Home page. Each row in the table on the Home page represents a version of an application. 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.

To manage your application's lifecycle, look for the Application Options menu:



The following table describes the options you might see in the menu:

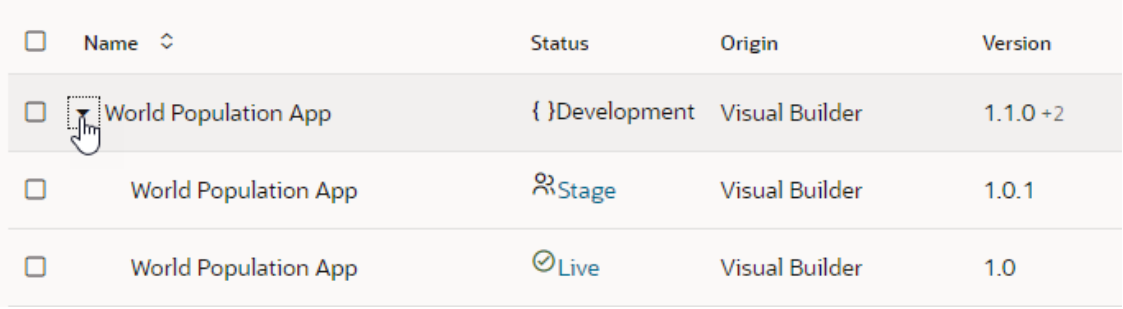
Menu Item	Description
Open	Opens the development version of the application
Duplicate	Creates a clone of this version of the application, including the content of the database.

Menu Item	Description
Rename	Opens a dialog box where you can change the name of the application.
Export	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 business objects.
Import	Opens a dialog that you can use to create an application by uploading an application archive (ZIP or OVB) from your local system.
New Version	Creates a new version of the same application. By default the new version is a development version. Version numbers are automatically increased incrementally.
Settings	Opens an editor for configuring the application's settings and viewing the application API URLs. Each application version has a dedicated Settings editor.
Stage	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.
Publish	Opens a dialog box where you can specify the database option and publish the staged version of your application.
Lock / Unlock	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.
Rollback	Rolls back the live version to the previous live version. This is only available for the current live version.
Move to trash	Deletes the application from the identity domain. You have 30 days to recover the application after deleting it.

View an Application's Status

Your application's status is always shown on the Home Page. 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 and status. 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 drop-down list to organize the order that the tiles are listed on the page.



The following table describes the application status:

Status	Description
Development	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.
Stage	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.
Live	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. To make changes to a live version, use the Application Options menu to create a new version of the application.
Live Locked	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.
Obsolete	This status indicates that this version was published but has been superseded by a newer version. Obsolete applications are read-only.
Soft Deleted	This status indicates that application was deleted from the Home page, but won't be completely deleted until you either delete it permanently or 30 days passes with no action. Soft-deleted applications are read-only. If needed, the app can be restored within 30 days of deleting it.

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, you will need to manually make those changes in other versions of your application that you want to include that change.

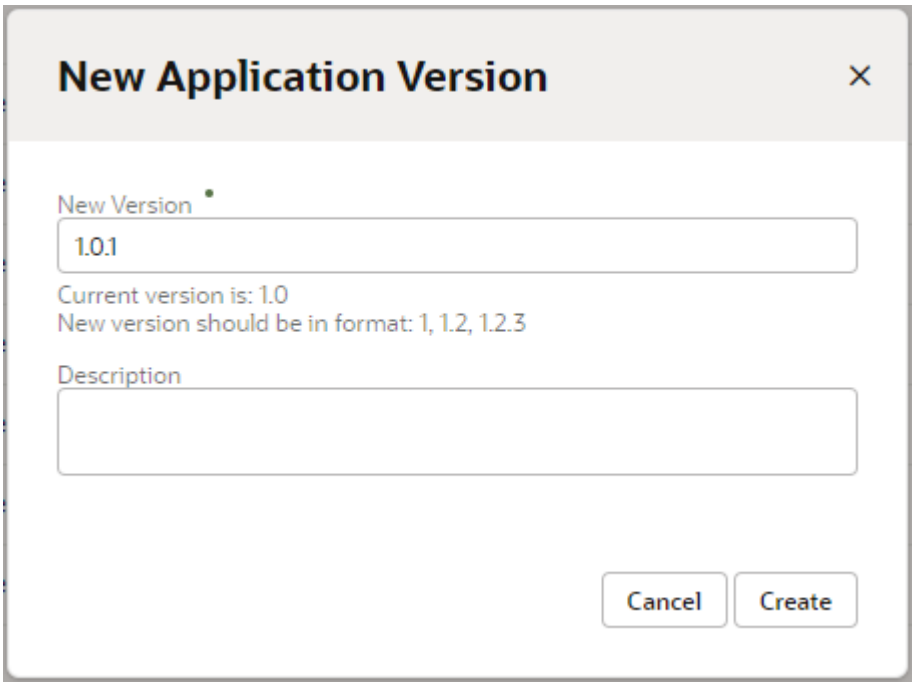
To create a new version of an application:

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



4. Click **Create**.

   The new version (with a Development status) is created on the Home page.

> **Note:**
>
> If you're unable to create a new version, it may be that your application has reached its total limit of 100 versions. Check the application's Versions column on the Home page, where you can see the number of versions next to the current version number, for example, `2.8.9 +100`. If the number of versions is 100, you need to permanently delete some versions of your app, starting with the oldest obsolete version (see Delete a Visual Application). Once you've deleted enough versions to fall within the version limit, try creating a new version again.

Delete a Visual Application

If you no longer need a visual application or a particular version of it, you can delete it from the Home page at any time during the development lifecycle. You can also permanently delete an application to completely remove it from the system.

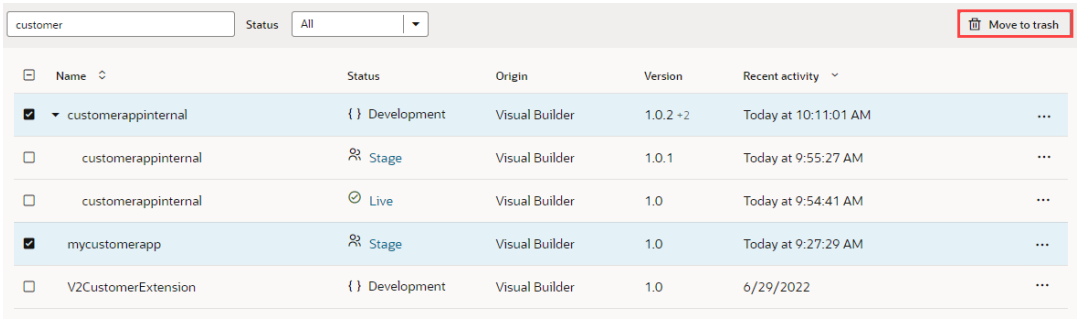
It's especially important to delete obsolete versions of your application, created when a published version is superseded by a newer version. If your app has 100 versions (the maximum number of versions an app can have), you won't be able to create new versions until you permanently delete some versions to stay within the version limit.

> **Caution:**
>
> Permanent deletion of an application cannot be undone. It completely removes application metadata and any data stored in its database. Before you delete, consider the impact of deleting your app, especially if the version is live. While you still have a chance to restore the app within 30 days of deletion, once a version has been removed from the system, it cannot be recovered.

If your teammates have access to the apps you want to delete, you might want to remove them before you delete the app. Use the **Team** tab in the app's Settings editor to remove access.

To delete visual applications:

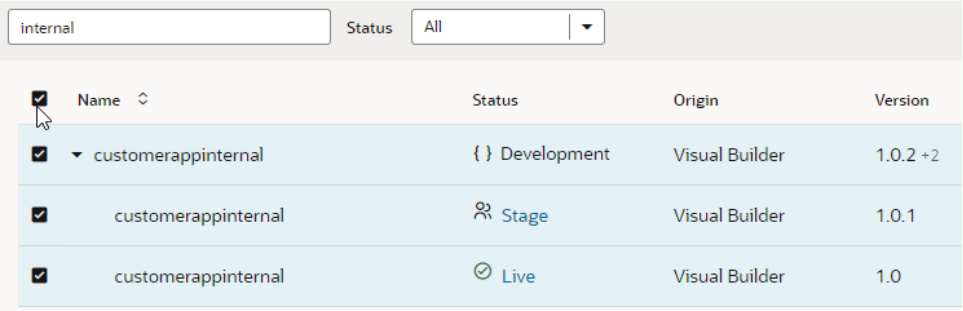
1. On the Home page, locate the applications you want to delete. You may find it helpful to filter your applications by status. For example, to locate obsolete applications, select **Obsolete** from the **Status** list, then further filter the results using the **Filter by Name** field.
2. Select one or more of the applications you want to delete, then click **Move to trash**. You'll also find the **Move to trash** option in an application's **Options** menu ⋯.



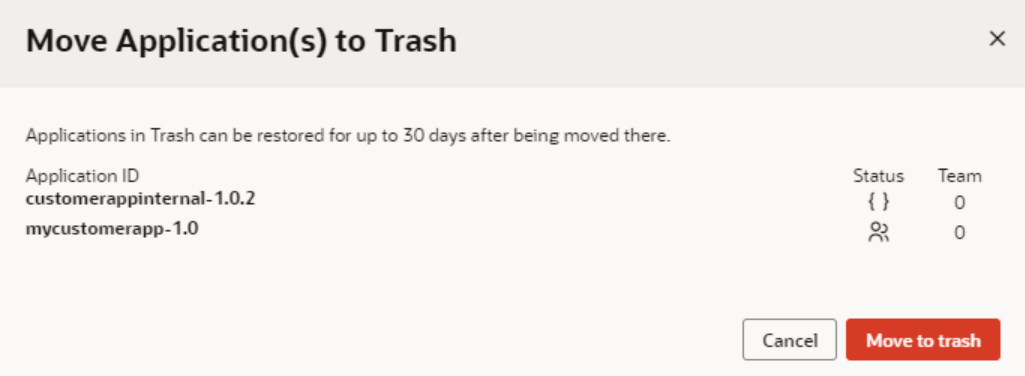
Selecting an application at its root level only selects its current version. If this isn't what you want, expand the application and select one or more of the versions you want to delete.

Tip:

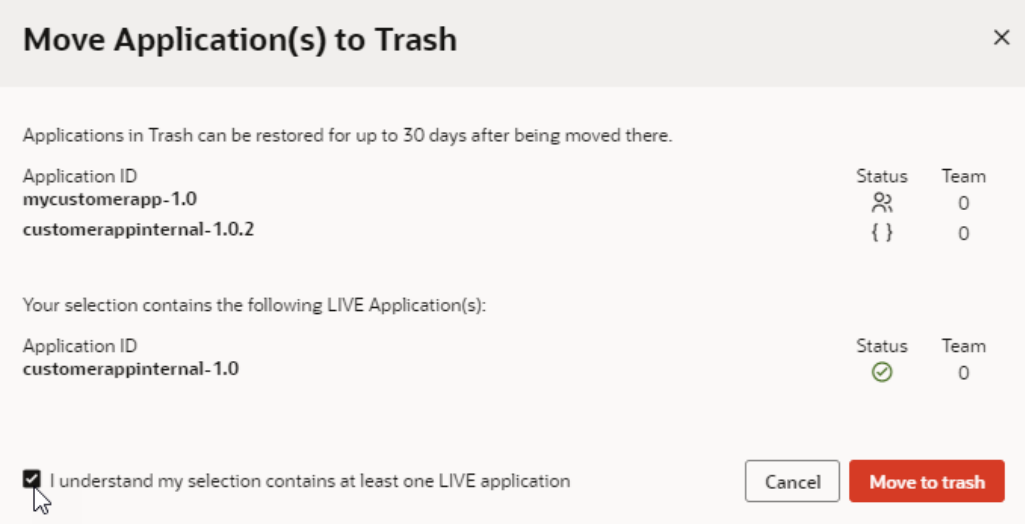
To delete multiple versions of an app all at once, enter the app's name in the **Filter by Name** field, click **Name** in the header column to select all versions of the app, then select **Move to trash**.



3. Confirm your selection:
   - For development or staged versions, click **Move to trash**:



   - For live or live locked versions, select **I understand my selection contains at least one LIVE application**, then click **Move to trash**:

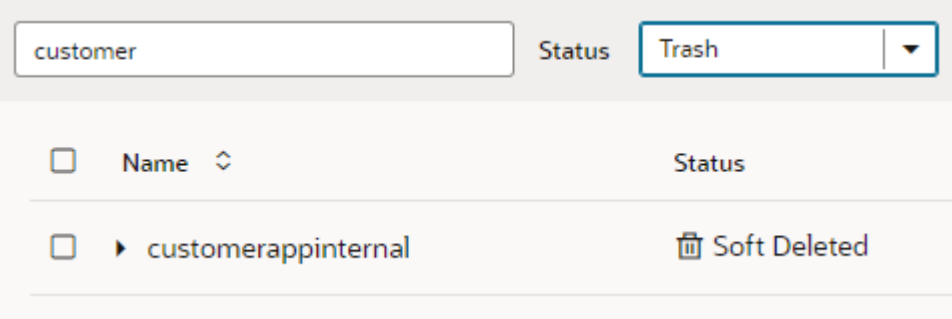


A confirmation appears and your applications no longer appear on the Home page. Instead, they are moved to a Soft Deleted status and can be restored within 30 days after initial deletion.

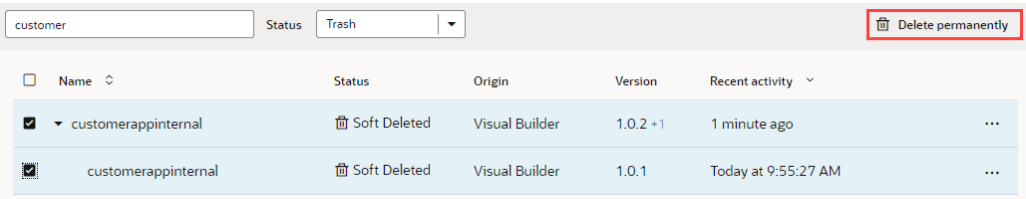
4. If you're sure you no longer need a deleted application, you have the option to permanently delete it. You might also need to do this when your application has too many versions, preventing you from creating new versions.

> **Caution:**
>
> Permanent deletion of an application cannot be undone. It completely removes application metadata and any data stored in its database.

   a. On the Home page, filter applications by **Trash** status. You can apply additional filter criteria by entering a partial or full name for the application.



   b. Locate the application you want to remove from the system and click **Delete permanently**. You'll find the same option in an application's **Options** menu ⋯.

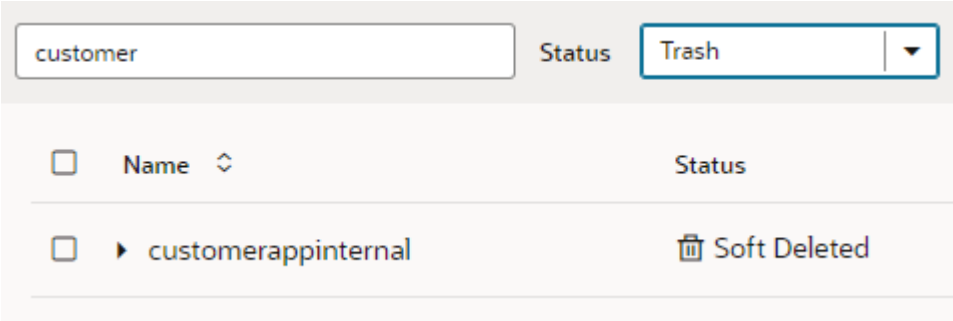


Restore a Deleted Application

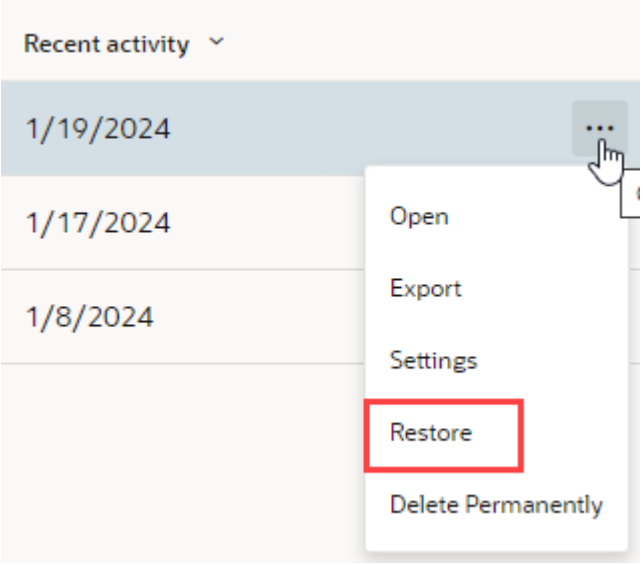
Visual applications that you've deleted are moved to a Soft Deleted status for 30 days after initial deletion and can be recovered if they've not been removed from the system. Soft deleted apps are permanently deleted if you do so explicitly or 30 days pass without any action. Permanently deleted applications **cannot** be restored.

To recover a soft deleted application within 30 days of deleting it:

1. On the Home page, search for the deleted application using the **Trash** status filter. You can apply additional filter criteria by entering a partial or full name for the application.



2. Locate the application (or version) you want to restore, click its **Options** menu ⋯, and select **Restore**.



Opening an application from Trash allows you to view its contents in read-only mode.

3. Once your application is restored, change the Status filter to **All** and locate your restored application.

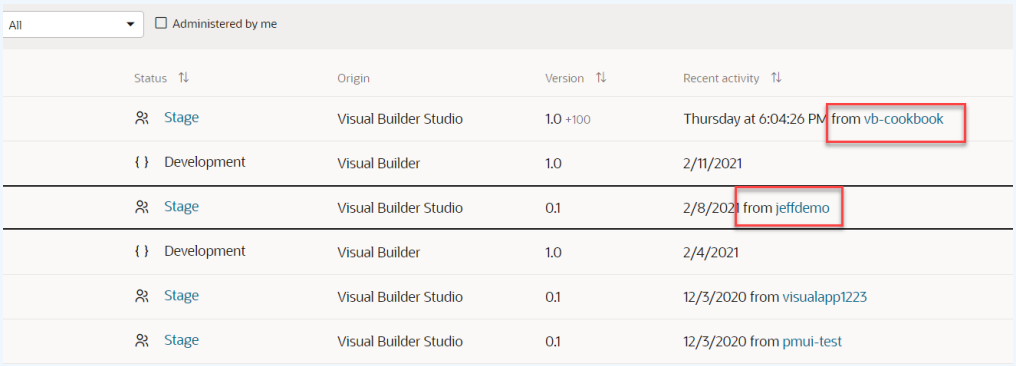
Manage Applications Created in Visual Builder Studio

The way you manage visual applications created in Visual Builder Studio is fundamentally different from the way you manage Visual Builder-native applications. For starters, VB Studio applications are managed in the context of a *project*, you do your work within a *workspace* tied to a Git repository, and your app is deployed via a sophisticated CI/CD pipeline; Visual Builder has none of these underlying concepts. As a result of these and other disparities, the actions you use to manage applications created in VB Studio *must be performed within VB Studio*, and not from the Visual Builder Home page where you manage Visual Builder-native applications.

This table lists all the actions available to native Visual Builder applications and explains a bit about how to perform them for applications originating in VB Studio.

Note:

The first step for each of these procedures is to go to the VB Studio project, using either the "from *project_name*" link in the VB Home page (shown below) or the VB Studio URL.



For each of these actions, go to the Visual Builder Studio project, then:

Visual Builder Action	In VB Studio:
Open	On the Project Home page, click your workspace name to open the app in the Designer. (The workspace may have a different name from the visual app as it appears on the VB Home page.)
Duplicate	The Duplicate action isn't exactly the same in VB Studio as it is in Visual Builder, and you shouldn't have as much of a need for the action in the context of VB Studio. One way to duplicate your app in VB Studio is to: 1. On the Project Home page, click your workspace name to open the app in the Designer. 2. In the upper right menu, select **Export**. This creates a .zip file of the exported app. All the changes you've made in your workspace, even those you haven't committed to your local Git repository yet, will be included in the .zip. 3. In the upper left corner, click the left arrow to exit the Designer. 4. On VB Studio's Project Home page, in the Workspaces pane, click **Manage Workspaces**. 5. Click **Import** to create a new workspace with the imported app. 6. Drag and drop the file you exported in step 2 and click **Import**. 7. Once you are in the Designer, click the menu in the upper right and click **Settings**. 8. In the **Root URL** field, give the app a new name (assuming you want it to be different from the original). 9. **Share** or **Publish** the app. The name you specified in step 8 will appear on the VB Project Home page.

Visual Builder Action	In VB Studio:
Rename	In VB Studio, the name of a visual application is determined by the Settings tab's **Root URL** field at the time the app is deployed or shared. By default, the Root URL is set to the repository name you specified when you created a workspace for the app. To rename a visual app: 1. On the Project Home page, in the Workspaces pane, click **Manage Workspaces**. 2. Once you are in the Designer, click **Source View** in the Navigator. 3. Click **visual-application.json**. 4. Enter the new name for the visual app in the **rootURL** field: 5. **Share** or **Publish** the app. The name you specified in step 4 will appear on the VB Project Home page.
Export	1. On the Project Home page, in the Workspaces pane, click **Manage Workspaces**. 2. On the appropriate row, click the **Actions** menu on the far right. 3. Click **Export**.
Import	1. On the Project Home page, in the Workspaces pane, click **Manage Workspaces**. 2. Click **Import** to create a new workspace. 3. Drag and drop the exported .zip file.
Settings	1. On the Project Home page, click your workspace name to open the app in the Designer. 2. In the Designer, click the menu in the upper right. 3. Click **Settings**. **Note:** While Visual Builder Settings provide a **Team** tab, in VB Studio, team membership is handled at the project level. See Add and Manage Project Users.

Visual Builder Action	In VB Studio:
Stage	VB Studio does not have the same concept of stage that Visual Builder has. Whereas in Visual Builder you staged an app whenever you wanted to make a particular version available to others, in VB Studio you actually have two options for this, depending on whom you want to share the app with: • To make a version of your app available to another person, or perhaps to a small group (that is, somewhat informally): 1. On the Project Home page, click your workspace name to open the app in the Designer. 2. In the upper right menu, click **Share**. This takes a snapshot of whatever is currently in your workspace and assigns it to a URL unique to that snapshot. • To officially package the app's resources to, say, share it with another group, like QA (as opposed to just sharing what you have in your own private workspace): 1. On the Project Home page, click **Builds** in the left navigator. 2. Click **Configure** next to the app's Deploy job. 3. (Optional) On the **Steps** tab, add a version number to the **Application Version** field. If you don't, whatever is specified in the app's Settings > General tab will be used. 4. Select **Include the application version in the URL**. 5. (Optional) If your app contains business objects and you want to include their data, click **Add Step** > **Visual Application** > **Import Data** and supply the required information. 6. If you haven't already, supply your user name and password for the target environment. 7. Click **Save**. 8. On the Builds page, click **Pipelines**, then click the Build icon for this visual application. This will trigger the package job, followed by the deploy job. For both these options, the status on the VB Home page will be `Stage`. However, the Share option creates a dummy version number (which you can ignore), while the build job lists the version number you specified in the Application Version field.

Visual Builder Action	In VB Studio:
Publish	In Visual Builder, you used the Publish action to move a particular visual application to Live status. In VB Studio you can do this from the Designer, as long as you configure the deploy job properly first. 1. On the Project Home page, click **Builds** in the left navigator. 2. Click **Configure** next to the app's Deploy job. 3. On the **Steps** tab, uncheck **Include the application version in the URL**. 4. If you haven't already, supply your user name and password for the deployment credentials. 5. Click **Save**. 6. Click **Workspaces** in the left navigator and open the workspace for the app. 7. In the header, click **Publish**. This commits your changes to the project's Git repository and kicks off the package and deploy jobs. When the job finishes, you will have a permanent URL for this visual app that will remain viable even if you have to republish it later, so you don't have to keep giving people different URLs. On the VB Home page, you will see your app with a status of `Live`. If you click the app name you'll see the URL, which contains the word "live" (unless you're using a custom domain). Only one version of an app can be live at a time. While the version number is not included in the URL, if you republish this app, you must be sure to increase the version number (in Settings) from its predecessor or the deploy job will fail.
New Version	To change the version number of your app: 1. On the Project Home page, click your workspace name to open the app in the Designer. 2. In the Designer, click the menu in the upper right. 3. Click **Settings**. 4. Enter the version you want in the **Version** field. 5. Click **Publish**. (This will kick off your Deploy job, so make sure **Include the application version in the URL** is checked or unchecked, as needed. See Stage and Publish above.)
Move to trash	1. From the Project Home page, click **Environments**. 2. Click the **Deployments** tab. 3. Click the **Actions** menu on the appropriate row. 4. Click **Undeploy**.

24

Integrate Your Visual Application With a Git Repository

To collaborate with other team members and back up your sources, you can integrate your visual application with a Git repository hosted in a Oracle Visual Builder Studio 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. In case of merge conflicts when pulling sources, you can view a log of conflicts in the Audits pane, then open the files in the source editor to resolve conflicts.

Add Credentials for Your Oracle Visual Builder Studio Account

To integrate your visual application with a Git repository, you must provide the URL and credentials for an Oracle Visual Builder Studio account. Those credentials are used when linking your visual application to a Git repository of an Oracle Visual Builder Studio project.

You will need to contact your service administrator if you do not know the credentials for your Oracle Visual Builder Studio account. Credentials are not stored when you export your application.

The credentials for connecting to Git in Oracle Visual Builder Studio must be for a user in Oracle Identity Cloud Service (IDCS) with the DEVELOPER_USER role. Note that:

- The credentials you provide must be for a local IDCS user. Local users are those created directly in IDCS, with the password also specified in IDCS for basic authentication. You cannot use federated or single sign-on (SSO) credentials to connect to Git.
- The IDCS user must have the correct IDCS role. Check with your service administrator to confirm that the user is assigned the correct role. The service administrator can add a user in IDCS and assign the user the DEVELOPER_USER role. See Add Users to a Cloud Account with IDCS in *Administering Oracle Visual Builder in Oracle Integration Generation 2*.

To add your Oracle Visual Builder Studio account credentials:

1. Open your visual application.
2. Click the Git icon ( ) in the toolbar and select **Configure Visual Builder Studio Credentials**.
3. Click **Add Credentials** in the Configure Visual Builder Studio Credentials dialog box.
4. Enter the URL, user name, and password for your Oracle Visual Builder Studio account. Click **Save Credentials**.

   After your credentials are checked and saved, the new credentials are added to the list in the Configure Credentials dialog box.
5. Click **Close**.

After you enter and save your credentials for the Oracle Visual Builder Studio 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 credentials for your Oracle Visual Builder Studio account, you can create a link between your visual application and a Git repository of your Oracle Visual Builder Studio project.

Linking your application to the branch of a Git repository lets you 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 Visual Builder Studio 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 Visual Builder Studio Git Repository**.

   The dialog box displays the location of a Git repository if one is already linked to your visual application.
3. Click **Add Link** to open the Link Git Repository dialog box.
4. In the URL with Credentials field, select the URL of the account that you want to use.

   The drop-down list displays the Oracle Visual Builder Studio accounts that you've provided credentials for.
5. Select the project, the Git repository, and the repository branch.

   The drop-down list displays the projects, repositories, and branches that are available to you in the instance. The branches and repositories shown are determined by the project you select.

   Once you select a branch, the Branch Head field shows the ID of the latest revision on the remote branch in Visual Builder Studio. HEAD is a special reference to the latest revision in your local copy of the branch in Visual Builder. You can choose to set HEAD either to the remote branch head or select **Use Custom Head** to set it to any revision ID (which is a 40-character SHA-1 hash). See Get the Revision ID of a Commit in *Building Responsive Applications with Visual Builder Studio* for information on how you can find a particular revision ID.
6. Click **Save Configuration**, then **Close**.

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** option 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** 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.

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** 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**.

   The comment message that you provide in the dialog box is displayed when you examine the Git activity log for the branch.

4. Click **Close** when the push is finished.

   The dialog box displays a status message when the push is complete.

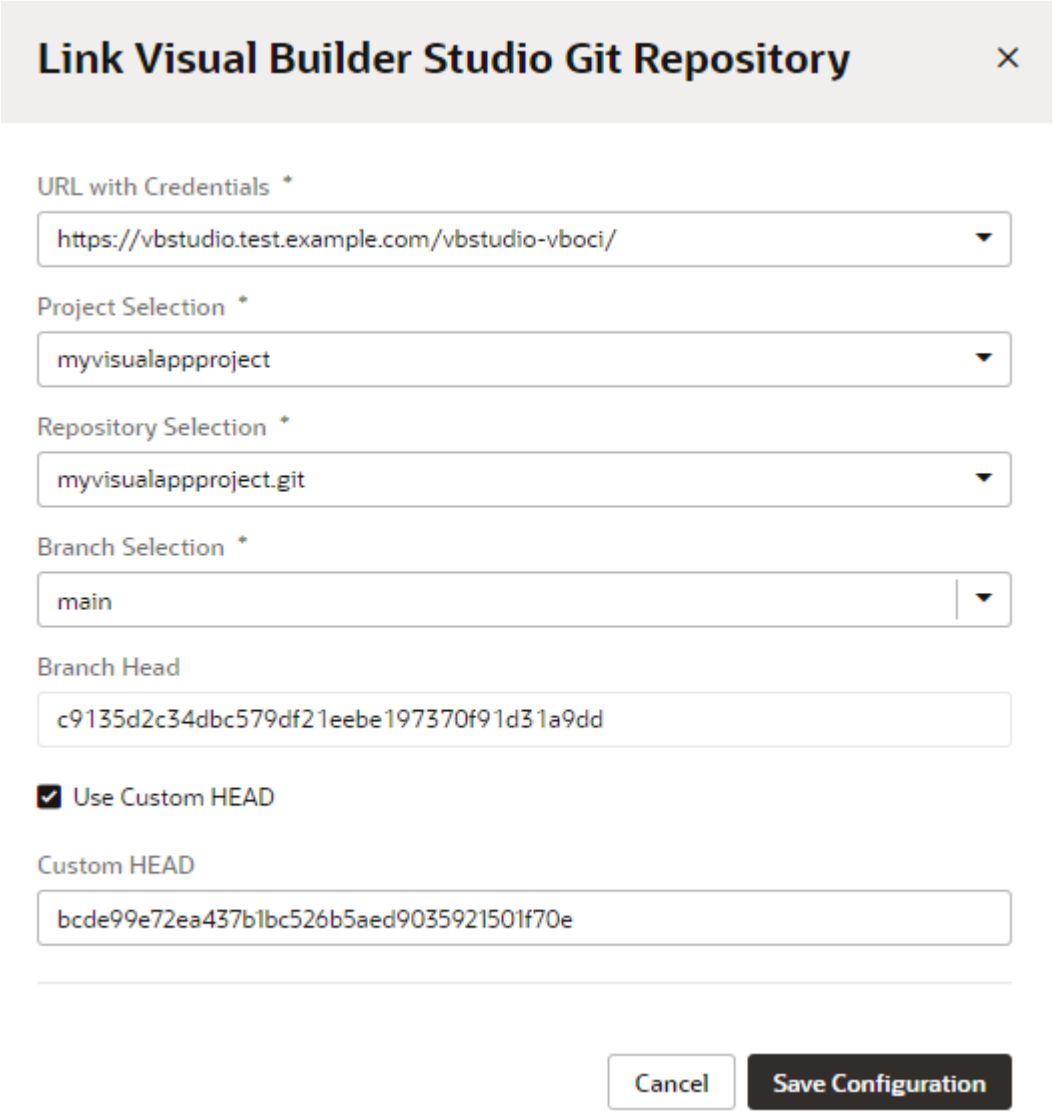
Change the Local Branch HEAD in a Linked Git Repository

When you link a visual application to a Git repo in a Visual Builder Studio project, the branch HEAD in your local copy, by default, references the most recent commit that was pulled or pushed. If the commit referenced by the current local HEAD is removed from the remote branch (for example, because somebody rebased the remote branch and garbage collection occurred), push and pull requests will fail. In this scenario, you can change the current branch HEAD to use the latest branch head or any other commit to resolve the issue.

To change the branch HEAD in your local copy of a linked Git repository:

1. Open your visual application.
2. Click the Git icon and select **Link Visual Builder Studio Git Repository**.
3. Click **Edit Link** to change details of the Git repository linked to your visual app.

   In the dialog box that opens, the Branch Head field shows the current remote branch head of the selected branch. The Custom HEAD field shows the current local HEAD. If Custom HEAD is not the same as the branch head, then the Use Custom HEAD option will be selected initially, as shown here:



If you select a different branch, the Use Custom HEAD option won't be selected; it will stay that way if you reselect the original branch.

4. To change the current local branch HEAD, select **Use Custom HEAD**, then enter the revision ID you want to use in the **Custom HEAD** field.

   The revision ID (also known as the commit ID in Git) is a 40-character SHA-1 hash. See Get the Revision ID of a Commit in *Building Responsive Applications with Visual Builder Studio* for information on how you can find a particular revision ID.

5. Click **Save Configuration**, then **Close**.

You can now use the push and pull commands in your visual app's Git menu to push and pull sources that match the new branch head.

25

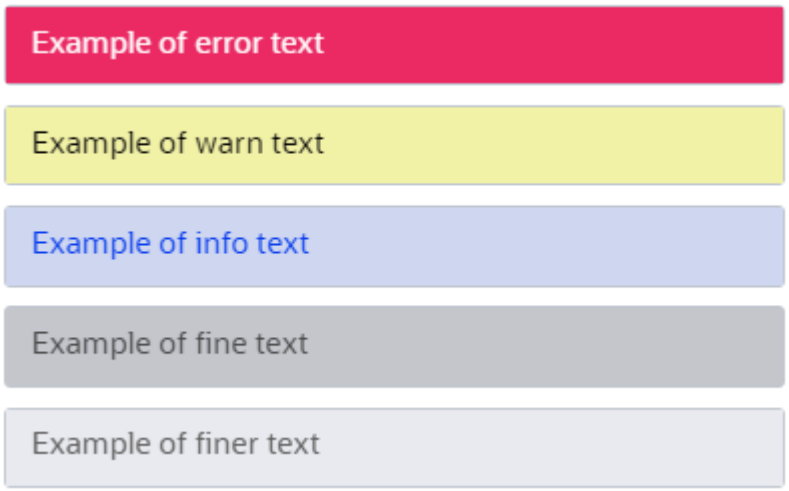
Test and Debug Applications

As your work on your application, it's important to test and debug it to ensure a smooth end-to-end experience for your users.

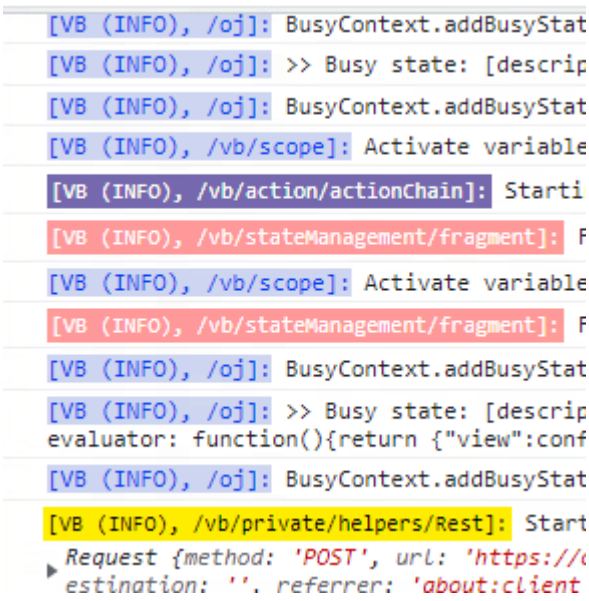
You have several options to ensure your application's code is error-free:

- Browser tools: Because Visual Builder applications are essentially client-side HTML applications written in JavaScript, you can use your browser's development tools for debugging. When you view your Visual Builder application in Chrome (for example), you can use the **Developer tools** option to diagnose problems quickly:
  - Network tab: Use this tab to view network traffic between the client's browser and your REST data sources.
  - Console tab: Use this tab to track error messages and notifications from your application. Visual Builder apps output `info`-level log messages to the console by default. Other log levels are `error`, `warn`, `fine`, and `finer`. See Change an Application's Log Level to change the log level.

    Console log messages are color coded to make the output more readable. By default, *error* messages are red-magenta, *warnings* are yellow, *info* messages are blue-purple, while *fine* and *finer* are varying shades of gray, as shown here:



    Further, *info* messages from particular modules use different colors as a way to differentiate them from other console messages. For example, action chain messages are in dark purple, fragments in coral, and REST in yellow, as shown here:



Console colors cannot be customized, but you can turn them off by setting `window.vbInitConfig.LOG.mode = 'simple'` in the app's `index.html` file:

```
<script type="application/javascript">
  window.vbInitConfig = window.vbInitConfig || {};
  window.vbInitConfig.LOG = {
    mode: 'simple'
  };
</script>
```

- Audits pane: Use an audit framework to make sure your application's code is error free. See Audit Application Code.
- Debug Preview mode: Use this mode to troubleshoot issues with the Visual Builder runtime and Oracle JET libraries—runtime dependencies that make sure your application works as intended. See Preview an App in Debug Mode.

Audit Application Code

Use the Audits pane to check and verify your application's code as you develop it.

When you open application artifacts, Visual Builder automatically scans its code and displays errors and warnings (if any) in the Audits pane. You can view details of each issue and take action to resolve it. In addition to JET component errors, the Audits pane displays syntax errors, translation issues, and warnings for missing dependencies.

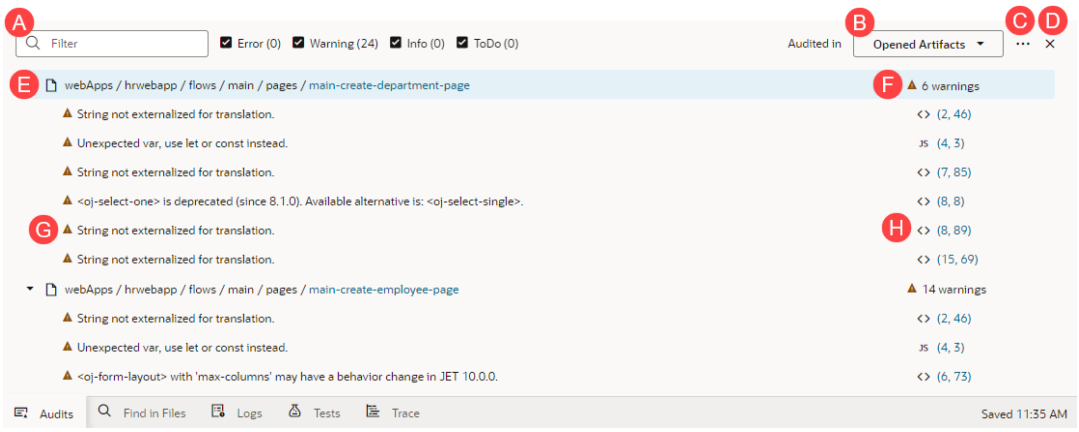
If an artifact contains errors, it is also badged in the Navigator to indicate that action must be taken to resolve the issue. For example, page errors are indicated by a red dot on the Web Applications icon in the Navigator toolbar as well as against the particular artifact in the Web Apps pane. The badge persists until the error is fixed or until the session ends.

Here's how you can audit your application's code:

1. Click **Audits** at the bottom of your browser.

   If the Audits pane displays a message that no artifacts are open, you'll need to open the artifacts you want audited, for example, one or more pages or a .JS file. Here's a quick

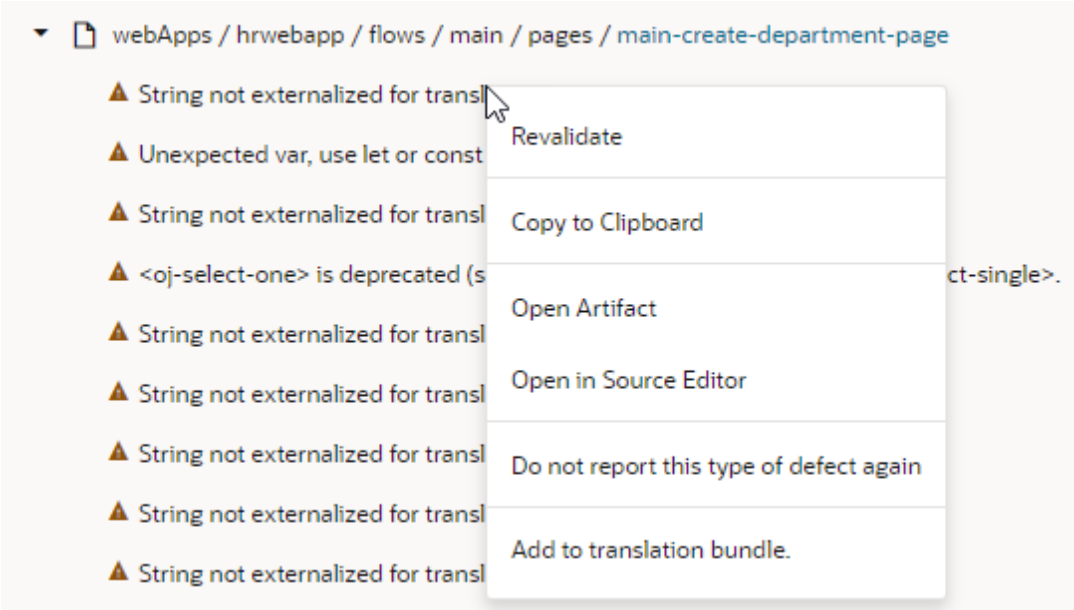
walkthrough of the results displayed in the Audits pane when artifacts are open (with the active artifact always highlighted):



Label	Description
A	Options to filter audit results either by text or by severity: • Enter text to filter the results based on your search string. For example, you might enter `main` in the filter text box to search for all issues in the `main` flow. • Use **Error**, **Warning**, **Info**, and **ToDo** to filter the results by severity. For example, when your application contains errors, you can clear the **Warning**, **Info**, and **ToDo** checkboxes to focus on errors.
B	Scope of the audit. By default, only artifacts that are open are audited, for example, the `main-create-department-page` and `main-create-employee-page`. To audit the application as a whole, change **Audited in** from **Opened Artifacts** to **All Artifacts**. Any scope change you make is retained for the application, and will persist even if the Audits pane is closed and reopened.
C	Menu with options to edit audit settings or revalidate the entire application.
D	Option to close the Audits pane.
E	Path to the artifact that contains issues, for example, the `main-create-department-page` under `webApps/hrwebapp/flows/main/pages/`.
F	Number and type of issues in the artifact, for example, `7 warnings`.
G	Message about the issue in the artifact, for example, `String not externalized for translation`.
H	Line and column number of the issue, for example, `(8, 89)` indicates that the `String not externalized for translation` issue exists in line 8, column 89 of the `main-create-department-page` artifact in Code view.

2. Decide how you want to resolve the issue. You can choose to resolve an issue directly from its right-click menu or by opening it in the source editor.

   For example, to resolve a translation issue, right-click the issue and select **Add to translation bundle**:



To work with an issue in its source editor, simply click the issue, or select **Open in Source Editor** in the issue's right-click menu. You can then use code actions suggested in the editor to resolve the issue. For example, for the `String not externalized for translation` issue, you'll see the same quick fixes suggested in the right-click menu available in the source editor as well:

Whether you use the right-click fixes or the source editor is entirely up to you, but will mostly depend on the issue you're working with.

- Resolve the issue, if possible, by selecting the appropriate option in the right-click menu. For example, for the `String not externalized for translation` issue, select **Add to translation bundle**.
- If a resolution isn't available for your issue, select **Open in Source Editor** to view the issue in the source editor. Hover over the highlighted issue to view problem details and see how you can fix it.

> **Note:**
>
> When you're working with code editors (such as Code view in the Page Designer or the JavaScript editor), audit markers show in the file even after you make changes to resolve issues. They disappear only after the file is revalidated and audit results are regenerated, if the issue has been fixed.

- Optionally, select **Do not report this type of defect again** to ignore similar defects in future.

Once you make a choice, your application is rescanned and the audits results updated.

> **Tip:**
>
> If you want to copy an issue to your clipboard for further processing, right-click the issue and select **Copy to Clipboard**. To copy all issues in a particular artifact, use the right-click menu at the artifact level.

Edit Audit Rules

The Audits feature in the Designer includes built-in rules from the Oracle JET Audit Framework (JAF) by referencing the rule pack and the JAF utility (ojaf) hosted on the Content Delivery Network (CDN) at `https://static.oracle.com/cdn/jet/`.

Every JET release includes the JAF utility and JAF metadata, plus the JAF metadata for previous releases of JET. You can configure the built-in JAF rules to include disabled rules. Custom JAF configurations that deal with the file system, custom rule plug-ins, and so on will not be evaluated because JAF does not execute on the Visual Builder backend; it runs on the client.

Here's how to enable a built-in rule that is disabled by default:

1. In the Audits pane, click **Menu** (•••) and select **Edit Settings**.
2. Edit the `audit.json` file in the Source editor, for example, here's the syntax to reference JAF from the CDN and enable a disabled built-in rule:

```
{
    "paths": {
        "exclude": [
            "build/**",
            "docs/**",
            "scripts/**",
            "tests/**",
            "**/private",
            "+(web|mobile)Apps/**/resources/components/**/lib/*"
        ]
    },
    "rules": {},
    "auditors": {
        "jaf": {
            "cdnPath": "https://static.oracle.com/cdn/jet/",
            "version": "9.0.0",
             "jafOptions": {
                "ruleMods": {
                    "JET": {
```

```
                    "oj-css-style-override": {
                        "enabled": true
                    }
                }
            }
        }
    }
}
```

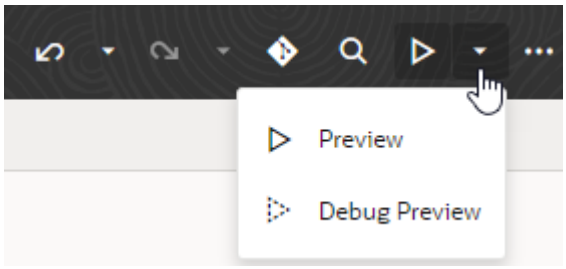
To revert the built-in rules to their default settings, click **Reset Settings** in the Menu (•••).

For more information about JAF, including the built-in rules that it includes, see About Auditing Oracle JET Applications in *Using and Extending the Oracle JET Audit Framework*.

Preview an App in Debug Mode

When testing your application, you typically preview it to see it the way your user would. Sometimes though, you might want to preview the application in debug mode to troubleshoot issues with the Visual Builder runtime and Oracle JET libraries—runtime dependencies that make sure your app works as intended.

The **Preview** option in the header facilitates both modes:



The default Preview mode uses the optimized variants of the VB runtime and JET libraries. In this mode, all unnecessary characters (such as whitespaces and comments) in the application's source code are removed and variable names shortened to minify code and optimize performance.

The Debug Preview mode, on the other hand, uses the debug variants of the VB runtime and JET libraries. In this mode, line breaks and white spaces are preserved, allowing you to view the application's source code in a more readable format. You can then use your browser's debugging tools to step through your code and figure out exactly where an error occurred.

Because no performance optimization is done in debug mode, previewing an app in this mode can be misleading about how quickly—or slowly—an app opens. As a result, you might want to use debug mode only in a development environment.

- To view your app like a user would with the pages and data displayed, click .
- To view your app in debug mode, click the Preview drop-down, then select **Debug Preview** .

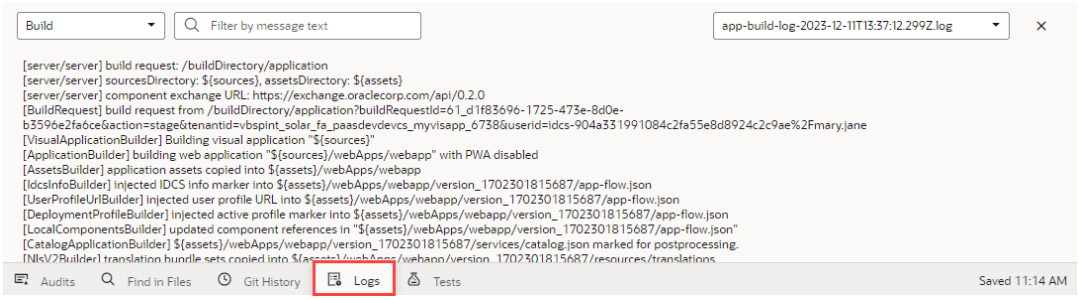
Troubleshoot Build Issues

If you run into errors when you try to stage or publish your application, you might need to check the build logs to troubleshoot build-related issues that prevent your app from being staged or published.

Build logs are available in the Logs tab (at the bottom of the window), where you can view up to five of the most recent logs. Logs are generated when your application is successfully staged or published, but you are not explicitly notified. However, when build issues prevent your app from being staged or publishedshared or deployed (for example, when a missing JavaScript library hinders the app's sources from being bundled, or when errors exist in `flow.json`, or if optimized application configuration is not correctly defined in `build.json`), you might see an error message in the Stage Application dialog, with a pointer to open the build logs.

Typically, most build errors occur when you stage your app and must be resolved before you publish it. Here's what to do when you encounter build errors:

1. Click **Open build logs** in the dialog.
2. When the build log opens in the Logs tab, look for errors in the `---Build Error Start---` section, then take steps to resolve the issues. Use the options in the right-click menu to copy and paste messages as needed.



   If you want to view older logs, click the `app-build-log-`*`timestamp`*`.log` list on the right and select the log whose contents you want to see.
3. After resolving the issues, stage or publish your app again.

Debug Business Objects

Because Visual Builder uses a multi-tier architecture, you might need to debug your application at different levels to identify the root cause of an issue. This might include the business object layer, where data access to and from a client application occurs through REST API endpoints.

While you can use the Network tab in your browser's development tools for external REST APIs, Visual Builder's built-in tracing and logging mechanisms can help you troubleshoot issues when you create business objects as the data source for your user interface. Because data from a business object is written to the UI via REST APIs, you might want to enable tracing to track the response times of individual REST calls. You can also enable logging to view events triggered by your business rules, in addition to diagnostic messages logged by custom Groovy scripts.

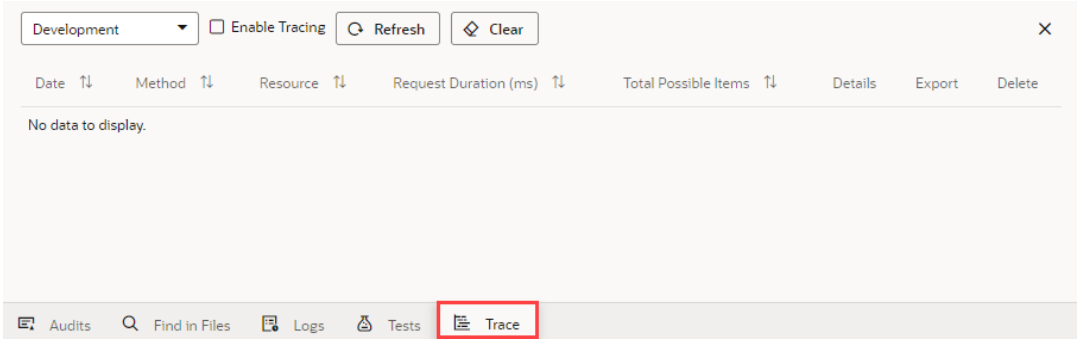
Enable Tracing to Monitor Endpoint Calls

When your application contains one or more business objects, enable tracing of an object's endpoint requests to diagnose performance bottlenecks.

Tracing tracks all REST requests made when the current user executes CRUD operations or invoke functions on business objects. It provides a visual representation of the operations taking place and the time it took to invoke each REST call—information you can use to locate bottlenecks in your application. You also enable tracing separately for a particular version of your application (development, stage, or live), so you can isolate issues in that version and fine-tune your app for better performance.

To enable tracing for an application:

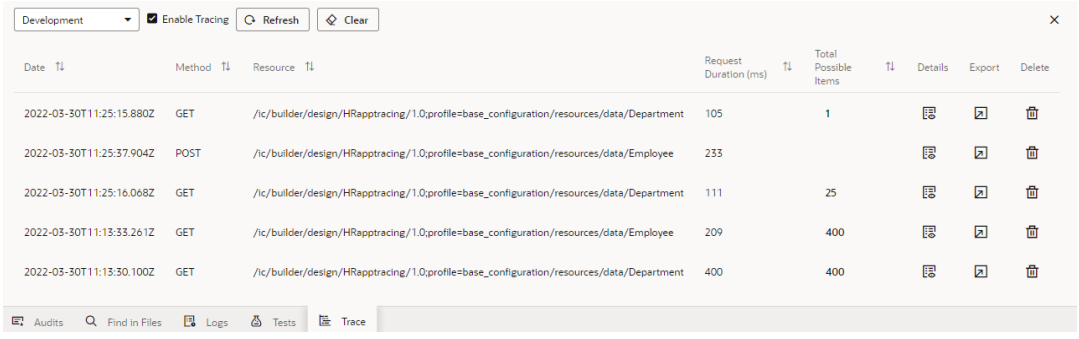
1. Click **Trace** at the bottom of your application window.



2. If your app has staged or live versions, select the version you want to enable tracing for.
3. Click **Enable Tracing**.

   Once tracing is enabled, all REST requests made by the current user are traced, both when running the app using **Preview** or when making updates in the **Data** tab. If you selected a staged or live version of your app, endpoint requests made at runtime are traced. (You'll need to be the app designer or a team member assigned to the app to view traces for these runtime requests.)

   Click **Refresh** to view the latest traces.



   A trace is a collection of operations for an application transaction. When you edit business object data on a page, your application interacts, for example, with the object's GET endpoint to request the data to display, and then with the PATCH endpoint to update the

data. You'll see each endpoint request as a separate entry, with the time spent by your application processing the REST request shown in the Request Duration column.

For GET requests, you'll also see an indicator of the maximum response size in the Total Possible Items column. This value is a theoretical maximum number of items in the response payload for the business object and for any child items returned by referenced business objects, for example, in the `fields` query parameter. It is calculated based on the `limit` query parameter and the relationships in the request and can help you determine whether you need to take steps to reduce the response size. (Click the Details icon ( ) to view additional performance metrics in the REST Request Info tab.)

> **Note:**
>
> In a cluster with multiple nodes, a REST API request (particularly one made by tools such as cURL or Postman) won't be traced if it was handled by a node other than the one on which you enabled tracing. To avoid this issue, make sure you use the browser on which you enabled tracing to also make your REST requests.

View Trace Details

You can view a particular REST request's trace for details such as span data and elapsed time. Each trace consists of one or more spans and you can drill down an individual trace to view its root span, which is the beginning of a transaction. You can also view Groovy logs if you included the `print` or `println` function in custom Groovy code.

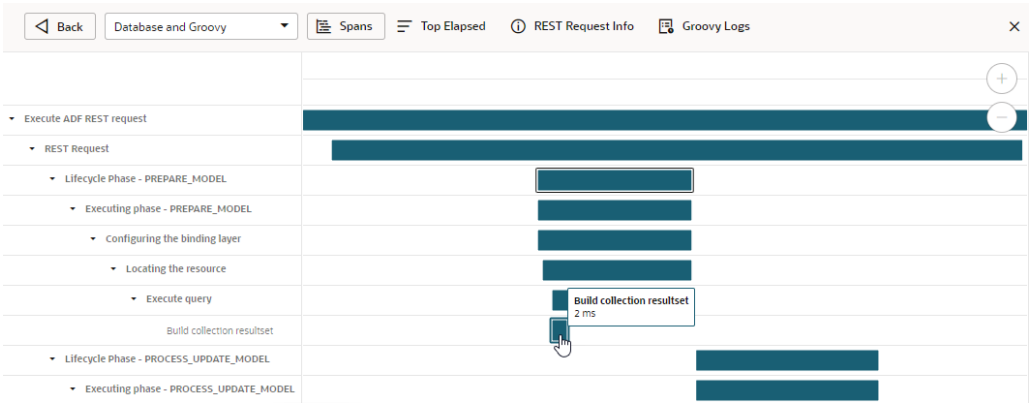
To view details of an individual trace:

1. Click the Details icon ( ) for the REST request you want to view.

   The REST request's tracing data is broken down and displayed in different tabs.

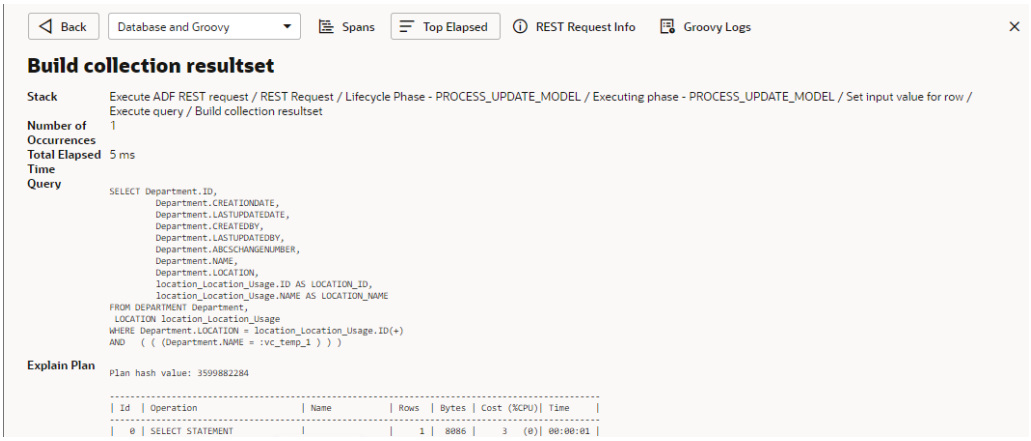
2. Click **Spans** and **Top Elapsed** to view the request's tracing data.

   - The **Spans** view shows a Gantt chart that visually represents the steps in the REST request's execution path. You can hover your cursor over each step's bar graph to see the time the step took.



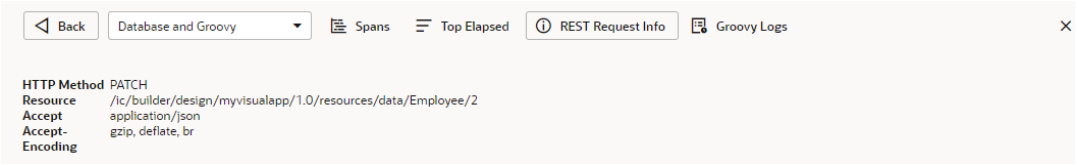
   To drill down and see details of a selected span and its children, click a span to select it, then right-click and select **Top Elapsed of Selected Span** to use the span as the context for the **Top Elapsed** view.

- The **Top Elapsed** view shows operations performed during the request. Repetitive operations, for example, repeated calls to the same SQL Select statement or execution of the same Groovy function, are aggregated.

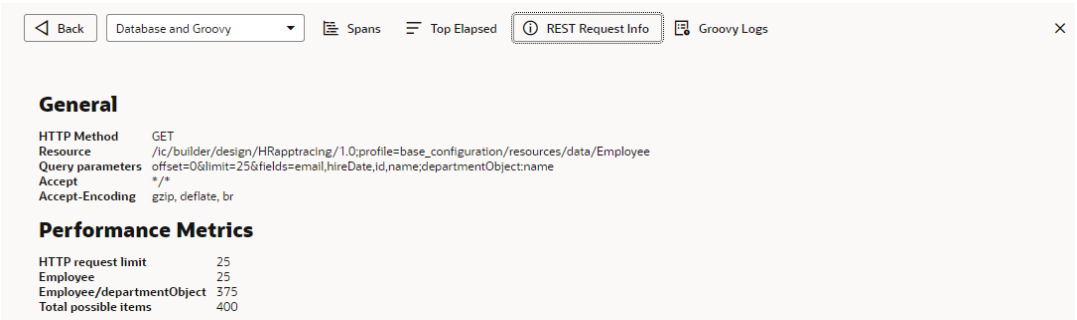


You can filter operations in your Spans and Top Elapsed views by selecting **Groovy** or **Database** in the drop-down list when in the **Spans** or **Top Elapsed** tab.

3. Click **REST Request Info** to view a summary of the REST request.

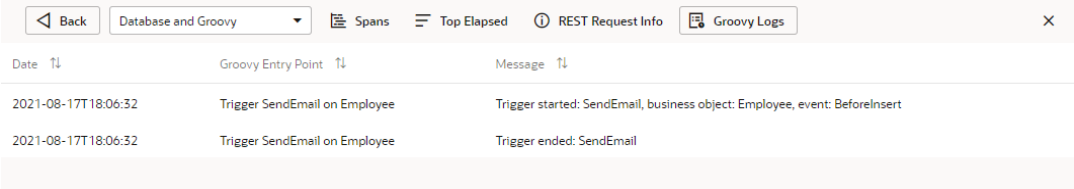


For GET requests, you can view additional **Performance Metrics** to get a breakdown of the **Total Possible Items** column, which provides a theoretical maximum number of items in the response payload based on `limit` or `expand` query parameters or any accessors in the `fields` query parameter:



Use these metrics to determine whether a request that does adequately in testing might have a performance issue with a different data set (for example, live). Because the ideal response size for any request is the smallest possible size that contains all the information the client needs, you can take steps to reduce the response time (say) by requesting less data or by using pagination (where multiple smaller requests using a lower `limit` query parameter and incrementing the `offset` query parameter may help).

4. If you included log messages in your Groovy scripts (for example, by adding `print` or `println` statements), click **Groovy Logs** to view the messages your script has generated.



Manage Tracing to Control Disk Usage

When tracing is enabled for an app, all its trace files are stored on Visual Builder server. To avoid disk-usage issues on the server's file system, Oracle sets a maximum disk space limit for all trace files. Once this limit is reached, the oldest trace files are removed to make way for new trace files.

Here are some options to help you manage your app's trace files:

- Deselect **Enable Tracing** to pause tracing. Use this option to control the accumulation of trace files when you're not actively tracking your app's REST calls.
- Click the Export icon ( ) to export the trace files for an individual REST call. Use this option to save the files to your local file system and import it later when required. This way, you won't lose the data even if trace files hit the server's disk-usage limit.
- Click the Delete icon ( ) to delete trace files for individual REST calls. Use this option to remove trace files you don't need and clear up space.
- Click **Clear** to remove all trace files for the app's current version. Use this option to delete tracing data for a particular version of an app. For example, when you're more interested in data for the staged and live versions of an app, you can clear trace files for the development version to remove previous data that might be taking up disk space on the server.

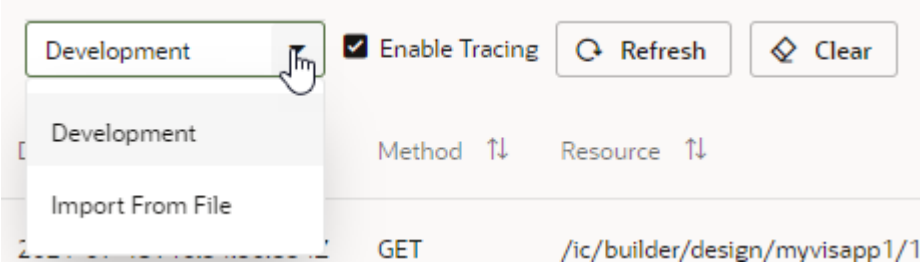
Export and Import a Trace File

You can export a particular REST call's trace files to your local file system to avoid losing data if the server reaches its disk-usage limit for trace files. Then, when you are ready to analyze the data, you can import the file back in to Visual Builder. Importing a trace file is a browser function that doesn't affect disk usage on the server.

To export and import a trace file:

1. Click the Export icon ( ) for the REST call whose trace file you want to export.

   The trace file is saved as a .JSON file in the directory specified for your browser's downloads.
2. When you are ready to import the file back into the system, click **Import From File** in the drop-down list.



3. Click the upload box and navigate to the .JSON file you previously exported. You can also drag and drop the file in the upload box.

   The file's contents display in the Trace panel. You can now view the trace's details and spans.

Enable Logging for Scripting Events

To assist with debugging when developing rules for a business object, you can enable logging and use the log viewer to view events triggered by your business rules. You can also view runtime exceptions as well as diagnostic messages that your own Groovy scripts might have generated.

While trigger starts and trigger ends are always recorded in logs, you'll need to add the `print` or `println` statement when you want messages in your script to be written to the log.

By default, logging is not enabled. Once you enable logging, it remains enabled in your session for as long as you are logged in. If your session has expired, you will need to re-enable logging after you log in.

To enable logging:

1. Click **Logs** at the bottom of the window to open the Logs page.
2. Select **Enable Logging**.
   The viewer in the Logs window displays the most recent 250 log entries, by default in chronological order. When a runtime exception occurs, you can see additional details about the offending script and line number where the error occurred in a tooltip when you hover your mouse over the exception message.

   Use search to filter out messages based on the text you enter. To export your log as a text file to your local system, click the Export icon ( ) in the toolbar.

   If you keep the **Logs** window open while you work, consider the following approach.

   Before starting a new attempt to reproduce the problem, click the Clear icon ( ) to remove any previous messages generated. After encountering the error you are diagnosing, click the Refresh icon ( ) to see the latest log messages generated.

> **Note:**
>
> You can also view log messages included in your Groovy code as part of trace files generated for a business object's REST requests when tracing is enabled.

Change an Application's Log Level

You might sometimes want to change the app's log level to change output in the browser console. For example, the default `info` log level for console messages may be useful during development stages, but once the app goes into production, you might want to reduce console output to only critical information.

1. To change an application's log level:
   a. On the **Web Apps** tab in the Navigator, select your web app, then click the **JSON** tab to open the `app-flow.json` file. Alternatively, switch to the app's **Source** view and locate the `app-flow.json` file under `webApps`.
   b. Add the `logConfig` snippet to the file:

      ```
      "logConfig": {
        "level": "loglevel"
      },
      ```

      where *loglevel* can be `warn`, `error`, `info`, `fine`, or `finer`. For example:

      ```
      "logConfig": {
        "level": "error"
      },
      ```

      You can also use an expression. For example, if you've defined `window.myConfig` as a global variable, you can define the log level as:

      ```
      "logConfig": {
        "level": "{{window.myConfig?.VB_LOGCONFIG_LEVEL ?
      window.myConfig.VB_LOGCONFIG_LEVEL : 'error'}}"
      },
      ```

      where the log level is `window.myConfig.VB_LOGCONFIG_LEVEL`'s value. If `window.myConfig.VB_LOGCONFIG_LEVEL` isn't defined, the log level is set as `error`.
2. Run your app and view output in the browser's console.
3. It's also possible to dynamically set the app's runtime log level in your browser's current session to temporarily troubleshoot issues. You do this using the `globalThis.vbInitConfig.LOG.level` property in your browser's console. Here's how to do this in Chrome:
   a. Open Chrome DevTools.
   b. Click the **Console** tab.
   c. Enter the following command in the console:

      ```
      globalThis.vbInitConfig.LOG.level = 'loglevel'
      ```

where *loglevel* can be `none` (least verbose), `error`, `warn`, `info`, or `finer` (most verbose). The ideal level is `info`, which is a good balance between least and most verbose. For example:

```
globalThis.vbInitConfig.LOG.level = 'info'
```

d. To dynamically set the log level for JET elements, you use the `sessionStorage.setItem()` call in your browser's console.

   This setting requires your application to be on JET 14.0.6 or higher. Check the Settings editor to make sure your app uses a supported version. Upgrade your app, if needed.

   Enter the following command in the console:

   ```
   sessionStorage.setItem('ojet.logLevel', 'loglevel')
   ```

   where *loglevel* can be `none` (least verbose), `error`, `warning`, `info`, or `log` (most verbose). For example:

   ```
   sessionStorage.setItem('ojet.logLevel', 'error')
   ```

   > **Note:**
   >
   > If the JET log level is set to `info` and the runtime log level is set to `error`, JET logs will not show. You must use the same value for runtime and JET logging to see both logs.

e. Refresh your browser to view updated information in the console.

f. To reset the log level for your browser's current session, enter `sessionStorage.removeItem("vbSessionLog")` for runtime logs and `sessionStorage.removeItem("ojet.logLevel")` for JET logs in the console, then refresh your browser.

26

Stage and Publish Visual Applications

You can stage and publish visual applications from the Visual Builder Home page as well as from the Designer.

> **Note:**
>
> Staging a visual application that contains a mobile application requires you to enable your mobile app as a PWA, then build it to generate a QR code. Building your PWA-enabled mobile application deploys your visual application in Development to Stage. See Run Mobile Applications as PWAs.

What Happens When You Stage and Publish Visual 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.

> **Note:**
>
> If you want to deploy a visual application to multiple instances, the best option is to use VB Studio to set up additional deployment instances. See Add Additional Deployment Instances.

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. The web app accesses 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 permanent URL is created for accessing the live web app. The web app accesses 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 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 to requests to services in another domain and you cannot modify its allowlist. 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

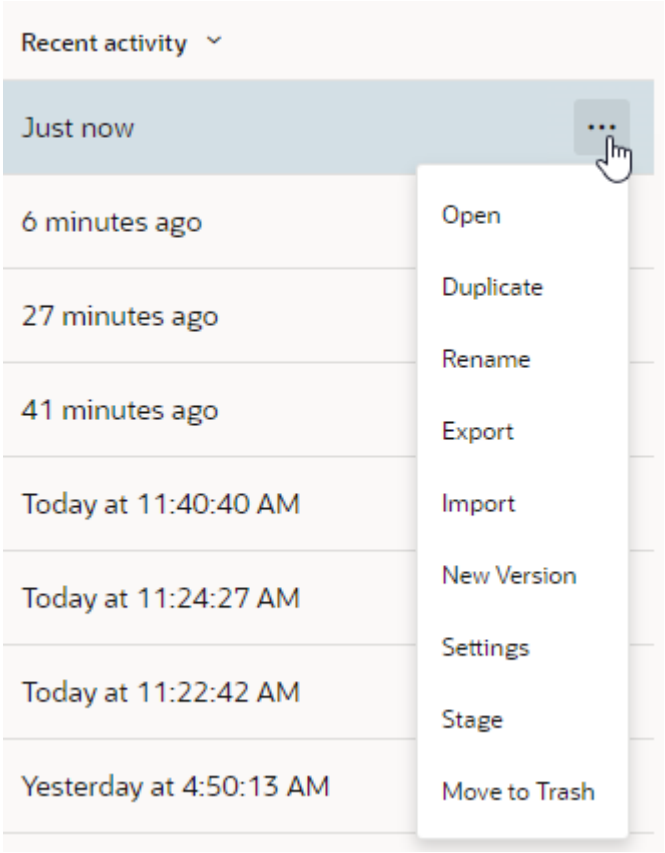
You can stage unpublished versions of your application at any time from the Home page or from the main menu.

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 Preview button in the Designer. You can't share the URL of the app preview with other team members.

> **Note:**
>
> If you want other users (including admins) to edit an app or perform operations such as staging, they must be added to the app as a team member (see Add Team Members).

To stage your application:

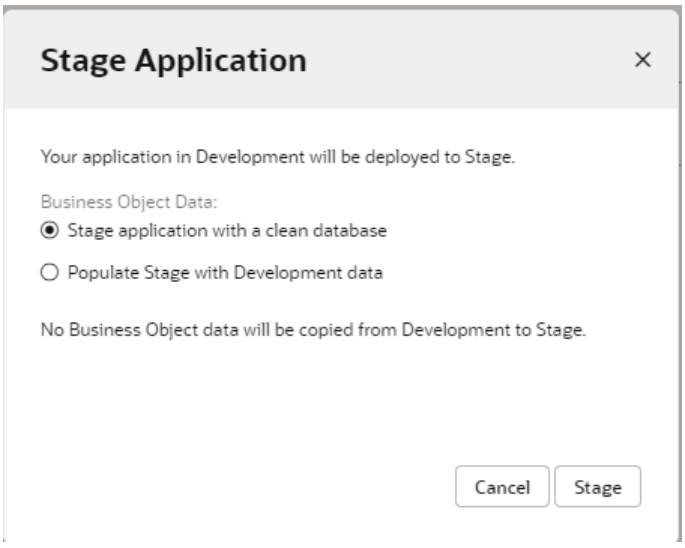
1. On the Home page, open the Application Options menu of the application and click **Stage**.



If your application is open in the Designer, select **Stage** from the visual application's Menu in the toolbar.

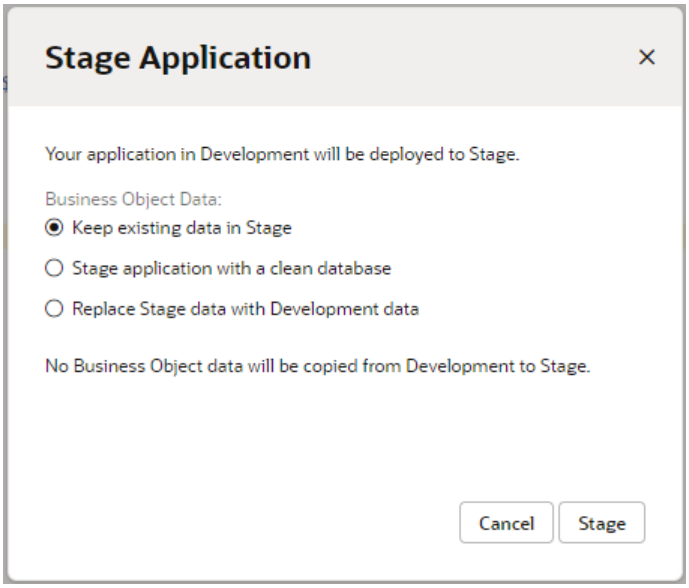
2. In the Stage Application dialog, select a database option. Click **Stage**.

   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.



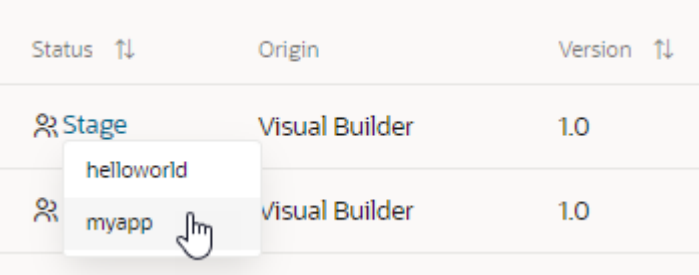
After the initial staging, each time that you stage your application you need to specify how you want to manage the data in the staging database. You can choose to keep the data,

replace the data with data from the development database, or delete all data in the database and start with a clean database.



> **Note:**
>
> If your application cannot be staged, click **Open Logs** in the Stage Application dialog to view and fix build-related errors.

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 becomes read-only and can no longer be changed. If you want to make changes to update the app, you need to create a new version (see Update a Published Visual Application).
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 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.

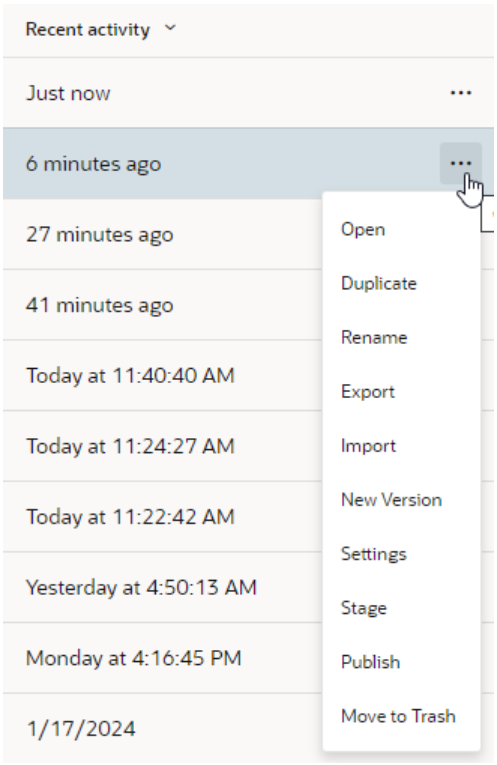
After an app is published, the resources used by the app (for example, metadata, images, stylesheets) will not change until you publish a new version. The resources of the published app are cached on the client and are fetched from the local cache instead of retrieved again from the server. An app's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

If the application you want to publish includes a mobile app, make sure you build the PWA-enabled mobile app to generate a QR code (see Build a Mobile Application as a PWA).

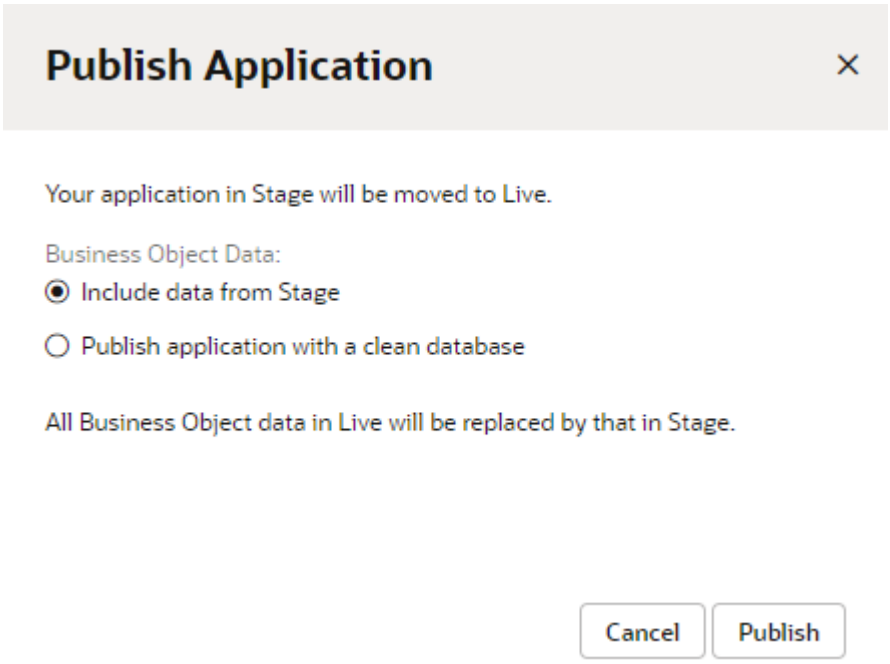
To publish your application:

1. On the Home page, locate the application you want to publish, then click the Application Options menu and select **Publish**:

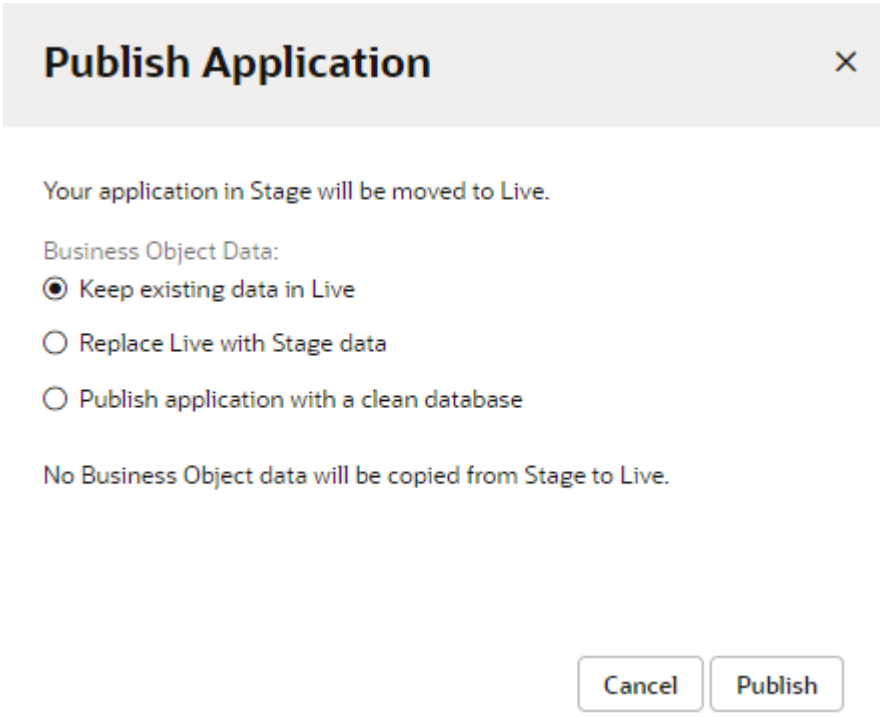


If your staged application is open in the Designer, select **Publish** from the visual application's Menu in the toolbar.

2. In the Publish Application dialog box, select an option for handling the business object data:



If you are updating the live version of the app, each time you publish you need to specify whether you want to keep the data in the live database, replace it with data from the staging database, or delete all data and start with a clean database:



The schemas for your app's databases (development, staging, and live) typically change when you create new versions. While you can't change the name the schema is created under, once a live app's schema is created, it will stay the same as long as you select **Keep existing data in Live** when you publish a new version. Use this option to retain the database schema over subsequent deployments to the same environment.

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

The application's status changes to Live in the Status column. To view the published app in your browser, click **Live**, then select your application. The application opens in a new browser tab. You can now share this URL with your users.

View Database Schemas Used During an App's Lifecycle

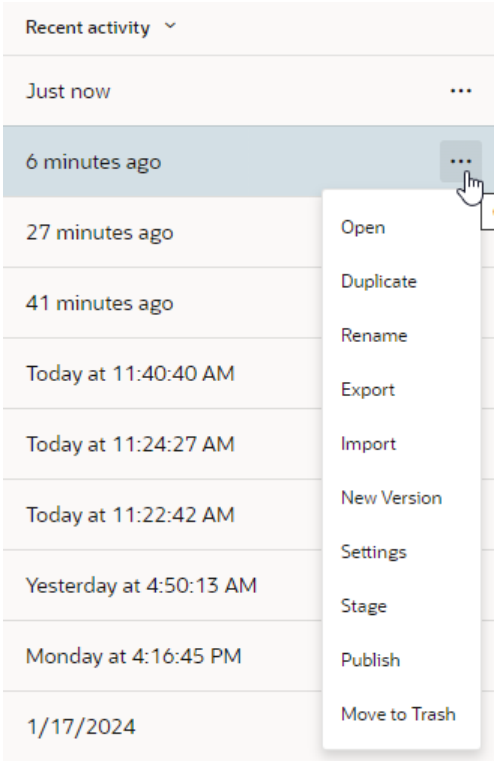
As your application progresses through its development lifecycle, you might want to know which database schemas store the app's business objects in each phase (development, stage, and live).

Visual Builder automatically manages the schemas and tables for apps and business objects in your database. It also re-creates the development and stage schemas with different names for every new version of your app. (The live schema for a published app doesn't change if you choose to not replace the data.)

If you are using your own Oracle database, knowing the name of the schema associated with each phase of your app can help you access data using other tools for import or export, or access the tables for your own purposes. You also have the option of using application profiles to switch the schema for each phase.

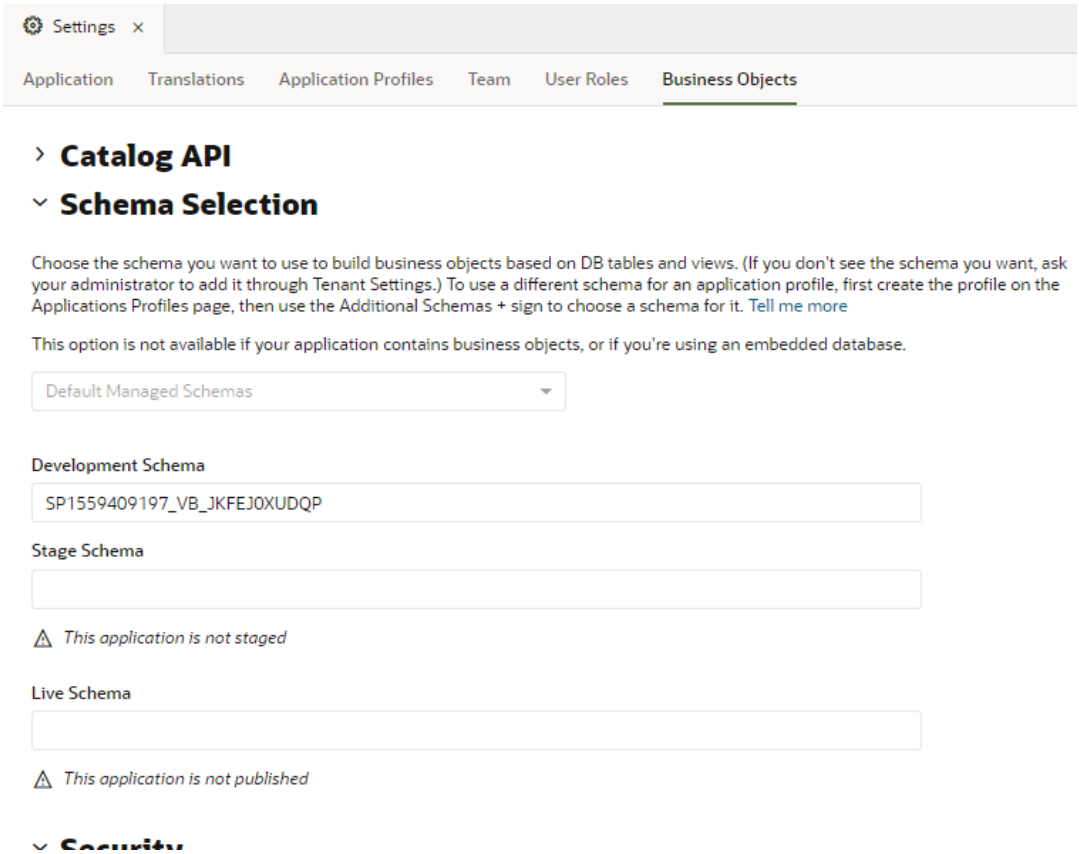
To view the schema associated with a particular phase of an application's lifecycle:

1. On the Home page, locate your application, click the Application Options menu, and select **Settings**:



   Alternatively, click the visual application's Menu option in the upper right corner of the Designer and select **Settings**.

2. Select the **Business Objects** tab in the Settings editor.
3. Under **Schema Selection**, look for the schema associated with each development phase. For example, to check the name of the schema used by your development database, look in the **Development Schema** field.



Schema details show only when a schema exists. So if you are yet to create a business object for your visual application, you won't see the Development Schema field. Similarly, you won't see details in the Stage Schema and Live Schema fields until your app is staged and/or published. The Stage Schema also won't show when you're viewing the Live version of the app.

Specify a Custom App URL

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

To use a custom domain for your application, your service administrator must configure your instance to support the custom domain. For the steps to do this, see Configure a Custom Endpoint for an Instance in *Provisioning and Administering Oracle Integration Generation 2*.

After the visual application is staged and published, the web application and the business object APIs can be accessed directly using the custom domain. You can also access web applications deployed as PWAs.

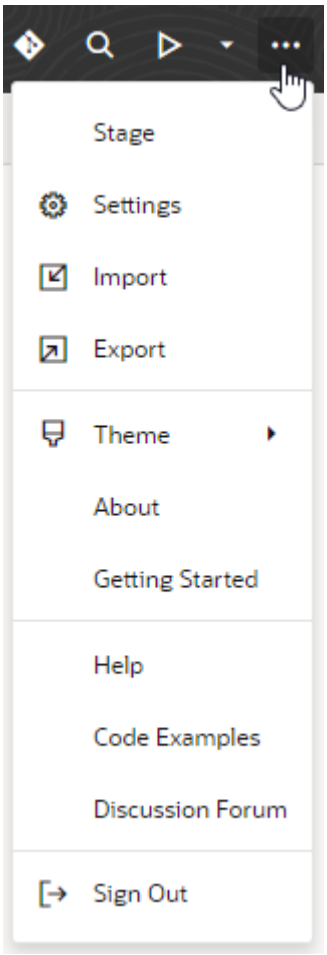
Note that only one custom domain can be mapped to a visual application, and it can only be used to access one web application in the visual application. To ensure that the correct web application is loaded when using a custom domain, your visual application must contain only one web application.

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`).

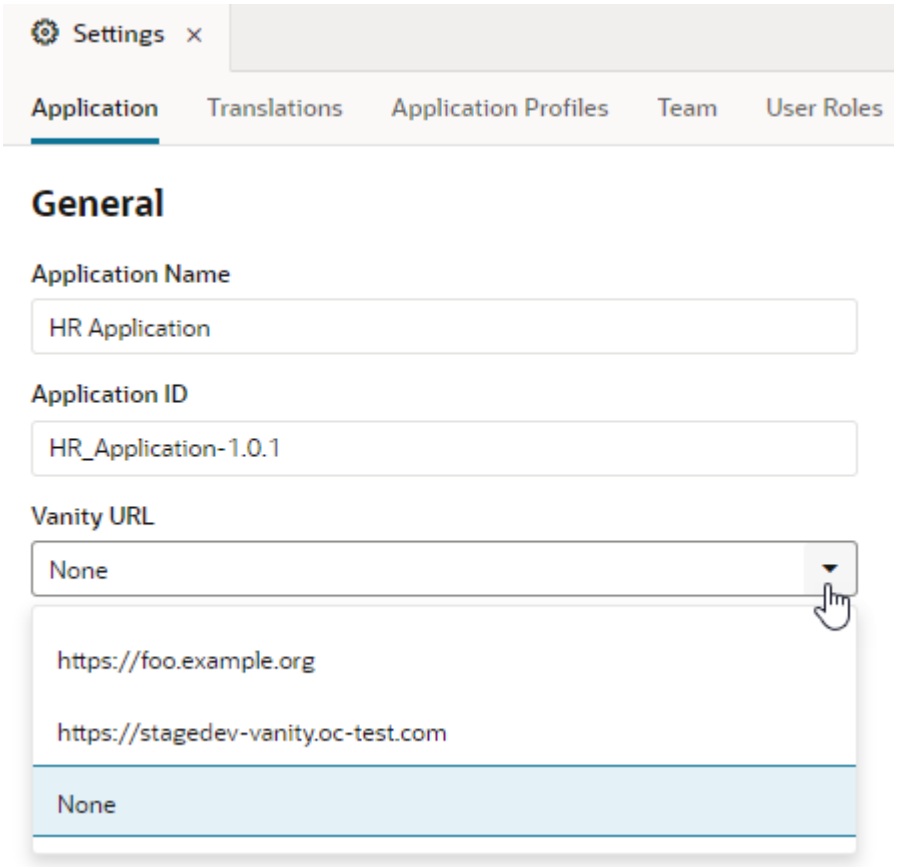
To map a custom domain to a visual application:

1. Open your visual application, then click Menu in the upper right corner of the header and select **Settings**.



   Alternatively, on the Visual Builder Home page, locate the visual application where you want to change the settings and choose Settings in the Application Options menu.

2. In the Settings editor's **Applications** tab, select the custom domain you want to use in the **Vanity URL** list.



If no custom domains are listed, your instance isn't configured to support custom domains. Talk to your service administrator to enable this functionality.

After you've set the vanity URL, it's important you work with the visual app using its custom domain. For example, if the app's vanity URL is `https://foo.example.org`, access it in a browser at `https://foo.example.org/ic/builder`, then proceed to stage and publish the app.

> **Note:**
>
> Working with a vanity app from a non-vanity host can cause issues when you try to access the deployed app, so make sure you access the Designer using the app's custom domain.

After you publish the visual application, a visitor can enter 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.

Update a Published Visual Application

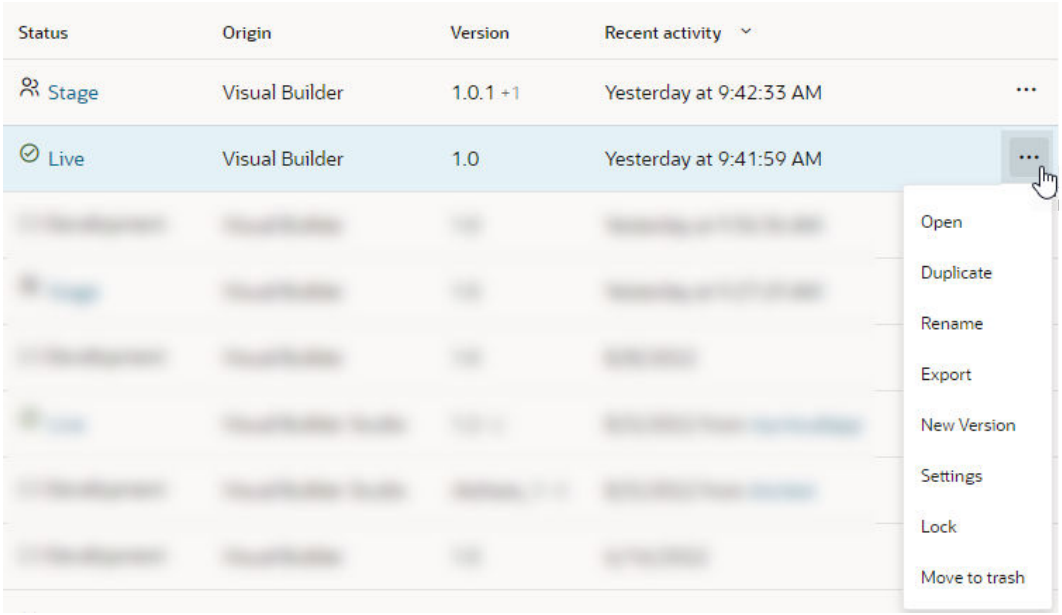
If you want to make changes to a published app, you need to create a new version. This creates a development version of the app for you to work on while the published version stays

live. Once you are ready to go live with your updates, you can re-stage and re-publish the new version.

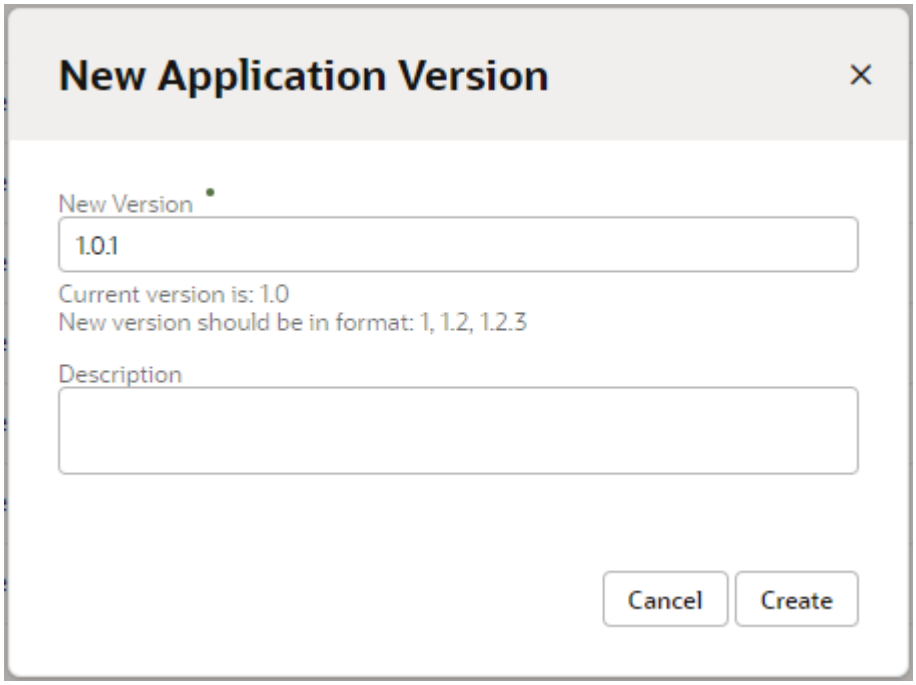
Staging the new version will generate a new URL that you can use for testing the app. Publishing the new version will replace the published version of the live app on the existing URI. This way, your customers can access the updated app with the same URI they've been using all along.

To create a new version of a published application:

1. On the Home page, locate the live version of the application you want to update, then click the Application Options menu:

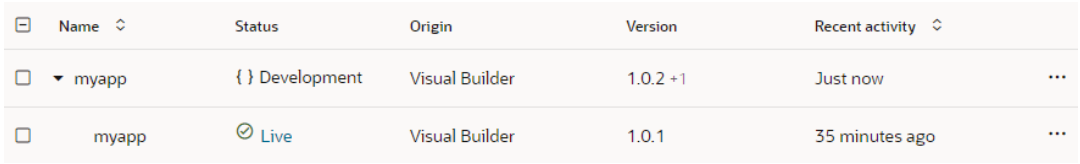


2. Select **New Version**.
3. In the New Application Version dialog box, enter a new version and enter a description to help track your changes. You can number your versions according to your versioning scheme, but the version number must be unique.



4. Click **Create**.

   A new Development version of your app is created on the Home page:



> **Note:**
>
> If you're unable to create a new version, it may be that your application has reached its total limit of 100 versions. Check the application's Versions column on the Home page, where you can see the number of versions next to the current version number, for example, `2.8.9 +100`. If the number of versions is 100, you need to permanently delete some versions of your app, starting with the oldest obsolete version (see Delete a Visual Application). Once you've deleted enough versions to fall within the version limit, try creating a new version again.

After you've successfully created a new version, click the Development version and update it in the Page Designer as required. When you're done making updates, stage and publish the app again.

Roll Back Application to the Previously Published Version

When you create a new version to update an application's published version, you re-stage and re-publish the new version to make your changes live. If you want to revert these changes for

some reason, you can roll back the application from the current live version to the previously published version.

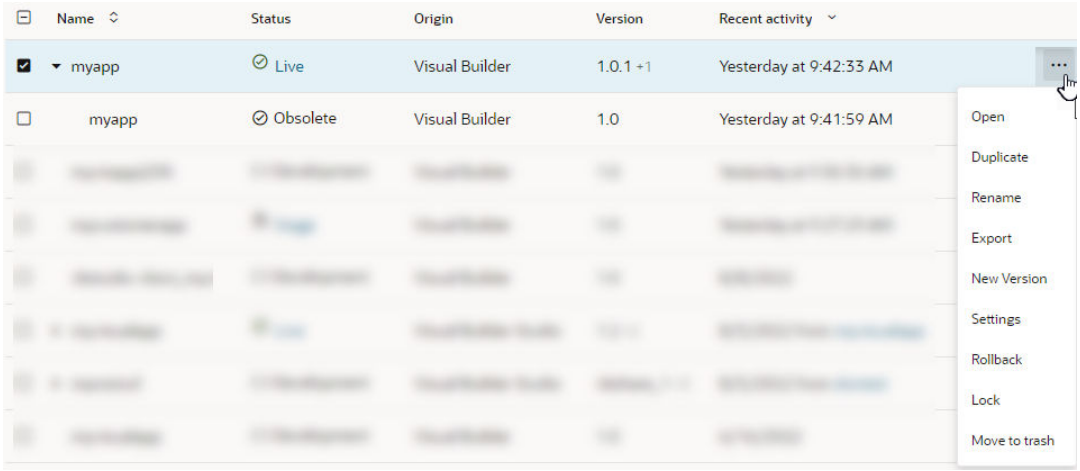
Publishing a new version of a published version (say, 1.0.2 after 1.0.1) renders the previous live version obsolete. If you decide to roll back from 1.0.2 to 1.0.1, the obsolete version becomes the live version again. This reverts all your application's content, including any database schema changes.

> **Note:**
>
> When you roll back to a previous version, the data of the previous live application is restored. So, if the new version was accidentally published with the **Publish application with a clean database** option, the rollback recovers the database schema of the previous version.

Keep in mind that you can only roll back an application to its last published version. So if your application includes a 1.0 version that was previously published and is now obsolete, you can't roll back from 1.0.2 to 1.0.

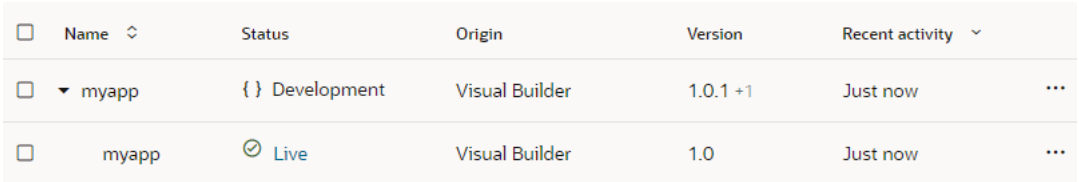
To roll back an application's current live version to its previous live version:

1. On the Home page, locate the live version of the application you want to roll back (for example, version 1.0.1), then click the Application Options menu:



2. Select **Rollback**.
3. When prompted to confirm, check the versions and click **Rollback**.

   The current live version (for example, version 1.0.1) is set as Development and the previous published version (for example, version 1.0) is set as the Live version, as shown here:

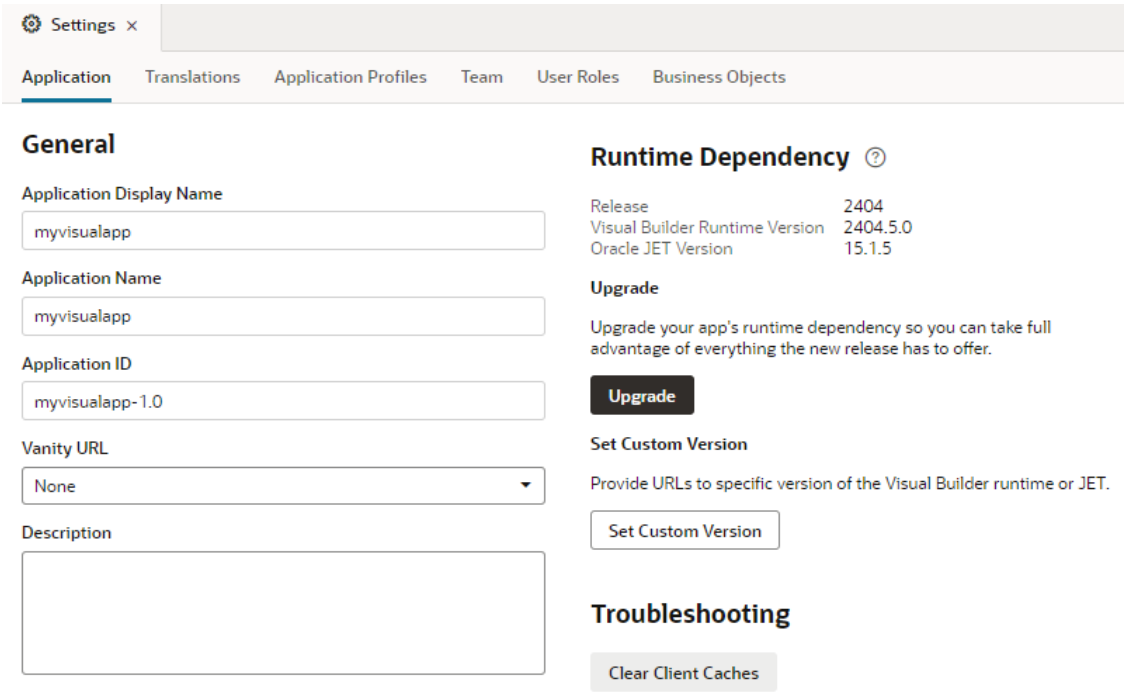


27

Manage Runtime Dependencies for Visual Applications

In Visual Builder, **runtime dependencies** refer to a set of client-side libraries that, along with the accompanying version of Oracle JET, determine features and other improvements available to your visual application, like what JET components you can use.

To see what the runtime dependencies are for your visual app, click **Menu** in the upper right corner, then select **Settings**:



A visual app's runtime dependency comprises three values:

- **Release**: The latest certified combination of Visual Builder Runtime and JET within a major release, such as `24.10`, `24.07`, or `24.04`.
- **Visual Builder Runtime Version**: A set of client-side libraries hosted on a Content Delivery Network (CDN). These libraries help the constituent files in your app talk to each other at runtime. For example, when you include a component (like a button) from the Component Palette in your app, the code for that component is stored in your app's HTML file. If you then add an action chain to that button to navigate to a new page, the action chain code is stored in your app's JSON metadata file. At runtime, the Visual Builder Runtime enables the HTML, JSON, and other dependent files in your app to communicate with each other, so that your app behaves as intended.
- **Oracle JET Version**: A JET release hosted on CDN. Each Visual Builder Runtime Version is certified to be compatible with one or more JET versions. If you'd like to see what's in the latest JET release before deciding to upgrade, go to JET Release Notes and select the JET version stated on the Settings page.

Runtime dependencies are set for a visual application as a whole; you can't set different versions for individual web (or mobile) apps within the visual app.

When you create a new visual app, Visual Builder automatically sets your runtime dependencies to the latest Visual Builder Runtime and JET versions. If you've already staged or published an app, however, it's up to you to decide when to upgrade, as long as you do so within a certain time period.

As a general rule, you can run a published Visual Builder application built on the current runtime version, and continue running it on the three previous versions—but new **Visual Builder** *runtime* libraries are released with new versions of the **Visual Builder Studio** *design time* (around four times a year). The Visual Builder design time is only updated twice a year, so it gets only two of those four versions (another reason we recommend using Visual Builder Studio as your design environment). As a result, you need to upgrade every other release in Visual Builder, while in Visual Builder Studio, you upgrade after four releases, with the design time supporting the last three versions.

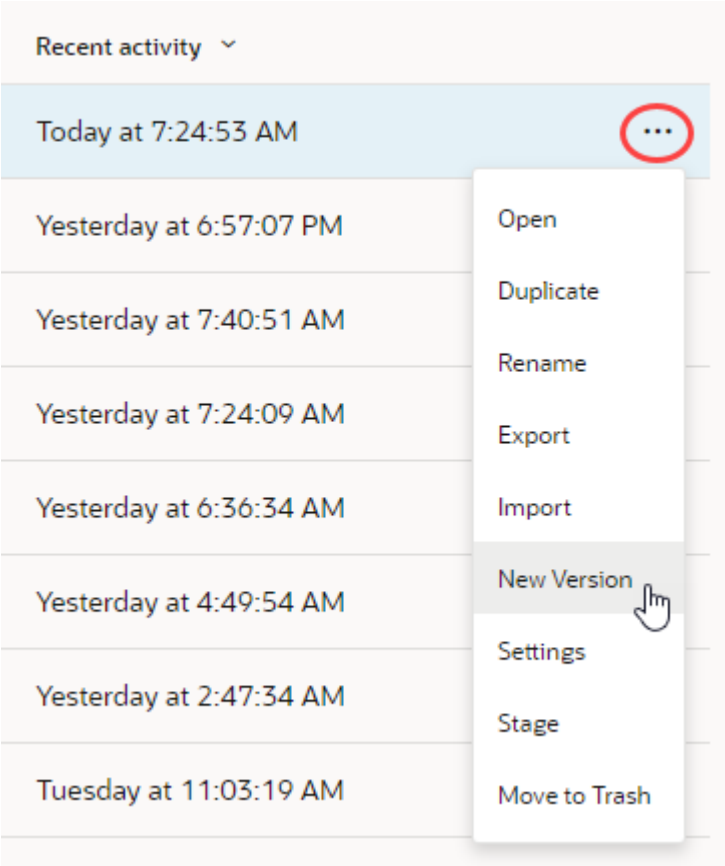
So if you built your app in 24.10 (for example), Visual Builder supports the app not only on the 24.10 runtime version, but also on 25.01, 25.04, and 25.07. Once 25.10 comes out, however, support for the 24.10 runtime version will drop off, so we'll ask you to upgrade your app before you can work on it in the Designer. If you choose not to upgrade at that time, you run the risk that newer browser versions will break your app. You also won't be able to take advantage of any important security and performance improvements. For all of these reasons, we encourage you to build time into your development cycle to keep abreast of current changes, and to make sure you upgrade your app (you should version it first) **before** support for your current runtime version expires.

Upgrade Your App

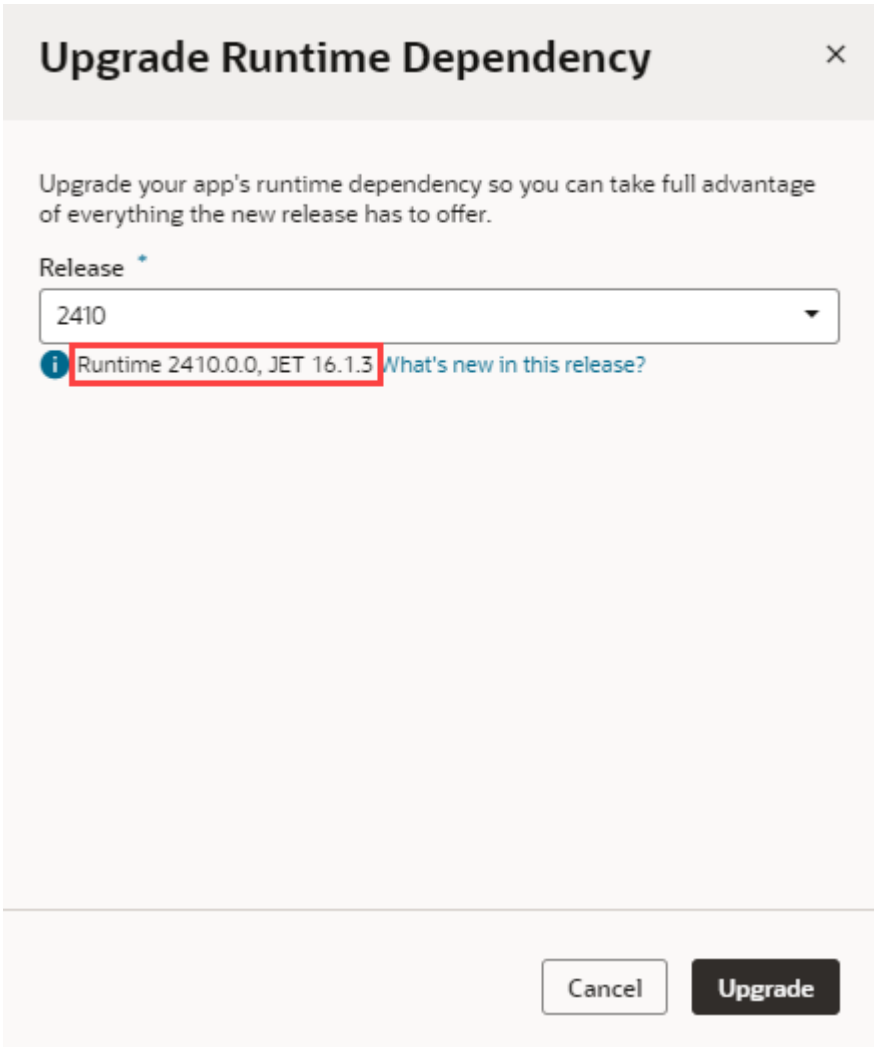
If you see a banner in the header telling you to upgrade your runtime dependencies, here's what to do:

> **Note:**
>
> You don't have to wait until you see a message to upgrade your app. If the **Upgrade** button under Runtime Dependency (on the Settings page) is active, that means you can—and should—upgrade as soon as you can. Once the message appears in the header, you're likely close to the end of your window according to Visual Builder's upgrade policy. Be sure to create a new version of your app before upgrading it.

1. Click **Oracle Visual Builder** in the top left corner to return to the Visual Applications Home page.
2. Click the Applications Options menu and select **New Version**:



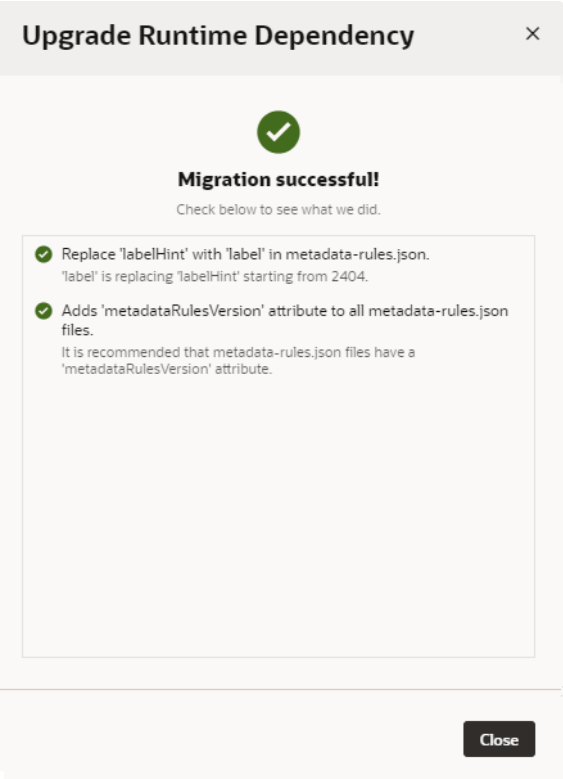
3. In the New Application Version dialog, enter a new version for your app, then click **Create**.
4. Click your app to re-open it in the Designer.
5. Click **Upgrade** in the banner.
6. In the Upgrade Runtime Dependency dialog, select the release you want to move up to, then click **Upgrade**:

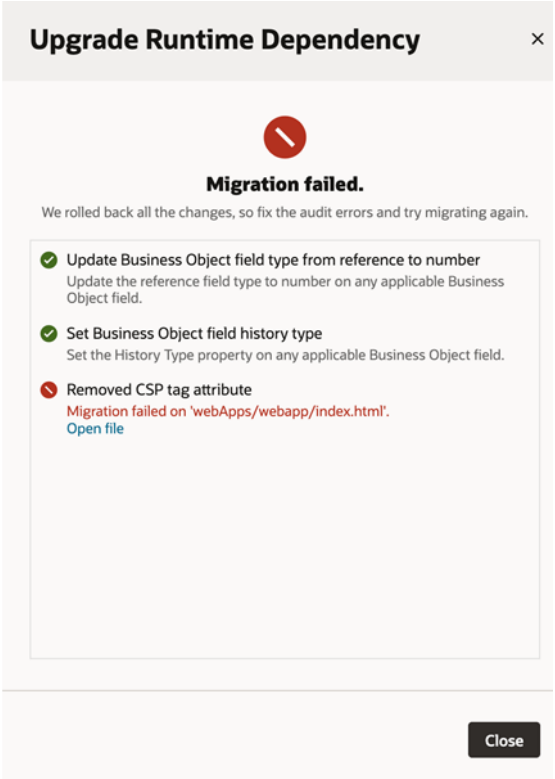


After Upgrading

Once you trigger an upgrade, Visual Builder makes changes to your app to better align it with the upgraded release. We may, for example, address deprecated properties or move things from one file to another.

You'll see details of all the changes made to your application during the migration, so you know exactly what happened behind the scenes. If your app has syntax errors and migration fails—conveniently displayed in the upgrade dialog—you'll need to fix those issues before you can upgrade. Here's an example of what you may see when a migration succeeds and when a migration requires your action:





Immediately after migration, if you decide you don't want the upgrade, you can close the Upgrade Runtime Dependency dialog, then use the **Undo** icon in the header to roll back all the changes, though upgrading is the recommended course of action.

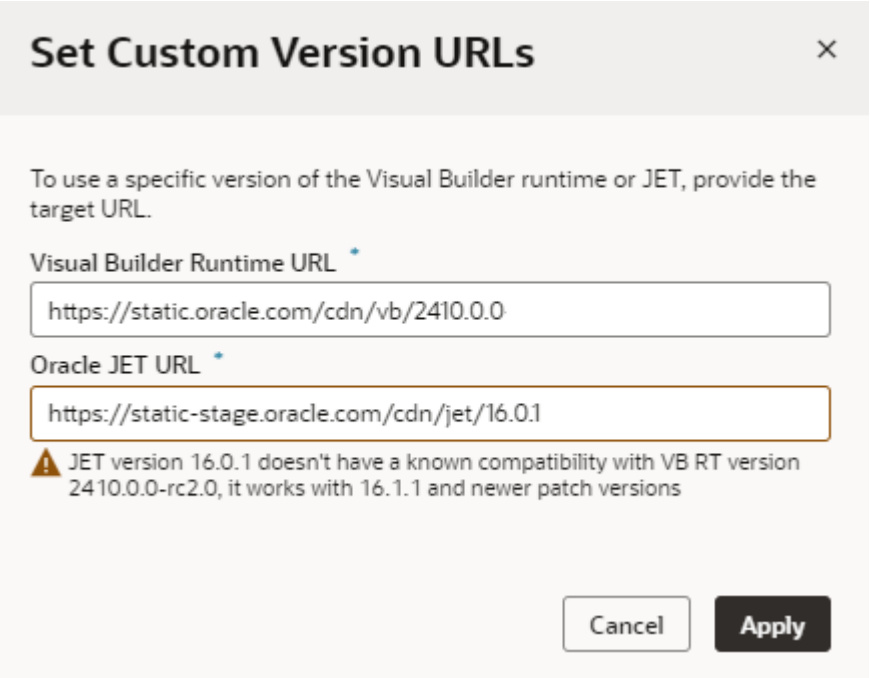
After you've upgraded your app, you still have to stage or publish the app for the upgrade to take effect for your running application.

> **Note:**
>
> If your app was last opened in the Designer before 19.4.3, the next time you open the app, it will be automatically upgraded to the latest Visual Builder Runtime/JET versions. The ability to control when to upgrade is available only to apps last opened in 19.4.3 or later.

Set a Custom Version

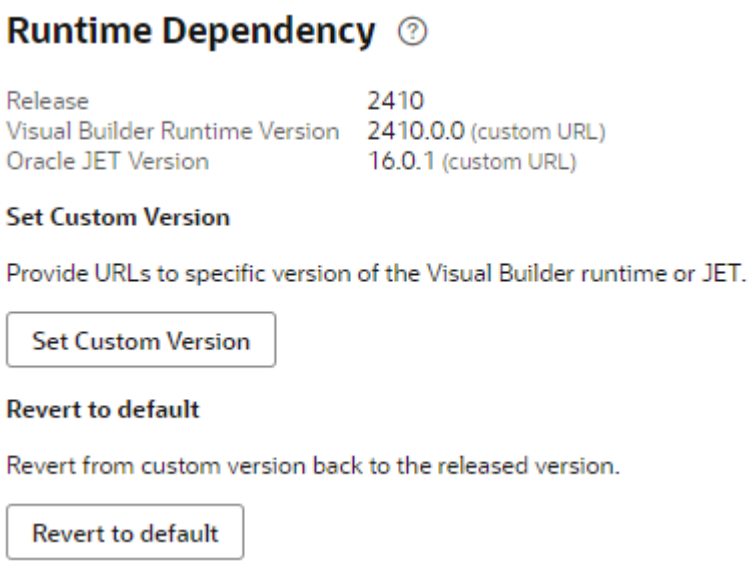
Use the **Set Custom Version** option only when directed to do so by Oracle. This option is provided in case you temporarily need to freeze your runtime dependencies for some reason, or point to a version of JET or the Visual Builder Runtime that may not yet be widely available.

The versions you specify must be compatible with each other; if they aren't, you'll see an error message, like this:



Although not recommended, you can ignore the warning message and click **Apply** to apply your changes.

The versions you specify in the **Set Custom Version URLs** dialog remain in effect until you set another custom version or use **Revert to default** to upgrade to the latest Visual Builder Runtime and JET versions:



Understand What's Happening in visual-application.json

When you use the UI to influence your upgrade preferences, Visual Builder makes the corresponding changes to your visual application's underlying `visual-application.json`

file. Although you don't have to change any values physically, it can be helpful to understand what's going on behind the scenes.

In this example:

```
{
    "vbcs.dt.version": "20240618-24.10.0",
    "dependencies": {
        "upgrade": "micro",
        "paths": {
            "jet": "https://static.oracle.com/cdn/jet/16.1.3",
            "telemetry": "https://static.oracle.com/cdn/trace/8.2.1",
            "oracleImageGallery": "https://static.oracle.com/cdn/fnd/gallery/
2410.0.2",
            "visualRuntime": "https://static.oracle.com/cdn/vb/2410.0.0"
        }
    },
    "source.version": "2410",
    "migration": {
    ...
    }
}
```

- The `upgrade` property is set to `micro`. This means that if Visual Builder releases a minor version of the software (with numbers like 24.10.1, 24.07.2, etc.), your application will be automatically upgraded to that version the next time you open your application in the Designer. (Incompatibilities, while rare, tend to occur when shifting to major releases, especially when changes to a major JET release are also involved.) The `micro` setting is the default for new applications.

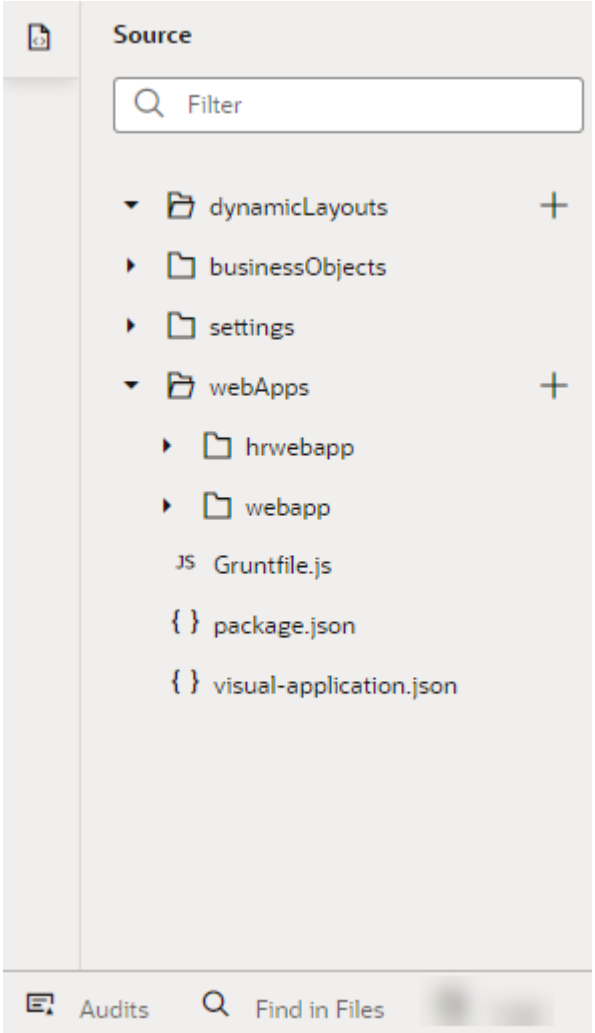
  If you are directed by Oracle to set custom versions for the Visual Builder Runtime and JET, the `upgrade` property is set to `none`, which means your app won't be upgraded to newer available minor versions, should they become available. In effect, your changes are frozen until you use the **Revert to Default** option, at which point the `upgrade` property is set back to `micro`.
- The version of JET (under `paths`) is `16.1.3`, while the Visual Builder Runtime version is `2410.0.0` (a minor version would have `2410.0.1`).

Resolve Upgrade Issues

When you access an application following an upgrade, Visual Builder tries to update the application's metadata for compatibility with the newer version. If this update fails, Visual Builder opens your application in **recovery mode** to let you manually resolve the issues and retry the upgrade.

In addition to server-side upgrade issues, recovery mode may be triggered if the JSON information in the application's configuration file (`visual-application.json`) cannot be parsed. This can happen when there's an unresolved merge conflict or the JSON syntax is invalid.

If your app launches in recovery mode (as shown here), you no longer have access to the Navigator and Design view in the Designer:



Use the subset of features (Source view, Code view, Audits, and Find in Files) that Visual Builder makes available to find and resolve the issues that prevent the upgrade. The file information in the notification can help you navigate to the file that needs to be fixed.

Once you resolve all issues, click **Upgrade** to restart and complete the upgrade. Click the button as often as required until all issues are resolved.

Importing an application exported from an older version than the target instance also triggers an application upgrade. If the upgrade fails, the app is imported with the issues, but will be placed in recovery mode to help you fix the issues. However, if the imported application zip file contained entity or setup data, this data will not be imported if the import triggers recovery mode. Instead, once the application has been upgraded, you'll need to import the data in the application zip using the Data Manager on the Business Objects page (see Work with the Data Manager).

What Happens During Software Maintenance?

Visual Builder will occasionally need to undergo planned maintenance to bring the software to the most current release.

Your service will not experience any downtime during the update. Once the update is complete, you'll be able to start using the new features immediately. While the update should not affect existing published applications, Oracle recommends that you update applications still in

development to the latest version of the Oracle JET and Oracle Visual Builder runtime libraries. Not only will this allow you to take advantage of the new features and bug fixes they offer, but you'll also be on a supported platform.

For published applications, we recommend that you create a new version of the application, update the runtime libraries in Application Settings, and test your application to make sure it functions properly. Once confirmed, publish the new version of the app. See Upgrade Your App for more information.

28

Optimize Your Builds and Audit Your Code Using Grunt

Visual Builder provides a number of NPM packages that you can use to audit, optimize, build, and deploy the web (and mobile) apps in your visual applications using Grunt.

Overview

You can use Grunt to build your application from the sources stored in your Git repository or stored locally. Your visual application includes a Grunt file that you modify to define custom tasks that you want to include in the build process and to configure built-in tasks.

The root folder of your visual application includes two resource files that Grunt uses when building applications:

File	Description
`Gruntfile.js`	Contains a basic Grunt script for building the app that can be modified to add custom build tasks, and to configure built-in tasks.
`package.json`	Declares the dependency and specifies the URL reference to the `grunt-vb-build` NPM package that provides the Grunt tasks used to build and audit visual applications. Visual Builder automatically updates the package version and URL whenever Oracle publishes a new version of the package.

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

Step	Description
Process the application sources	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 configuration into the application's `index.html`, processing other application templates, the creation of service definition files, and so on. You run the `vb-process-local` task to process application sources. This task creates an output directory (`./build/processed`) with built assets that can be consumed by the `vb-package` and `vb-deploy` tasks.
Optimize the processed sources	This step consists of a number of parts: optimize images, optimize styles, create require module bundles, and, if necessary, deploy the app's resources to a content delivery network. To optimize the processed sources, you run either the `vb-optimize` task on its own, or you run the `vb-package` task (which includes the `vb-optimize` and `vb-manifest` tasks by default). The `vb-optimize` task runs the `vb-css-minify`, `vb-require-bundle`, and `vb-optimize-cdn` tasks, and generates the `./build/optimized` directory with the optimized sources.
Deploy the application artifact	This step consists of creating the deployment package archive and deploying it to the Visual Builder instance. You run the `vb-deploy` task to deploy the application artifact.

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

Build Your Application Using Oracle Visual Builder Studio

You can configure a continuous integration and delivery pipeline in Oracle Visual Builder Studio to build and deploy your visual application to a Visual Builder environment.

When you create a workspace for a new or imported application, VB Studio automatically creates a pipeline for your visual application. This pipeline provides built-in Grunt-based package and deploy jobs that automate the task of continuous integration and delivery for you. You can configure these built-in jobs (or manually create jobs) to package and optimize your application, then deploy it to the Visual Builder instance defined in your project's environment. The high-level steps to do this are the same regardless of whether you deploy your visual application to a development, test, or production environment.

For an overview of how you set up a visual application project to build and deploy, see Set Up Visual Builder Studio for Visual Applications in *Administering Visual Builder Studio*. For specific steps, depending on where you are in the application development lifecycle (development/test or production), see:

- Create and Set Up a Project for Development
- Set Up the Project to Deploy for Production

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.

To build an application locally, you need to install Node.js and its package manager (npm) on your local system. You'll also need to save the sources for the visual application on your local system. You can get your visual application's sources by either:

- Cloning the Git repository containing the sources, or
- Exporting the visual application from Visual Builder and extracting it to your local system

Build and Deploy Your Application

Once you get your application sources, you need to run a number of tasks to build and deploy the application to the Visual Builder environment where you want to use the application.

A typical series of tasks that you run if you want to build and deploy your visual application to a Visual Builder instance is:

1. `vb-process-local`
2. `vb-package`
3. `vb-deploy`

To build and deploy your application locally:

1. Open a command-line interface and enter `node -v` to confirm that version 18.x or later of Node.js is installed and enter `npm -v` to confirm that NPM is installed.
2. In the command-line interface, navigate to the folder on your local system containing the `package.json` and `Gruntfile.js` files.
3. Enter `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`.

> **Tip:**
>
> If you add a `package-lock.json` file to your Git repository, you can avoid unexpected changes to NPM dependencies when you perform a build. When this file is present, the build pipeline needs to run the `npm ci` command, which uses the dependencies listed in the `package-lock.json` file, because the `npm install` command ignores the generated file.
> To take advantage of this functionality:
>
> a. Clone the Git repository locally.
>
> b. Run `npm install` to generate the `package-lock.json` file.
>
> c. Commit the generated file locally and then push it to the remote repository.
>
> Whenever the package.json file changes, you need to regenerate it the same way.
>
> Changes can occur when:
>
> - You upgrade the VB Runtime release in **Settings**.
> - Visual Builder automatically upgrades library dependencies to the latest available micro-release during a service upgrade.
> - You manually edit the generated file.

4. Enter the task names in the command-line interface to process your application sources, package the application, and deploy the application to the Visual Builder instance. The following example shows how you execute these tasks along with some supported parameters:

```
# First build application sources
./node_modules/.bin/grunt vb-process-local
    --url:ce=<URL of Component Exchange service>
    --username:ce=<username to access Component Exchange service>
    --password:ce=<password to access Component Exchange service>

# Package the application sources. This task in turn executes
# vb-optimize and vb-manifest
./node_modules/.bin/grunt vb-package


# Deploy and publish the visual application to the target Visual
# Builder instance using the data schema from the previously
# published live version of the application.
./node_modules/.bin/grunt vb-deploy \
    --url:rt=https://vbruntime-instance.example.com/ic/builder/ \
    --username:rt=$vbuser \
    --password:rt=$vbpass \
    --id=helloworld \
    --ver=0.1 \
    --remoteProjectId=learning_demo_28288 \
    --remoteGitRepo=https://vbstudio-instance.example.com/vbstudio-vboci/s/
learning_demo_28288/scm/learning_demo.git \
```

```
    --schema=live
    --publish
```

To view the full list of supported parameters for each task, see:

- vb-process-local
- vb-package
- vb-deploy

When these Grunt tasks finish, you can test your application by opening the URL of the deployed application in your browser. The command-line interface where you run these Grunt tasks displays the URLs that identify the deployment location of the web (or mobile) apps.

Authentication

Build tasks exposed by the `grunt-vb-build` package use OAuth access tokens to authenticate calls to Visual Builder instances. There are multiple ways to tell a Grunt task to obtain this access token:

- Explicitly pass an OAuth access token through the `--accessToken` option. To get an access token, access your application's Settings editor, then click Business Objects. Click Get Access Token and look for the Access Token Value. You can now use the access token's value in the build task for authentication, for example:

```
./node_modules/.bin/grunt vb-deploy \
    --url=https://vbruntime-instance.example.com/ic/builder/ \
    --
accesstoken=eyJ4NXQjUzI1NiI6IlVub283NXlZaUtqdXJ2Z3dDMW4tRHdqM1NWRFIwSGRwdVR
vd045c3NTWGsiLCJ4NXQiOiI5bEFQMTMxSWFuY1BQbFRQdnA3dWRZb2JkZFkiLCJraWQiOiJTSU
dOSU5HX0tFWSIsImFsZyI6I... \
    --id=helloworld \
    --ver=1.0
```

- Use Basic authentication for the Visual Builder instance through the `--username` and `--password` options, for example:

```
./node_modules/.bin/grunt vb-deploy \
    --url=https://vbruntime-instance.example.com/ic/builder/ \
    --username=mary.jane@example.com \
    '--password=***' \
    --id=helloworld \
    --ver=1.0
```

- Use Oracle Identity Cloud Service (IDCS) integration through the `--username`, `--password`, `--idcsClientId`, `--idcsClientSecret`, and `--idcsScope` options
This table describes the authentication options when you use IDCS to get the OAuth access token:

Name	Mandatory	Default Value	Description
url[:dt]	no	n/a	URL of the Visual Builder design-time service. The `:dt` suffix is optional.

Name	Mandatory	Default Value	Description
username[:dt]	no	n/a	The username used to obtain an OAuth access token for further communication with Visual Builder services. The `:dt` suffix is optional.
password[:dt]	no	n/a	The password used to obtain an OAuth access token for further communication with Visual Builder services. The `:dt` suffix is optional. A password specified via the `--password` Grunt option may need to be enclosed in single quotation marks (') if it contains special characters. In general, it's advisable to always use quotation marks for the `--password` option, especially in VB Studio jobs where the password is provided via a job variable. For example: `grunt vb-deploy '--password=Jkl@#&!%^23' ....`
idcsClientId	no	n/a	IDCS client ID (username)
idcsClientSecret	no	n/a	IDCS client secret (password)
idcsScope	no	n/a	IDCS scope

For example:

```
./node_modules/.bin/grunt grunt vb-deploy \
--url=https://vbdesigntime-instance.example.com/ic/builder/ \
--username=mary.jane@example.com \
'--password=***' \
--idcsUrl=https://idcs-
xxxxx14a21d74e72baf5a26c65900c12.identity.dev.ocqadev.com \
--idcsClientId=5b4e306598374529959bc8e18c91a86e \
'--idcsClientSecret=****'\'
--idcsScope=https://
AB2C5D66E78F4GHI83J9304219KL8939.integration.test.ocp.oc-
test.com:443urn:opc:resource:consumer::all'\
--id=test \
--ver=1.0
```

Grunt Tasks to Build Your Visual Application

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.

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-clean

Cleans the build directory.

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

Detail	Description
subtasks	n/a
multitask	n/a
hooks	`vb-pre-clean`, `vb-post-clean`
input	`build/*`
output	n/a

The following table describes the sole build option for the `vb-clean` task:

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.

vb-process-local

Processes application sources locally in the Grunt process.

The processing operation resolves various templates and further modifies (adds and updates) several application resources.

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. If you provide both credentials and an OAuth access token, the task authenticates using the access token.

The following table describes the subtasks, hooks, and inputs and outputs of the `vb-process-local` task:

Detail	Description
subtasks	n/a
multitask	n/a
hooks	n/a
input	`${gitSources}`
output	`build/processed/*`

The following tables describe the options for the `vb-process-local` task. The `:ce` suffix is required only when your application includes Web Components and, as a result, references the Component Exchange.

Authentication options

Name	Mandatory	Default Value	Description
url[:ce]	no	n/a	URL of the Component Exchange service. Not needed if the application does not reference Web Components. Talk to your administrator to get the URL of the Component Exchange used by your Visual Builder instance, available on the Tenant Settings page. For the URLs of publicly available Component Exchanges, see Manage Your Component Exchange in *Administering Oracle Visual Builder in Oracle Integration*.
username[:ce]	no	n/a	The username to access the Component Exchange service. The `:ce` suffix is optional.

Name	Mandatory	Default Value	Description
password[:ce]	no	n/a	The password to access the Component Exchange service. The `:ce` suffix is optional. A password specified via the `--password` Grunt option may need to be enclosed in single quotation marks (') if it contains special characters. In general, it's advisable to always use quotation marks for the `--password` option, especially in VB Studio jobs where the password is provided via a job variable. For example: `grunt vb-deploy '--password=Jkl@#&!%^23' ....` Here's an example of the `vb-process-local` task with the `--username` and `--password` options when your app is using components from the publicly available Component Exchange: `grunt vb-process-local --url:ce=https://devinstance4wd8us2-wd4devcs8us2.uscom-central-1.oraclecloud.com/profile/devinstance4wd8us2-wd4devcs8us2/s/devinstance4wd8us2-wd4devcs8us2_compcatalog_3461/compcatalog/0.2.0 --username:ce=comp.catalog --password:ce=bXwphh6RMFjn#g`

Name	Mandatory	Default Value	Description
accessToken[:ce]	no	n/a	The value of an OAuth access token to access the Component Exchange service. If provided, username and password options are not necessary. The `:ce` suffix is optional.

Build options

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.
git-sources	no	./	Location of the visual application's sources.

Other options

Name	Mandatory	Default Value	Description
mode	no	default	Defines the build mode: • default: Builds a visual application's assets for deployment to Visual Builder runtime service • fa: Builds a visual application's assets for deployment to an Oracle Cloud Applications environment
fa-indexHtml-resolveVariables	no	true	When `mode=fa` and if this property is set to `false`, the generated `index.html` does not resolve JET and VB URL and version template variables. The file will contain %JET_CDN_PATH%, etc.

Name	Mandatory	Default Value	Description

fa-inject-pwa-tokens	no	false	When `mode=fa`, the build injects the following tokens into `index.html`
			
			`<!-%APPLE_ICONS_LINKS%>`
			`<!%SPLASHSCREENS_LINKS%->`
			
			Oracle Cloud Applications servlet should resolve these tokens in the runtime.

enableTelemetry	no	false	Injects the Trace.ConsoleProfile initializer into the generated `index.html`

You can better control the template-resolving process for the web (or mobile) application's `index.html` using the `fa` build mode rather than the `default` build mode. Resolving the following templating marks:

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

can be controlled by the following template variables, defined in the configuration of the `vb-process-local` task:

- %BASE_URL%
- %JET_CDN_PATH%
- %JET_CDN_VERSION%
- %VB_CDN_PATH%
- %VB_VERSION%
- %JET_UI_STYLE_PATH%

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 an 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. The exception is the %BASE_URL% variable, which doesn't have a default and

the template variable reference will be kept in the resulting `index.html` until the value is provided explicitly in the **resolveTemplate** configuration object, as in the following example:

```
grunt.initConfig({
  "vb-process-local": {
    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.

Setup to Resolve Environment Variables

The `vb-process-local` task creates static application assets that contain various environment variables, which are resolved when application assets are deployed to a Visual Builder runtime instance by the `vb-deploy` task. If you don't deploy the application to a Visual Builder instance, you may want to resolve these variables during the build. This may be handy, for example, for local application development which in combination with the vb-serve task allows you to quickly test your application locally.

You can define environment variable values by adding the following configuration to your `Gruntfile.js` file (you can also add the configuration into the object you pass to the `grunt.initConfig()` method call). Here's an example `Gruntfile.js` with the `vb-process-local` environment configuration:

```
grunt.config('vb-process-local.environment', {
       "env.userProfileUrl": "_currentuser",
       "env.vbServer.url": "http://127.0.0.1:3000/",
       "env.vbServer.context": "",
       "env.application.id": "myApp",
       "env.application.version": 1618840321144,
       "env.profileId": "base_configuration",
        });
```

The following table describes the list of environment variables used during the Grunt build process:

Name	Description
app-flow.json	
env.profileId	Selected application profile
env.userProfileUrl	User profile URL
env.oauthUserProfileUrl	OAuth user profile URL
env.idcsInfo	IDCS information
catalog.json	
env.catalogJson	Entire contents of the catalog.json file
index.html	
env.vbServer.url	URL of the Visual Builder runtime server

Name	Description
env.vbServer.context	Context path of the Visual Builder runtime server
env.application.version	Application version

vb-deploy

This task deploys a visual application to a Visual Builder runtime instance.

You must build visual applications using the `vb-process-local` task before you use the `vb-deploy` task.

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

Detail	Description
subtasks	n/a
multitask	n/a
hooks	n/a
input	`build/processed/*`, `build/optimized/*`
output	`build/deploy.zip`

The following tables describe the options for a visual application when using the `vb-deploy` task.

Application selection task options for visual applications

Name	Mandatory	Default Value	Description
id	yes	Read from the `rootURL` attribute in `visual-application.json`	ID of the visual application you're going to build.
ver	yes	Read from `version` attribute in visual-application.json	Version of the visual application.
url[:rt]	yes	n/a	A Visual Builder runtime service URL. The `rt` suffix is optional.
profileId	no	n/a	Application profile ID.
remoteProjectId	no	n/a	ID of the source project in VB Studio.
remoteGitRepo	no	n/a	URL of the source Git repository in VB Studio.

Visual application authentication options

Name	Mandatory	Default Value	Description
username[:rt]	no	n/a	The username to be used to obtain OAuth access token for further communication with Visual Builder runtime. The `rt` suffix is optional.
password[:rt]	no	n/a	The password to be used to obtain OAuth access token for further communication with Visual Builder runtime. The `rt` suffix is optional. A password specified via the `--password` Grunt option may need to be enclosed in single quotation marks (') if it contains special characters. In general, it's advisable to always use quotation marks for the `--password` option, especially in VB Studio jobs where the password is provided via a job variable. For example: `grunt vb-deploy '--password=Jkl@#&!%^23' ...`.
accessToken[:rt]	no	n/a	The value of OAuth access token. If provided, username and password options are not necessary. The `rt` suffix is optional.
sslCertificate[:rt]	no	n/a	The path to the SSL certificate for the connection to Visual Builder instances provisioned with self-signed certificates. The `rt` suffix is optional.

Visual application build options

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.

Visual application data processing options

Name	Mandatory	Default Value	Description
schema	no	new	Specifies data schema processing during application stage or publish. The value can be: • `new` to create a new data schema • `dev` to use the data schema from development • `stage` to use the data schema from the previous staged version of the application • `live` to use the data schema from the previous live version of the application

Other options for visual applications

Name	Mandatory	Default Value	Description
publish	no	false	Defines whether the deployed application should be published or not.

vb-optimize-cdn

Deploys the app's resources to a Content Delivery Network (CDN) service to improve a web (or mobile) app's loading performance.

Typically, the application's `index.html` resource that's mapped to the root application URL is deployed to a Visual Builder Runtime service, and the rest of the application's assets is uploaded to CDN. To link application assets deployed to the CDN with the `index.html`, you need to use a `base` tag, whose `href` value determines the root URL that's used to resolve relative references in `index.html` (`scripts/images/styles/modules`).

This task inserts this `base` tag with the `href` attribute that contains the base URL of the deployed application resources. The URL inserted in the `href` is taken from the following locations in this order:

- `cdnUrl` command-line option
- `cdnUrl` setting in the `vb-optimize-cdn` task's options
- `resolveTemplate.BASE_URL` entry from the `vb-process-raw-index-html` task's options (for Oracle Cloud Applications)

If you don't specify any of these options, the task will silently do nothing.

This task automatically runs from within the `vb-optimize` task for visual applications. It does not run for Oracle Cloud Application extensions.

Detail	Description
subtasks	-
multitask	(config generator task: vb-optimize-cdn-configuration)
hooks	-
input	`build/optimized/*`
output	`build/optimized/*`

The following table describes the options for the `vb-optimize-cdn` task:

Name	Mandatory	Default Value	Description
cdnUrl	n/a	n/a	URL of the deployed application assets. It's important that the CDN assets URL ends with slash ("`https://my.cdn.com/myapp/`"), otherwise resources will be resolved against the parent segment URL.
insertBaseUrl	n/a	false	Inserts BASE_URL token to `index.html`'s `vbInitConfig` section
insertBaseTag	n/a	true	Disables insertion of the `base` tag for Oracle Cloud Applications that define `resolveTemplate.BASE_URL` entry in the `vb-process-local` task's options; in this case, insertion of `base` tag is automatic.

vb-optimize

Optimizes application sources by minifying images and CSS, and creating minified `requirejs` module bundles.

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

Detail	Description
subtasks	`vb-prepare`, `vb-image-minify`, `vb-css-minify`, `vb-require-bundle` , `vb-optimize-cdn`
multitask	n/a
hooks	`vb-pre-optimize`, `vb-post-optimize`
input	`build/processed/*`
output	`build/optimized/*`

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

vb-prepare

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 the rawSourcesMode option is not 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, set by default as "./") 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:

Detail	Description
subtasks	vb-process-raw
hooks	n/a
input	`build/processed/*` or `${gitSources}`
output	`build/optimized/*`

The following table describes the options for the `vb-prepare` task:

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.
rawSourcesMode	no	false	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 the `build/processed` directory (true if the directory exists).

This task can be used on its own if you're fine tuning the application's require bundles optimization and you don't want to run the `vb-optimize` task every time you change settings. In this case, calling `vb-prepare` before `vb-require-bundle` will prepare fresh processed application sources for optimization.

vb-prerender-cache-warm

Warms the prerender server's cache to improve response times for requests made by Googlebot, the main web crawler used for Google search, when indexing your web application's pages.

When a request for a Visual Builder page is received from Googlebot (or any other search engine), the request is routed through the prerender server embedded in Visual Builder. The prerender server loads the page and runs any JavaScript required to fully render that page before returning the page to Googlebot. This way, Googlebot always receives a fully rendered page for indexing.

Once the prerender server renders a URL, the page is cached, so markup can be returned immediately for subsequent requests for the same URL. If markup is not yet cached, the process of rendering the page can take some time. Typically, Googlebot waits until this process completes—but it is recommended that you warm the prerender server's cache for URLs in your application's sitemap, so Googlebot always gets the fastest possible response when requesting those URLs.

Use the `vb-prerender-cache-warm` task to warm up the prerender server's cache after your application (or a new version of it) is deployed.

> **Note:**
>
> Before warming the cache, ensure that you have an up-to-date `sitemap.xml` file as part of the application's sources. See Add a Sitemap to a Web App's Resources.

The following table describes the subtasks, hooks, and input and output of the `vb-prerender-cache-warm` task:

Detail	Description
subtasks	n/a
multitask	n/a
hooks	n/a
input	`path_to_web_app's_sitemap_file`
output	`Prerender server cache populated`

The following table describes the options for the `vb-prerender-cache-warm` task:

Name	Mandatory	Default Value	Description
id	yes	n/a	Web application's ID, for example, `MyApplication`.
ver	yes	n/a	Web application version, for example, `1.0`.
url[:dt]	yes	n/a	URL of the Visual Builder design-time service, for example, `https://my.visualbuilder.com/ic/builder`. The `:dt` suffix is optional.
sitemap	no	sitemap.xml	Path to the `sitemap.xml` file on the file system, for example, `webApps/myWebApp/sitemap.xml`.
manifest	no	manifest.json	Path to the `manifest.json`. This file tracks what has been prerendered successfully, so that the process can be re-run if necessary. For example, if entries change in `sitemap.xml`, the process will determine which entries have changed and warm those alone. This file also stores details on any URLs that were not successfully retrieved so they can be retried.
userAgent	no	Default Chrome user agent	User agent to use when requesting the page.
clearCache	no	false	Set this flag to `true` to clear the cache before starting to warming it. This flag is recommended when a new version of the application is deployed because any existing cached pages will contain broken asset links.

See Warm the Cache for URLs in a Sitemap for examples of how you can use this task.

The `vb-prerender-cache-warm` task uses the same authentication options as other Grunt tasks that connect to the server:

Name	Mandatory	Default Value	Description
username	no	n/a	The username to be used to obtain OAuth access token for further communication with Visual Builder runtime.

Name	Mandatory	Default Value	Description
password	no	n/a	The password to be used to obtain OAuth access token for further communication with Visual Builder runtime. A password specified via the `--password` Grunt option may need to be enclosed in single quotation marks (') if it contains special characters. In general, it's advisable to always use quotation marks for the `--password` option, especially in VB Studio jobs where the password is provided via a job variable. For example: `grunt vb-deploy '--password=Jkl@#&!%^23' ...`.
accessToken	no	n/a	The value of the OAuth access token. If provided, username and password options are not necessary.
sslCertificate	no	n/a	The path to the SSL certificate for the connection to Visual Builder instances provisioned with self-signed certificates.

vb-test

Runs the action chain tests that you have defined in your visual application for web (and mobile) apps.

The tests runs on built application sources, with the test results stored in the `build/tests/results` directory as follows:

- jUnit tests: `[web|mobile]Apps/<appId>/<browserId>test-results.xml`
- Tests coverage report: `[web|mobile]Apps/<appId>/vbCoverage.json`

For more information, see Test Action Chains Using the `vb-test` Grunt Task .

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

Detail	Description
subtasks	n/a
multitask	config generator task: `_vb-test-generate-configuration`
hooks	n/a

Detail	Description
input	`build/processed`
output	`build/tests/results`

The following table describes the options for the `vb-test` task:

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.
karma-debug	no	false	Enables unit test debugging. When Chrome open, you'll need to manually click **DEBUG** button in the window for the tests to run. You can open the Chrome Dev Tools, set some breakpoints, and reload the page to re-run the tests.
karma-log-level	no	INFO	Karma log level, which you can set to DEBUG for verbose logging.
karma-browser	no	ChromeHeadless	Browser mode to use. Use "Chrome" to run tests in UI (window) mode.
mocha-timeout	no		Timeout of Mocha tests (in milliseconds).

vb-require-bundle

Creates minified `requirejs` module bundles.

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

Detail	Description
subtasks	n/a
multitask	config generator task: vb-require-bundle-configuration
hooks	n/a
input	`build/optimized/*`
output	`build/optimized/*`

The following tables describe the options for the `vb-require-bundle` task.

Build option

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.

Optimization options

Name	Mandatory	Default Value	Description
emptyPaths	no	n/a	Comma-separated list of require paths that are set "empty". Requirejs optimizer will not follow and bundle matching dependencies.

Name	Mandatory	Default Value	Description
requirePaths	no	n/a	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: (`--requirePaths='{ "foo": "boo" }'`).
bundles	no		Defines custom require module bundles. Configuration schema: • `<bundle name>` – `modules` * `find` * `ids` – `exclude` * `find` * `ids`
bundles.modules	no		Specification of the modules that are to be added to the enclosing bundle element. Configuration schema: • `find` • `ids`
bundles.exclude	no		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. Configuration schema: • `find` • `ids`
bundles.[exclude|modules].find	no		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.
bundles.[exclude|modules].ids	no		Specifies list of module IDs.
transpile	no	false	Determines whether a separate set of require module bundles transpiled to ES5 code using babel preset-env preset should be created and stored in the `bundles/es5` directory. When this option is set to true, the 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.
minify	no	true	When set to true, babel minify preset is used to minify generated `requirejs` module bundles. The minification is also applied to the ES5 variants of the bundles if created.
optimize	no		**Deprecated. You should use the `minify` option instead.**
include	no	n/a	**Deprecated. You should use the `bundles` option instead.**

Name	Mandatory	Default Value	Description
exclude	no	n/a	**Deprecated. You should use the `bundles` option instead.**

vb-require-bundle-clean

Deletes resources that are part of the generated `requirejs` resource bundles.

When a web (or mobile) application is optimized using the `vb-require-bundle` task, the bundled resources stay in the `build/optimized` directory. You can use the `vb-require-bundle-clean` task to clean these bundled files before deployment. The files should not be needed in the deployment as resources should be loaded from the resources bundle.

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

Detail	Description
subtasks	n/a
multitask	n/a
hooks	n/a
input	`build/optimized/*`
output	`build/optimized/*`

The following table describes the build option for the `vb-require-bundle-clean` task:

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.

vb-css-minify

Minifies CSS resources.

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

Detail	Description
subtasks	n/a
multitask	config generator task: vb-css-minify-configuration
hooks	n/a
input	`build/optimized/*`
output	`build/optimized/*`

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

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.

vb-image-minify

Minifies the image resources.

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

Detail	Description
subtasks	n/a
multitask	config generator task: vb-image-minify-configuration
hooks	n/a

The following table describes the options for the `vb-image-minify` task:

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.

vb-json-minify

Minifies JSON resources to remove white spaces.

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

Detail	Description
subtasks	n/a
multitask	config generator task: vb-json-minify-configuration
hooks	n/a
input	`build/optimized/*`
output	`build/optimized/*`

vb-export

Downloads application sources from the Visual Builder instance and expands the archive on the local file system for further processing.

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

Detail	Description
subtasks	n/a
multitask	n/a
hooks	n/a
input	`build/optimized/*`
output	`build/optimized/*`

The following tables describe the options for the `vb-export` task.

Application selection options

Name	Mandatory	Default Value	Description
id	yes	n/a	ID of the visual application you're going to build. The application needs to exist on the referred Visual Builder instance.
ver	yes	n/a	Version of the visual application
url[:dt]	yes	n/a	URL of your Visual Builder service instance. The `dt` suffix is optional.

Authentication options

Name	Mandatory	Default Value	Description
username[:dt]	no	n/a	The username to be used to obtain OAuth access token for further communication with Visual Builder services. The `dt` suffix is optional.
password[:dt]	no	n/a	The password to be used to obtain OAuth access token for further communication with Visual Builder services. The `dt` suffix is optional. A password specified via the `--password` Grunt option may need to be enclosed in single quotation marks (') if it contains special characters. In general, it's advisable to always use quotation marks for the `--password` option, especially in VB Studio jobs where the password is provided via a job variable. For example: `grunt vb-deploy '--password=Jkl@#&!%^23' ...`.
accessToken[:dt]	no	n/a	The value of an OAuth access token. If provided, username and password options are not necessary. The `dt` suffix is optional.
sslCertificate[:dt]	no	n/a	The path to the SSL certificate for the connection to Visual Builder instances provisioned with self-signed certificates. The `dt` suffix is optional.

Build options

Name	Mandatory	Default Value	Description
git-sources	no	./	The location of the sources of the visual application.

vb-manifest

Creates build manifests for the visual application's build assets. The assets can be non-optimized (in the `build/processed` directory) or optimized (in the `build/optimized` directory).

> **Note:**
>
> This task is a multitask so it requires an existing Grunt configuration to run. You can create your own in your `Gruntfile.js` or use the `vb-manifest-configuration` subtask to generate one.

For visual applications, it creates two kinds of manifests:

- A manifest in each web (or mobile) application. This manifest contains a list of application resources, `requirejs` bundles mapping, and the name of the version_<hash> directory. The manifest location is `build/[processed|optimized]/[web|mobile]Apps/<applicationName>/build-info.json`.
- A manifest for the visual application itself, which contains the URL of the sources Git repository. The manifest location is `build/[processed|optimized]/build-info.json`.

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

Detail	Description
subtasks	n/a
multitask	config generator task: `vb-manifest-configuration`
hooks	n/a
input	`build/[processed|optimized]/*`
output	`build/[processed|optimized]/*`

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

Name	Mandatory	Default Value	Description
target	no	build	Name of build directory.
vx-version	no	n/a	Overwrites 'version' key in a visual extension's manifest with the given value. This option is only used for application extensions.
git-repository-url	no	The Git repository URL read from `${gitSources}/.git/config`.	URL of the application's source repository. The value is written into the generated build manifest of the visual application (`build/optimized/build-info.json`) under the git-repository-url key.

vb-package

Packages the visual application's sources. It includes application optimization (`vb-optimize`) and build manifest generation (`vb-manifest`).

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

Detail	Description
subtasks	`vb-optimize`, `vb-manifest-configuration`, `vb-manifest`
multitask	n/a
hooks	`vb-pre-package`, `vb-post-package`
input	`build/processed/*`
output	`build/optimized/*`

The following table describes the options for the `vb-package` task:

Name	Mandatory	Default Value	Description
`skip-optimize`	No	`false`	If `true`, the `vb-package` task will not call the `vb-optimize` task.
`clean-bundled-resources`	No	`false`	If `true`, will run the `vb-require-bundle-clean` task after `vb-optimize` and before `vb-manifest`.

vb-archive

Creates ZIP archives for various application artifacts:

1. `build/sources.zip` with application sources
2. `build/processed.zip` with the contents of the `build/processed` directory; this directory is created either by the `vb-process` or `vb-process-local` tasks.
3. `build/optimized.zip` with the contents of the directory; this directory is created by the `vb-optimize` task.

If any of the directories don't exist, the task will ignore it, instead of failing. If the task target is used with any of these values: `sources`, `processed`, `optimize`, for example:

```
grunt vb-archive:optimized
```

the `vb-archive` task creates just the corresponding ZIP archive. If no task target is specified, all three ZIP archives are created.

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

Detail	Description
subtasks	n/a
multitask	n/a
hooks	n/a
input	`${gitSources}`, `build/processed/*`, `build/optimized/*`
output	`build/sources.zip`, `build/processed.zip`, `build/optimized.zip`

The following table describes the options for the `vb-archive` task:

Name	Mandatory	Default Value	Description
sources-zip-path	no	n/a	Custom path to the source archive. The path needs to include the archive name as well as the extension, for example: `--sources-zip-archive=dist/myAppSources.zip`
processed-zip-path	no	n/a	Custom path to the processed assets archive. The path needs to include the archive name as well as the extension.
optimized-zip-path	no	n/a	Custom path to the optimized assets archive. The path needs to include the archive name as well as the extension.

vb-process-raw

This task has been deprecated. Use the `vb-process-local --mode=fa` task instead.

For more information, see vb-process-local.

vb-process-raw-index-html

This task has been deprecated. Use the `vb-process-local --mode=fa` task instead.

For more information, see vb-process-local.

vb-application

Performs an operation with a remote visual application (in other words, an application that exists on a Visual Builder instance).

The following table describes the only supported operation for the `vb-application` task:

Detail	Description
delete	Deletes a remote application from the Visual Builder instance, for example: `grunt vb-application:delete --id=myApp --ver=1.0 --delete-application --url=https://myvb.oracle.com --username=me@oracle.com --password=foo`

The following tables describe the options for the `vb-application` task:

Application selection options

Name	Mandatory	Default Value	Description
id	yes	n/a	ID of the visual application you're going to build.
ver	yes	n/a	Version of the visual application.
url[:dt]	yes	n/a	Your Visual Builder instance URL. The `dt` suffix is optional.

Authentication options

Name	Mandatory	Default Value	Description
username[:dt]	no	n/a	The username to be used to obtain OAuth access token for further communication with Visual Builder services. The `dt` suffix is optional.
password[:dt]	no	n/a	The password to be used to obtain OAuth access token for further communication with Visual Builder services. The `dt` suffix is optional. A password specified via the `--password` Grunt option may need to be enclosed in single quotation marks (') if it contains special characters. In general, it's advisable to always use quotation marks for the `--password` option, especially in VB Studio jobs where the password is provided via a job variable. For example: `grunt vb-deploy '--password=Jkl@#&!%^23' ...`.
accessToken[:dt]	no	n/a	The value of an OAuth access token. If provided, username and password options are not necessary. The `dt` suffix is optional.
sslCertificate[:dt]	no	n/a	The path to the SSL certificate for the connection to Visual Builder instances provisioned with self-signed certificates. The `dt` suffix is optional.

Other options

Name	Mandatory	Default Value	Description
delete-application	yes	n/a	You need to add `--delete-application` to be able to run `vb-application:delete` task. This is required as confirmation of your intention to delete the application.

vb-serve

Serves static application assets on a local web server started during the Grunt build process (`vb-process-local` and `vb-deploy` tasks). The served static assets root is either in the `build/processed` directory for a processed (non-optimized) application or in the `build/optimized` directory for an optimized application.

> **Note:**
>
> Application assets produced by `vb-process-local` are environment agnostic and contain several unresolved environment references, which are resolved when application assets are deployed to a Visual Builder runtime instance by `vb-deploy`. See Setup to Resolve Environment Variables for details.

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

Detail	Description
subtasks	n/a
multitask	no
hooks	n/a
input	`build/[processed | optimized]/*`
output	n/a

The following table describes the task target of the `vb-serve` task:

Detail	Mandatory	Description
application_path	yes	Path to the served web (or mobile) application (for example, `webApps/myApp`). .

The following table describes the options for the `vb-deploy` task:

Name	Mandatory	Default Value	Description
application	no	First web (or mobile) application found	Path to the application to be served. Similar to passing the application path via task target, for example, `webApps/myApp`.
boundPortFile	yes	n/a	Path to the file where `vb-serve` writes the bound port number. This is useful if you don't specify the port and a free system port is used, but you need an automated system to connect to the served application.
open	no	false	If `true`, a new browser window will be opened for the served URL.
port	no	0 (binds to a free port)	Server port

The options can be set either as command-line options or via the Grunt configuration object. Here's an example from the command line:

```
# First build the application sources
./node_modules/.bin/grunt vb-process-local
# serve the assets at port 8888 and open the application URL in browser
./node_modules/.bin/grunt vb-serve:webApps/myApp --port=8888 --open
```

The `vb-serve` task does not finish until you stop it explicitly. You can also stop the task (and the server) by sending a GET request to `http://localhost:<port>/stop`.

vb-pwa

Injects a workbox pre-cache manifest of PWA-enabled web applications into the service worker script. (See the Chrome developer's workbox API doc for details.)

Manifest entries:

- application assets (`build/processed|optimized/webApps/<appId>/**`)
- `services/catalog.json` (this is created during application deployment. It is not generated by the build)
- JET dependencies
- JET persistence toolkit dependencies
- Visual Runtime dependencies
- Telemetry
- Workbox dependencies
- Imported stylesheets and their dependencies (fonts, images)

User may customize the behavior by adding a custom workbox config file into application sources root. The format of the config can be either a JSON file or JavaScript module:

- `vb-workbox-config.json`
- `vb-workbox-config.js`

There are two options when using a JavaScript module. The module can either export an object as the config, or it can expose a function that returns a config object.

In the case of exporting an object, the following context is passed into it as an argument and can be used to generate the config object:

```
{
  sources: {
    root, // path to the root of GIT sources
    application, // path to the web application in GIT sources
  },
  builtAssets: {
    root, // path to root of the build application assets (build/processed|
optimized)
    application, // path to application built assets build/processed|
optimized/webApps/foo
  },
  jetUrl, // JET URL
```

```
  visualBuilderUrl, // VB URL
}
```

The following example shows how to override the workbox config generated by the build, so the pre-cache manifest is injected into a custom file (`precacheManifest.json`). Please note the file has to exist in the sources. The custom config also adds a custom manifest entry for JET's `oj-redwood-min.css` stylesheet.

Example of custom workbox config

```
module.exports = (ctx) => {
  return {
    "swSrc": `${ctx.sources.root}/precacheManifest.js`,
    "swDest": `${ctx.builtAssets.application}/precacheManifest.js`,
    "additionalManifestEntries": [
      {
        "url": `${ctx.jetUrl}/default/css/redwood/oj-redwood-min.css`,
        "revision": null
      }
    ]
  }
}
```

The complete list of workbox configuration options is here: https://developers.google.com/web/tools/workbox/reference-docs/latest/module-workbox-build

Details

Detail	Description
subtasks	n/a
multitasks	n/a
hooks	n/a
input	`build/processed|optimized`
output	`build/processed|optimized`

Build Options

Name	Mandatory	Default Value	Description
target	n/a	build	Name of build directory
git-sources	n/a	check	Location of sources of the visual application

vb-fa-generate-base-app-config

Generates an Oracle Cloud Application extension's base application descriptor (`private/cache/base-application-config.json`) based on the URL of a given Oracle Cloud environment.

Options

Name	Mandatory	Default Value	Description
fa-url	Yes	n/a	URL of Oracle Cloud environment (for example, `https://adfcdr03.fa.dc1.dev2.example.com`).

vb-pwa-splashscreen

Creates PNG splash screens and a `links.html` file from the given SVG file. This task is typically used to overwrite the default Visual Builder splash screens with a custom splash screen.

The resources are generated into the **sources** of the specified visual web (or mobile) application, specifically into the `resources/splashscreens` directory.

The generated resources:

- a set of PNG files for each iOS device screen size
- a `links.html` file. This file is a template that's used to inject links to the splash screen into the `index.html` file of the built web application

You provide the path to the specific application where you want the assets generated as the task target (in the following example, `webApps/myApp` is the path to the application):

```
grunt vb-pwa-splashscreens:webApps/myApp --svg=mysplash.svg
```

It's possible to configure the underlying package by specifying `pwa-asset-generator` properties, for example, to specify a background color (see https://www.npmjs.com/package/pwa-asset-generator):

```
grunt vb-pwa-splashscreens:webApps/myApp --svg=mysplash.svg --pwa-asset-generator:background=yellow
```

Details

Detail	Description
subtasks	n/a
multitask	n/a
hooks	n/a
input	path to SVG file
output	`${sources}web|mobileApps/<appid>`

Build Options

Name	Mandatory	Default Value	Description
target	n/a	build	Name of build directory
git-sources	n/a	check	Location of sources of the visual application
svg	Yes	n/a	Path to SVG file

Name	Mandatory	Default Value	Description
pwa-asset-generator	n/a	n/a	Properties to pass to pwa-asset-generator config For a list of possible options, see: https://www.npmjs.com/package/pwa-asset-generator

vb-watch

Watches visual application or extension sources and re-runs build on changes. Initial build is performed.

Tasks that are involved in the build can be defined by the task target, for example:

```
# this will just rebuild the project, i.e. runs vb-clean vb-process-local
grunt vb-watch:rebuild

# this will also repackage (vb-clean vb-process-local vb-package)
grunt vb-watch:repackage

# this will also deploy (vb-clean vb-process-local vb-package vb-deploy)
grunt vb-watch:redeploy

# or without target ¿ defaults to repackage
grunt vb-watch
```

There's also an extra option **--watch** that can be added to any Grunt command. The behaviour is identical to `vb-watch`, but will run your custom tasks instead of the predefined sets.

```
grunt vb-clean vb-process-local vb-test --watch
```

Details

Detail	Description
subtask	n/a
multitask	n/a
hooks	`vb-watch-pre-run` `vb-watch-post-run`

Build Options

Name	Mandatory	Default Value	Description
target	n/a	build	Name of build directory
git-sources	n.a	`./`	Location of sources of the visual application.

Customize Your Grunt Build Process

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

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:

```
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-deploy` task. To redefine the task `vb-deploy`, your edited `Gruntfile.js` might be similar to the following:

```
module.exports = (grunt) => {
    require('load-grunt-tasks')(grunt);
    grunt.registerTask('vb-deploy', () => {
        // 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.

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

Host an Application on a Content Delivery Network (CDN)

You can host an application on a CDN to improve response times for clients that connect to your visual application.

Before you can publish your app, you need to stage it and export it your local machine. Once you have exported it to your local machine, build it using the following command from the root directory where you extracted the exported application:

```
./node_modules/.bin/grunt vb-build \
    --url=<url of visual app instance> \
    --username=<username> \
    --password=<password> \
    --id=<your visual app ID> \
    --ver=<your visual app version> \
    --cdnURL=<url of the deployed application's assets> \
    --insertBaseUrl
```

> **Note:**
>
> The URL for the `cdnURL` option must end with `/`. For example, `https://hostname:port/CDN/webApp/CDNLocation/`. If you omit the final `/`, the resources will be resolved against the parent segment of the URL.

The `insertBaseURL` task option inserts a `BASE_URL` token into `index.html`'s `vbInitConfig` section.

Copy `build/optimized.zip` that the `grunt vb-build` task generates to the CDN host and extract it to the directory location where the CDN will host it.

For information about using the `grunt vb-build` task, see Build Your Application Locally.

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.

```
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.

```
{
  "vb-require-bundle": {
     "webapp1": {
         "options": {
             "transpile": true,
             "minify": true,
             "bundles": {
                 "dashboard": {
                     "modules": {
                         "find": ["flows/dashboard"]
```

```
                        },
                        "exclude": {
                          "ids": ["helpers"]
                        }
                      },
                      "customers": {
                          "modules": {
                             "find": ["flows/customers", "!flows/customers/
    flows"]
                           },
                          "exclude": {
                            "ids": ["helpers"]
                           }
                      },
                      "customers-nested": {
                          "modules": {
                             "find": ["flows/customers/flows"]
                           }
                      },
                      "base": {
                        "modules": {
                          "find": [
                            "app-flow.js",
                            "^pages/",
                            "resources/strings",
                            "resources/css"
                          ],
                          "ids": ["helpers"]
                          }
                      }
                  }
              }
          }
      }
    }
```

Exclude Resources from a Bundle

In `gruntfile.js`, you can configure the `vb-require-bundle` task to specify resources that the Grunt optimizer (`grunt vb-optimize`) should exclude from an app's resource bundle when the app is packaged.

Here's an example of a `vb-require-bundle` task configured to create a resources bundle named "my-bundle". The resources that should be included in and excluded from "my-bundle" are listed under the "find" element (in bold):

```
module.exports = (grunt) => {
  grunt.initConfig({
    'vb-require-bundle': {
      "webApps/dcs": {
        "options": {
          "transpile": true,
          "minify": true,
          "bundles": {
```

```
            "my-bundle": {
              "modules": {
                "find": [
                  "app-flow.json",
                  "app-flow.js",
                  "^build",
                  "!^build/components/oj-odcs/",
                  "^flows/",
                  "^pages/",
                  "^resources/strings",
                  "^resources/css",
                  "^services/",
                ]
              }
            }
          }
        }
      }
    }
  }),
  require('load-grunt-tasks')(grunt);
};
```

The “find” element contains a list of regular expressions. In the example above, running the Grunt optimizer will include all the listed resources in the bundle *except* for the resources in the module with the exclamation mark (!) prefix (`!^build/components/oj-odcs/`). You mark resources you want to exclude from the bundle by prepending an exclamation mark (!).

Note:

You can also define the bundles configuration in `settings/build.json`. When developing a visual app, `build.json` is located in `webApps/foo/settings/build.json`; when developing extensions, it is located in `extension1/sources/settings/build.json`.

Here is an example of the `build.json` file:

```
{
    "configuration": {
        "minify": false,
        "bundles": {
            "coolBundle": {
                "modules": {
                    "find": [
                        ".*\\.(json|js)$"
                    ]
                }
            }
        }
    }
}
```

In this example, the "configuration" element corresponds to the "webApps/dcs/options" element in the grunt configuration example above.

If the `build.json` file and the grunt configuration both exist, the `build.json` file has precedence.

Now assume you have a web component in `resources/components/old-component` that contains some third-party libraries that you want to exclude from the bundle (for example, `ckeditor`).

You can exclude the entire component using a line like this:

```
"!^resources/components/old-component",
```

This will exclude all resources in the `old-component` directory, including the component itself. If you want to include the component in the bundle and only exclude the third-party libraries, you can use something like this:

```
"!^resources/components/old-component/older-libraries",
```

where the `resources/components/old-component/older-libraries` directory contains the third-party libraries you want to exclude.

> **Note:**
>
> This does NOT remove the individual files from the deployments. Resources that are not bundled will still be properly loaded in the runtime. The browser will load the required resources, which can impact the performance of your application.

Ignore resources required by dependencies

So you've excluded `older-libraries` in the `find` element, but when you run the optimizer, `older-libraries` is still bundled, breaking the app. Why?

It's because there's code in the bundle that *refers* to **older-libraries** resources. This might be code in `old-component` itself, or it might be some other code in your app that refers to `older-libraries` resources. For example, `app-flow.js` might include something like this:

```
define(['old-component/older-libraries/obsoletemodule'], (obsoletemodule) => {
   obsoletemodule.function();
});
```

When you run the optimizer (`grunt vb-optimize`), the optimizer excludes `older-libraries` resources initially, but then includes `older-libraries/obsoletemodule` in the bundle because of the dependency from `app-flow.js`.

In cases like this, you can add the `emptyPaths` element to the bundle configuration to instruct the optimizer to ignore the dependency (and sub-dependencies):

```
grunt.initConfig({
  'vb-require-bundle': {
    "webApps/dcs": {
      "options": {
        "transpile": true,
        "minify": true,
        "emptyPaths": ["older-libraries"],
        "bundles": {
          "my-bundle": {
            "modules": {
              "find": [
```

This will instruct the optimizer to ignore (not bundle) any module path prefixed with "`older-libraries`".

> **Note:**
>
> The module paths example above assumes the physical resources stored in `/resources/components/*` are mapped in `app-flow.json`. For example, in a web component you don't refer to `resources/components/myWebComponent`, but simply to `myWebComponent`.
>
> The "`find`" element does not take mappings into account, so the patterns must match the physical file paths. However, the "`emptyPaths`" element DOES respect the requirejs paths configuration.

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-deploy` 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-deploy` task in order to deploy to an instance other than where the sources were processed, your edited `Gruntfile.js` might be similar to the following:

```
module.exports = (grunt) => {
  grunt.initConfig({
   'vb-deploy: {
       options: {
           url: 'my production instance URL',
           username: 'production instance username',
       },
   },
    require('load-grunt-tasks')(grunt);
};
```

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:

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

Audit Your Application Using the vb-audit Grunt Task

Visual Builder provides an NPM package (`grunt-vb-audit`) that includes the `vb-audit` Grunt task that you can use to audit your visual applications.

The root folder of your visual application includes two resource files that Grunt uses when it audits applications: `Gruntfile.js` and `package.json`:

File	Description
`Gruntfile.js`	Contains a basic Grunt script that you can modify to add custom audit tasks, and to configure built-in tasks.
`package.json`	Declares the dependency and specifies the URL reference to the `grunt-vb-audit` NPM package that provides the Grunt task to audit visual applications. Visual Builder automatically updates the package version and URL whenever Oracle publishes a new version of the package.

As with the Grunt tasks that you use to build and deploy your visual application, you need to install Node.js and its package manager (npm) on your local system. You also need to save the sources for the visual application on your local system.

To audit your application using `vb-audit`:

1. In the command-line interface, navigate to the folder on your local system containing the `package.json` and `Gruntfile.js` files.
2. Enter `npm install` to retrieve the node dependencies required to audit the application. The install command retrieves the `grunt-vb-audit` NPM package defined in `package.json`.
3. Enter `grunt vb-audit` in the command-line interface to audit your application. The following illustrative example shows how you execute this task.

```
# Audit application sources
./node_modules/.bin/grunt vb-audit \
--url=<https://Visual-Builder-runtime-instance-url/ic/builder/> \
--username=<Visual-Builder-runtime-username> \
```

```
    --password=<Visual-Builder-runtime-password> \
    --id=<your visual app ID> \
    --ver=<your visual app version>
```

You can omit the task parameters (`URL`, `username`, and so on) if the visual application that you audit does not include modules that require a server connection, such as business objects, for the audit to complete. If you do not specify values for the task parameters when a server connection is required, error messages appear in the command-line interface and the corresponding modules fail the audit.

By default, `vb-audit` audits all files in your application according to the following glob patterns:

```
[
    '**/*',
    // Ignore npm directories during audit
    '!node/**',
    '!**/node_modules/**',
]
```

You can override the files to audit with your own glob patterns. The exclusion patterns in the default glob pattern are appended to the glob patterns that you supply.

Configure Audit Options in Gruntfile.js

You can specify server connection properties and `vb-audit` properties in your visual application's `Gruntfile.js` file.

The following example shows available options.

```
grunt.initConfig({
    vb: {
        url: '<VB-instance-url>',
        id: <your visual app ID>,
        ver: <your visual app version>,
        username: <username>,
        password: <password>,  // This is not encrypted
    },
    'vb-audit': {
        options: {
            config: {
                auditors: {
                    // Disable specific auditors
                    'lcm.VirtualRolesAuditor': { // Requires backend
                        enabled: false,
                    },
                    'components.InstalledComponentsAuditor': { // Requires
backend
                        enabled: false,
                    },
                    'serviceConnectionAuditor': { // Requires backend
                        enabled: false,
                    },
                    'deploymentProfileAuditor': { // Requires backend
                        enabled: false,
                    },
```

```
                },
            },
            files: [ // Globbing patterns to use for this application
                '*',
                'process/**',
                'services/**',
                'settings/**',
                'webApps/**',
                '!**/package-lock.json',
            ],
            processResults: (results) => { // Supply an alternate results
processor
                grunt.log.writeln(`Processed ${results.auditedResourceCount}
files`);
            },
            outputFile: 'auditoutput.txt', // Output file to receive the
audit results (e.g. auditoutput.txt, auditoutput.csv)
        },
    },
});
```

The entries in `'vb-audit': { options: { config: {` are passed to the audit system and override or augment the options that you set in your visual application's `settings/audit.json` file if you configured options in the latter file using Visual Builder's Audits feature.

For output, you can specify `processResults` or `outputFile`. If you specify neither, `vb-audit` writes the output to the command-line interface. Note that the example `Gruntfile.js` shown above displays both options, as it is an illustrative example of available options. Omit the option (`processResults` or `outputFile`) that you do not want to use, or omit both options if you want to write output to the command-line interface.

The results parameter for `processResults` has the following format:

```
{
    auditedResourceCount, <number of resources audited>
    totalResourceCount, <number of resources checked (some resources may not
have auditors)>
    issues[{
        ruleId, // String id of the rule
        severity, // ['error' | 'warning' | 'info' | 'todo']
        message, // String message
        filePath, // resource's file path
        location: {
            line,
            col,
            endLine,
            endCol,
            length,
        },
    }]
}
```

You can specify target-based options in `Gruntfile.js` that override or augment your standard options. The following example audits the modules in the `webApps` directory:

```
grunt.initConfig({
    vb: {
        ...
    },
    'vb-audit': {
        options: {
            ...
        },

        webApps: { // target to audit the modules in the webApps directory
            files: [
                'webApps/**'
            ],
        },
    },
});
```

To audit the webApps target, run the `vb-audit` task and specify the target:

```
./node_modules/.bin/grunt vb-audit:webApps
```

> **Note:**
>
> grunt.config.merge merges the target options, such as `webApps` in our previous example, into the `vb-audit` options. It merges array and object properties recursively. If, for example, `options` defines a files array with 5 elements and you provide a target options with a files array that has 2 elements, the first 2 elements in the `options` files array will be overwritten and the remaining 3 elements remain unchanged.
>
> A difference between the `files` option in the `Gruntfile.js` file and the `paths/exclude` option in your `settings/audit.json` file is that the `Gruntfile.js` files option assembles the set of files that `vb-audit` sends to the audit system, whereas the `audit.json` patterns are ignored by the audit system. In particular, if you supply a `files` option in your `Gruntfile.js`, you need to have an exclusion pattern for something like `node_modules`, as processing the thousands of files under `node_modules` can be time consuming. This exclusion pattern is included in the default configuration.

Override Configuration Options in Gruntfile.js

You can override the `Gruntfile.js` configuration options for `vb-audit` by specifying properties on the command-line interface.

The following example shows available options.

```
./node_modules/.bin/grunt vb-audit \
    --audit.files='webApps/**' \
    --
audit.disabledauditors='lcm.VirtualRolesAuditor,components.InstalledComponents
```

```
Auditor' \
    --audit.outputfile='auditoutput.txt'
```

Part VI
Use Cases & Troubleshooting

Look here for guidance on common use cases and issues in Oracle Visual Builder.

Topics:

- Common Use Cases
- Troubleshooting & FAQs

29

Common Use Cases

Find guidance for some common scenarios in Visual Builder:

- Work With Code Samples
- Change an Application's Logo
- Style and Theme Visual Builder Applications
- Add Login and Logout Capabilities to an Application
- Create a Custom Lock Page
- Access Data in an Existing Oracle Database Using ORDS
- Use a SOAP Web Service With Visual Builder
- Run Visual Builder Applications On Other Servers
- Embed a Web App in an Oracle Cloud Application
- Call Server-side Functionality from Visual Builder
- Add the Oracle Digital Assistant to Your Web App
- Abort Pending REST Calls in Visual Builder
- Forms
- Tables
- Components
- Pages and Flows
- Business Objects

Work With Code Samples

If you're looking for Visual Builder samples that demonstrate different use cases, the **Visual Builder Cookbook** is for you. The cookbook provides a collection of step-by-step recipes that describe how to implement different techniques to develop your application in Visual Builder. You also get the complete code for each sample, so you can inspect and learn from it.

Here's how you can access the cookbook right from your Visual Builder instance:

1. Click any application on the Visual Applications Home page.
2. Click **Menu** in the upper right corner and select **Code Examples**.

This opens the cookbook hosted as a Visual Builder application at https://vbcookbook.oracle.com/.

You can also create your own copy of the cookbook on your Visual Builder instance:

1. Click **New** on the Visual Builder Home page to create a new application.
2. Enter a display name for the application and click **Change Template**.
3. Select **Oracle Visual Builder Cookbook** as the app's template.

4. Click **Select**, then **Finish**.

This installs a complete copy of the cookbook in your environment. The cookbook app will have everything you need to run the samples, including the backing business objects, and the code for all the samples.

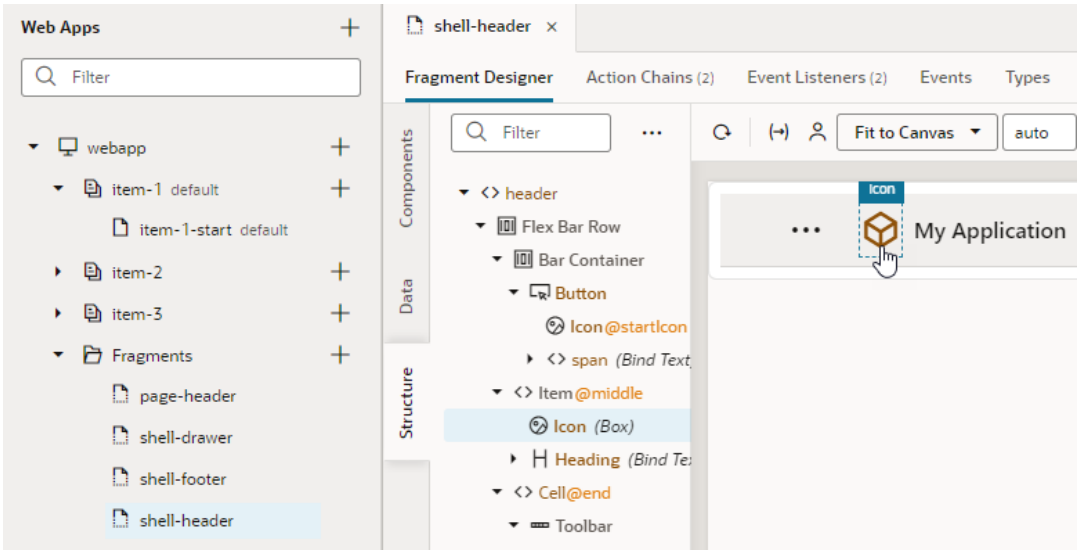
Change an Application's Logo

When a web app uses the Redwood theme, here's how you can customize the application to use your company logo.

> **Note:**
>
> Company logo and application title only display in desktop mode.

1. In the Navigator, click **Web Applications** and expand your web app.
2. Expand **Fragments** and double-click **shell-header** to open it in the Page Designer.

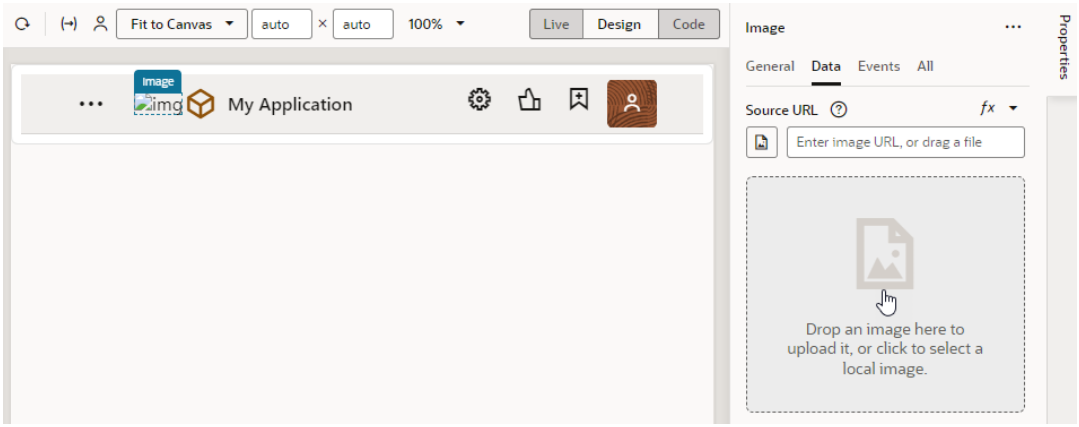
   Web apps by default use a Redwood icon as a placeholder for the company logo. You'll need to remove this icon and add your own image.



3. Select the icon component on the canvas, then click **Code** to switch to the fragment's code view.
4. Add an `img` tag just above the `span`, similar to `<img :src="" ></img>`. Place your cursor inside the `src` attribute's quotation marks, then click **Design** to return to design view:

```
12  <!-- Company logo and application title only displayed in desktop mode -->
13  <div class="oj-flex-bar-middle oj-sm-margin-4x-horizontal oj-sm-align-items-center">
14    <!-- Please Change the following for company specific logo   -->
15    <img :src="" ></img>
16    <span class="oj-typography-heading-lg oj-sm-margin-2x-end oj-icon-color-warning oj-ux-ico-box"
17      title="Application Logo"
18      alt="Application Logo"></span>
19    <h1 class="oj-typography-body-md oj-typography-semi-bold" title="Application Name">
20      <oj-bind-text value="My Application"></oj-bind-text>
21    </h1>
22  </div>
```

5. In the image's Properties pane, click the **Data** tab.
6. Drag your image into the drop target area in the Data tab.



Once you add the image, the Source URL field updates to show the path to the image.

7. Click the **All** tab to view and edit image attributes. For example, you can add a description in the **Alt** field and select `oj-sm-margin-2x-end` in the **class** field to add space after the image.
8. After you've made your changes, delete the icon component on the canvas.
9. Click **Preview** to see how your logo appears.

Style and Theme 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.

Visual Builder applications created with version 20.10 or later, by default, use the Redwood theme. Redwood is the Oracle standard for application look and feel. It includes components such as Waterfall Layout and Action Cards that enrich user experience, besides a collection of icons that you can readily leverage in your apps.

Because Redwood achieves its look and feel through hundreds of custom properties (also called CSS variables), you can override these variables to customize the default look and feel for your requirements.

> **Note:**
>
> Styling applications is supported only for web applications that use the Redwood theme. If you're styling applications that use the Alta theme, note that support for Alta themes has been deprecated since JET 10.

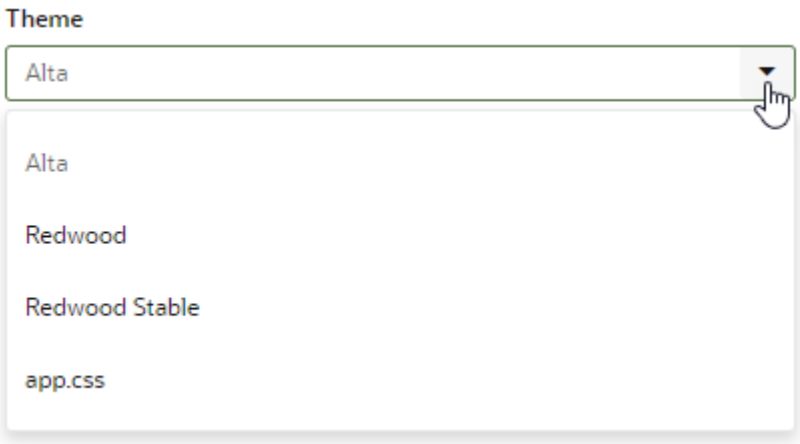
When you style your Visual Builder or JET applications, it's important to use theming correctly. Otherwise, you run the risk of finding that your re-styling breaks when you upgrade your platform versions. For more information about theming your application, see "Use CSS and Themes in Oracle JET Apps" in *Developing Oracle JET Apps Using MVVM Architecture*. To get the latest version, go to https://www.oracle.com/webfolder/technetwork/jet/index.html, click **Help and Support**, then scroll to the bottom and click **Theming**.

Transition a Web (or Mobile) App's Theme to Redwood

Visual Builder applications created with version 20.07 or earlier, by default, use JET's Alta theme as the base UI theme. If your app uses the Alta theme, we strongly urge you to transition your app to use the Redwood theme before support for the Alta theme ends.

To move your web (or mobile) app's theme from Alta to Redwood:

1. Select the application node in the Navigator.
2. Click **Settings**, then look for the **Theme** field in the **General** tab.
3. If the Theme is set to **Alta**, you can switch to **Redwood** or **Redwood Stable** (recommended) as the base theme.



   Because component dimension and styling have changed, make sure you verify the look and feel. You may have to redesign the app if needed.

With the theme now set to Redwood, you can:

- Override the default Redwood styles to suit your needs.
- Override the appearance of specific component instances. For example, if you've made a `div` element clickable, you may want to add a class called `clickable` to the `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.

Customize a Web App's Appearance

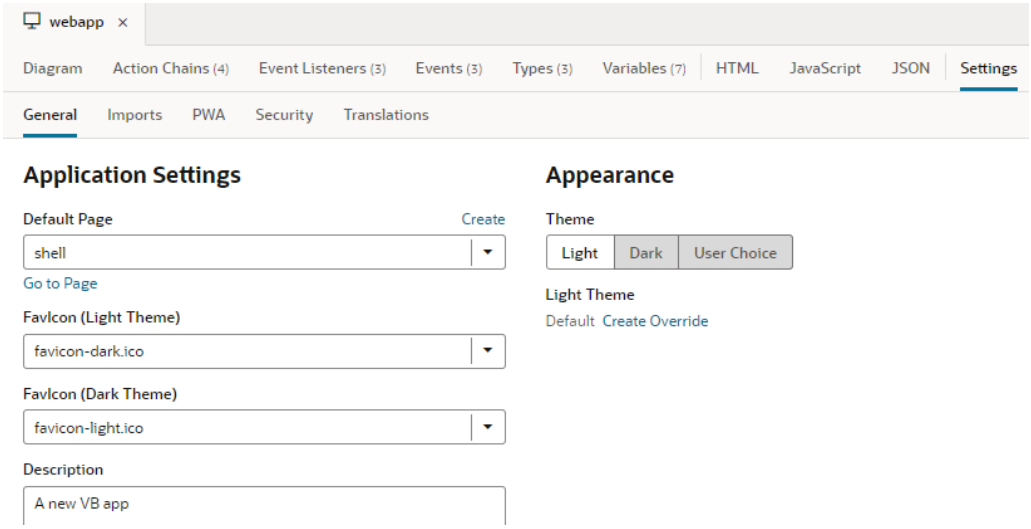
Web apps, by default, display against a light background with dark text. For web apps created with version 24.07 or later, you can customize the app's theme to show its components against a dark background, even let your users choose between a Light and Dark theme. The application's UI automatically adjusts to the theme you (or your end-user) chooses.

You can also use CSS variables to override the app's default look and feel, which is based on the *Redwood* base theme. The Redwood theme provides the Oracle look and feel for your app and inherits future updates to the Redwood theme, but these changes can potentially bleed into an app's custom theme. When you use CSS overrides to customize the default theme, the base theme switches to *Redwood Stable*, which is meant to minimize changes bleeding into a custom theme. With this option, your app uses the Stable CSS file as the base theme, but you override some variables in a separate CSS file to customize the base theme's look and feel.

The advantage of overriding CSS variables in a separate file is that you won't need to rebuild your web app with each new version of JET. Whenever the default Redwood theme changes, those updates will be picked up by your application's CSS files without requiring changes.

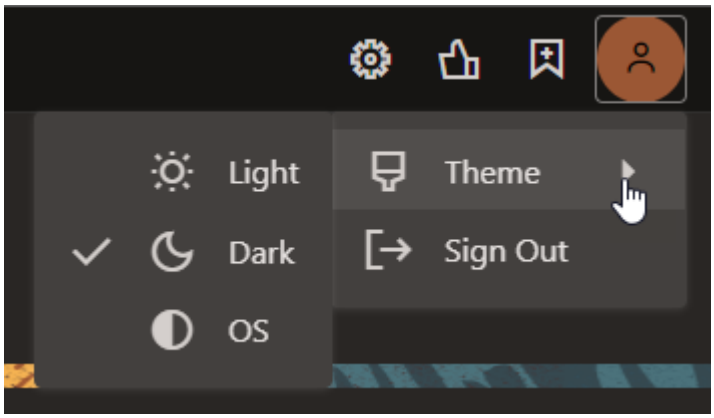
- Follow these steps to switch a web app's theme and optionally override CSS variables for apps created with version 24.07 or later:

  1. In the Navigator's **Web Applications** tab, select your application and click the **Settings** tab:



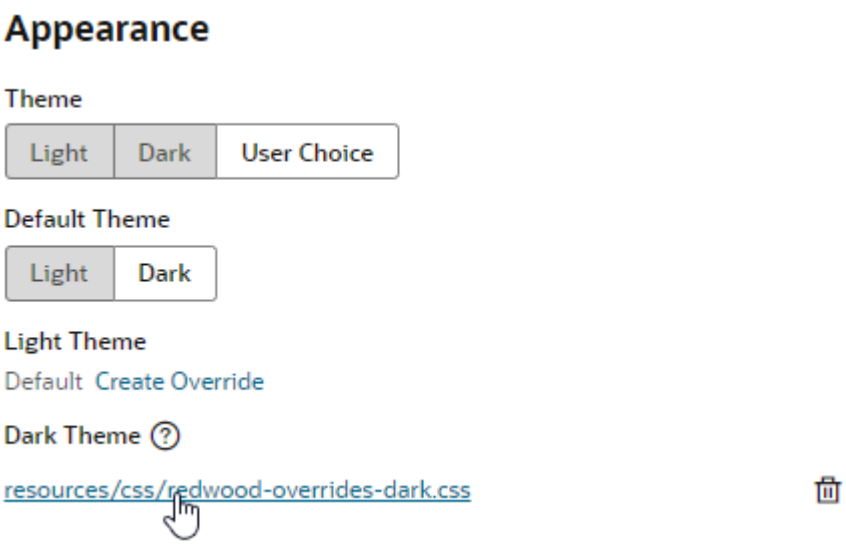
  2. Select a **Theme** under Appearance:
     - Select **Dark** to switch your app to a dark color display where components render against a dark background with light text.
     - Select **User Choice** to give your users the ability to set their preferred theme for the app, then select either **Light** or **Dark** as the default theme:

When you choose this option, users will be able to change the app's theme from the Avatar menu: **Light**, **Dark**, or **OS** (to inherit the theme specified in their OS setting):



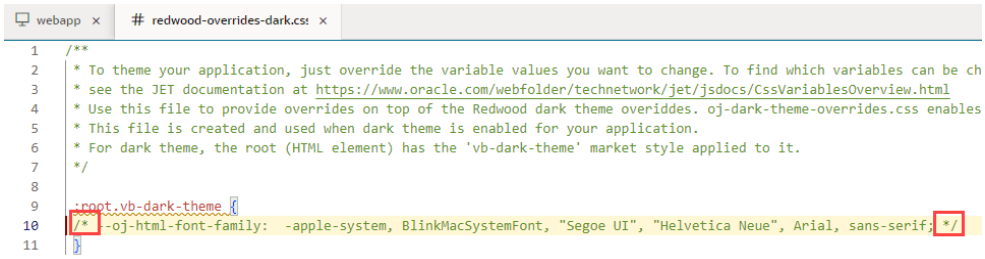
- If you changed the default, select **Light** to switch your app back to a lighter background with dark text display.

3. Preview your application to see what it looks like with your selected theme.
4. Optional: Override the default Redwood look and feel using CSS variables. Choose this option if you want to add a limited number of app-specific overrides to the base theme.
   a. Click the **Create Override** link for your theme.
   b. When a link to the new `redwood-overrides.css` or `redwood-overrides-dark.css` file is created, click the file to open it:



c. Specify the variables that you want to override. For a list of variables that can be changed, see JET documentation at `https://www.oracle.com/webfolder/technetwork/jet/jsdocs/CssVariablesOverview.html`.

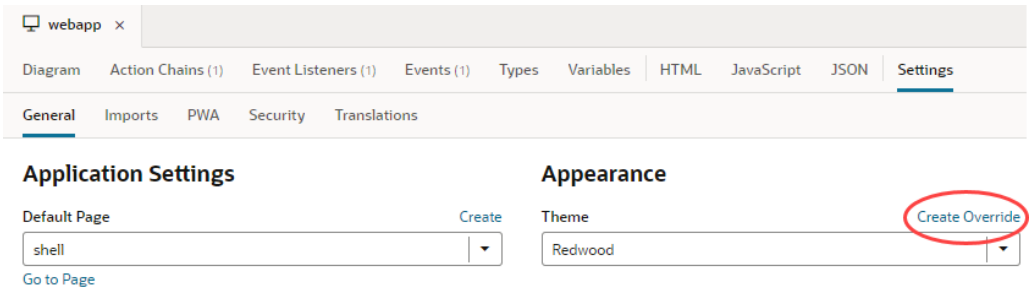
   For example, to override the font used by the Redwood Dark theme for your application, add the `--oj-html-font-family` variable and change its values. Make sure you remove the `/*` and `*/` before and after the variable to uncomment it:



d. Check your application and verify the changes.

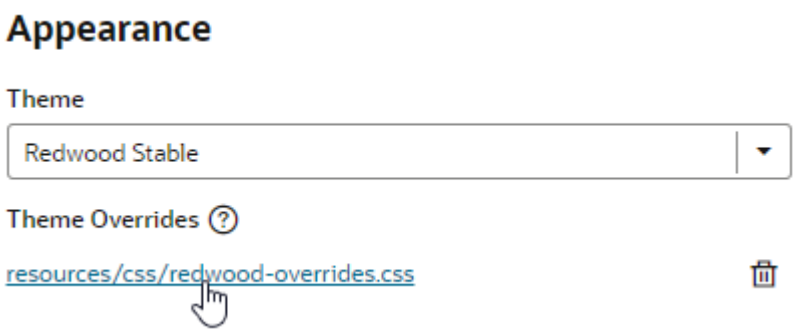
- Follow these steps to override CSS variables for apps created with version 24.01 or earlier that use the Redwood theme:

  1. In the Navigator's **Web Applications** tab, select your application and click the **Settings** tab.
  2. When the Theme is set to Redwood, click the **Create Override** link:



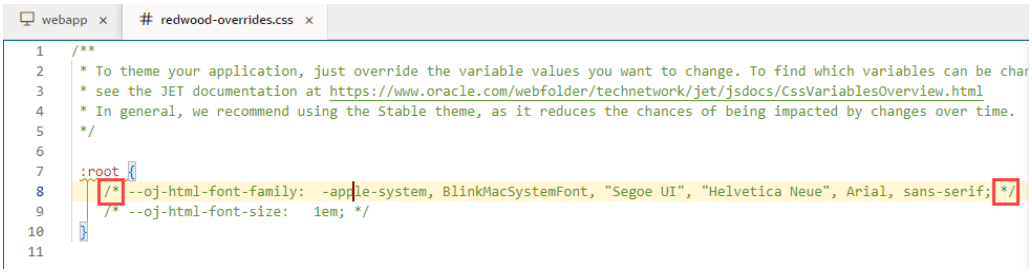
When customizing the theme, we recommend you choose **Redwood Stable** to reduce the chances of your app being impacted by future updates. This way, you only override the variables you want to change while inheriting all other updates to the Redwood theme.

3. When a link to the new `redwood-overrides.css` file is created, click the file under **Theme Override** to open it:



4. Uncomment and change the values of the variables you want to change.

   To do this, remove the `/*` just before the variable and the `*/` after it, then update the variable's value. For example, to override the font used by the Redwood theme for your application, uncomment and change the `--oj-html-font-family` variable's values:



   You can also add variables that you want to override. For a list of variables that can be changed, see JET documentation at `https://www.oracle.com/webfolder/technetwork/jet/jsdocs/CssVariablesOverview.html`.

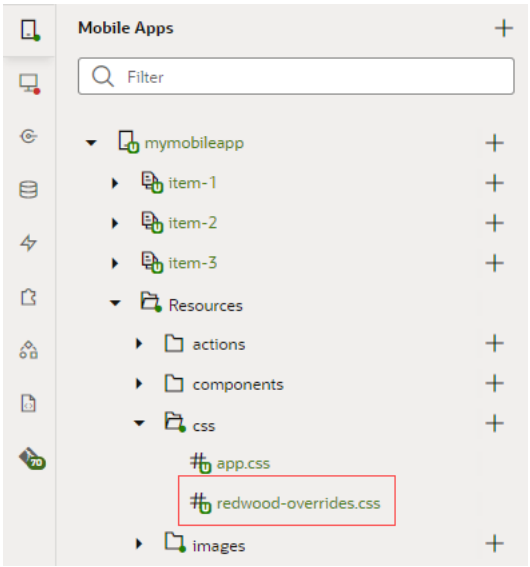
5. Check your application and verify the changes.

Override the Redwood Theme for a Mobile Application

If you would like to override the Redwood theme in your mobile app, you need to create a new CSS file and update `app-flow.json` to include the new CSS.

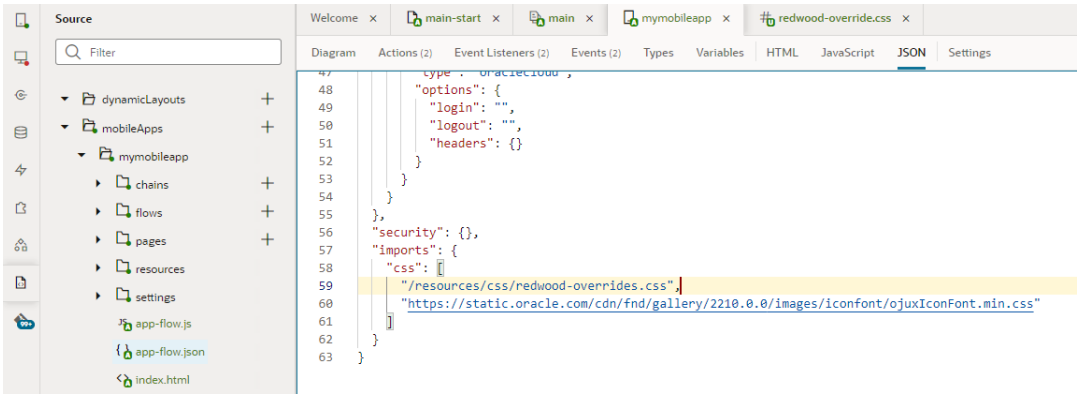
> **Note:**
>
> Mobile applications have been deprecated in favor of PWAs. You can continue to use an existing mobile app by deploying it as a PWA until July 2024 when mobile apps, including PWA-enabled ones, reach End of Life (EOL). See Run Mobile Applications as PWAs.

1. In your mobile app, expand the **Resources** node, and add a file named `redwood-overrides.css` to the **css** folder.



2. In the Source view, select `app-flow.json` and to the `"imports": { "css"` section, add:

   `"/resources/css/redwood-overrides.css"`



Alternatively, you can import the CSS file to the mobile app settings. See Manage Custom Component, CSS, and Module Imports.

Add a Custom Style to a Component

When you want to customize the appearance of specific component instances, you create a style class and define the style in your app's stylesheet, then assign that class to the specific component instance you want to override.

Some style classes are predefined in the app and are automatically applied to components when you add them to a page. Specific predefined 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`. Predefined style classes used by Oracle JET components are prepended with `oj-`.

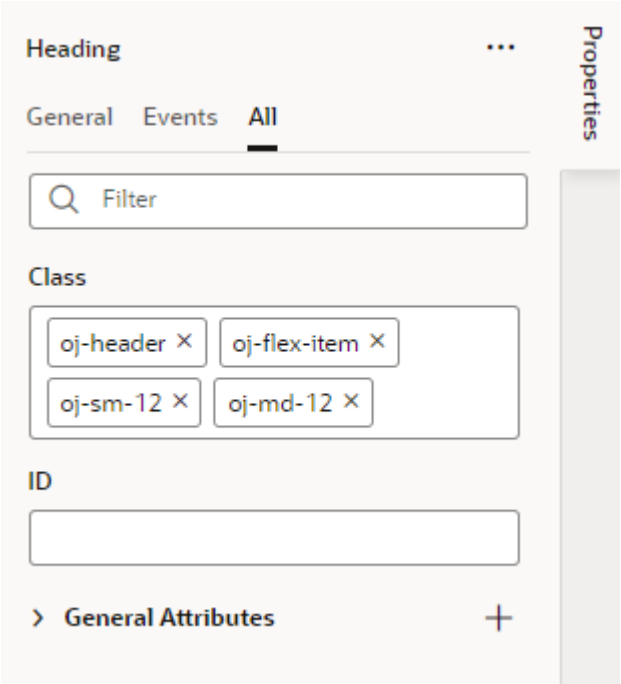
> **Note:**
>
> As a general rule, you should not override or modify the predefined classes or remove them from components. When defining and adding a custom class to a component, you should exercise caution to ensure that your class does not conflict with the predefined classes already applied to the component.

You can define your custom style classes in the `app.css` stylesheet of your app. An empty `app.css` stylesheet is 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 Properties pane in the Page Designer's Design view or in the page's Code editor.

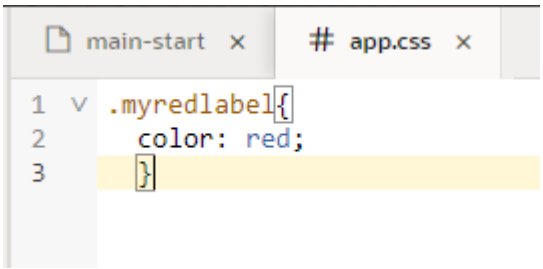
To add a custom style to a component:

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 located in the All tab of the Properties pane. 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`.



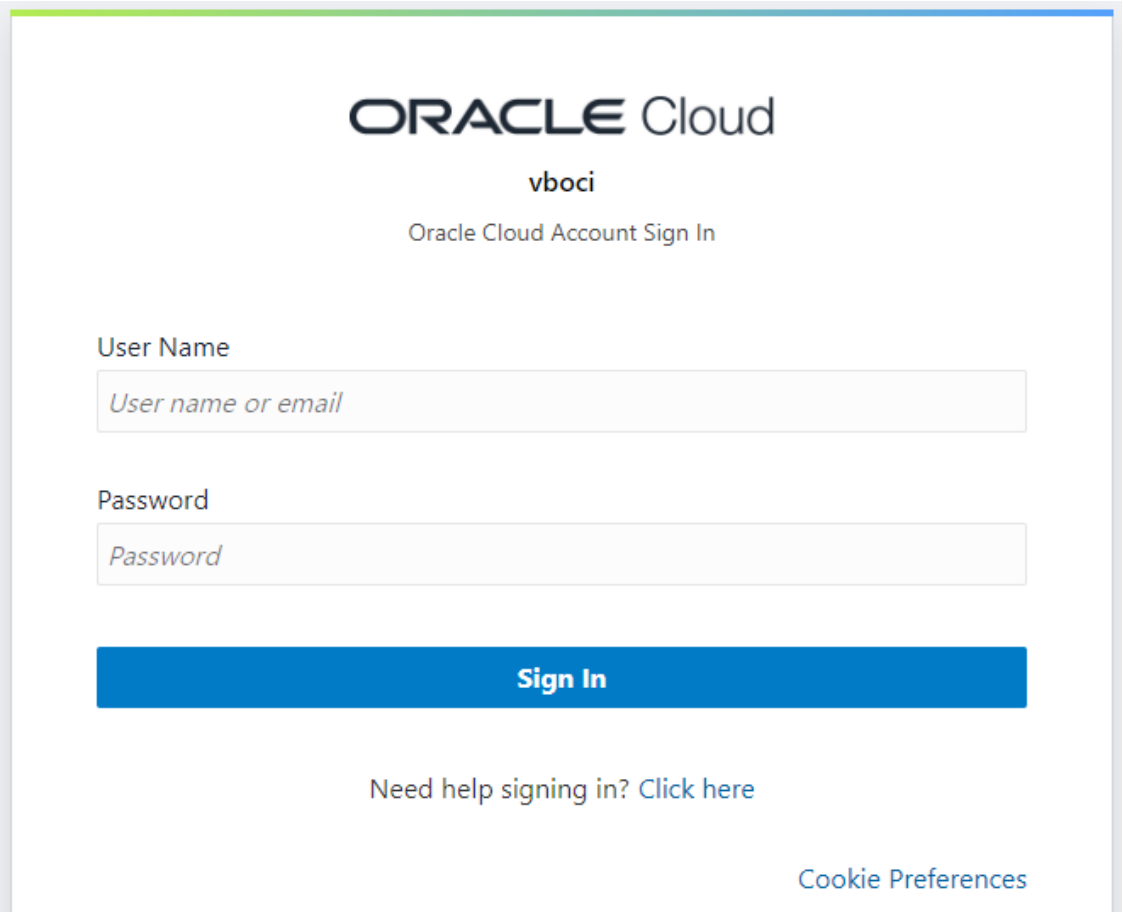
Reload the page in the Page Designer to see the class applied to the component.

Add Login and Logout Capabilities to an Application

Visual Builder applications provide built-in options for you to implement login and logout for your users.

Customize Application Login

By default, any application you build in Visual Builder includes a login screen—unless you enabled anonymous access that allows users to access your app without signing in. The default login screen points to the Sign-In page that Oracle Identity Cloud Service (IDCS) provides for token-based authentication:

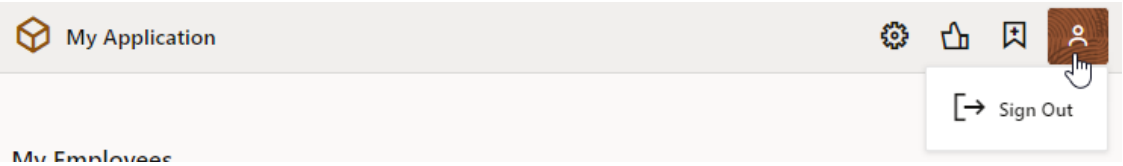


If you want to customize this sign-in page (you'll need rights to register applications in IDCS), you can use the Branding feature to change the company name and the login text, as well as upload logos to replace the defaults. The position of the text and images, and the colors and fonts, remains the same. For anything beyond what the branding feature supports, you'll need to use the Authentication REST API that IDCS provides to help you develop your own sign-in page.

Enable Application Logout

You can enable a logout function for your application by adding the built-in Logout action to any page component, for example, a button or a menu item.

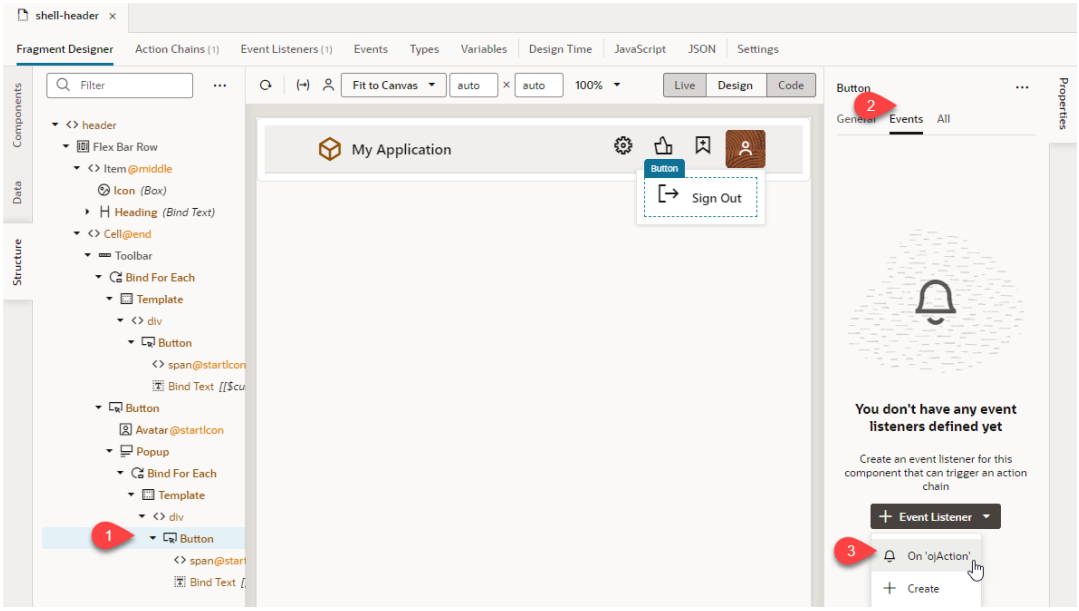
Web applications in Visual Builder come with a default shell that displays a **Sign Out** option under the logged-in user's Avatar, but you'll need to add the Logout action to the component to actually trigger a logout:



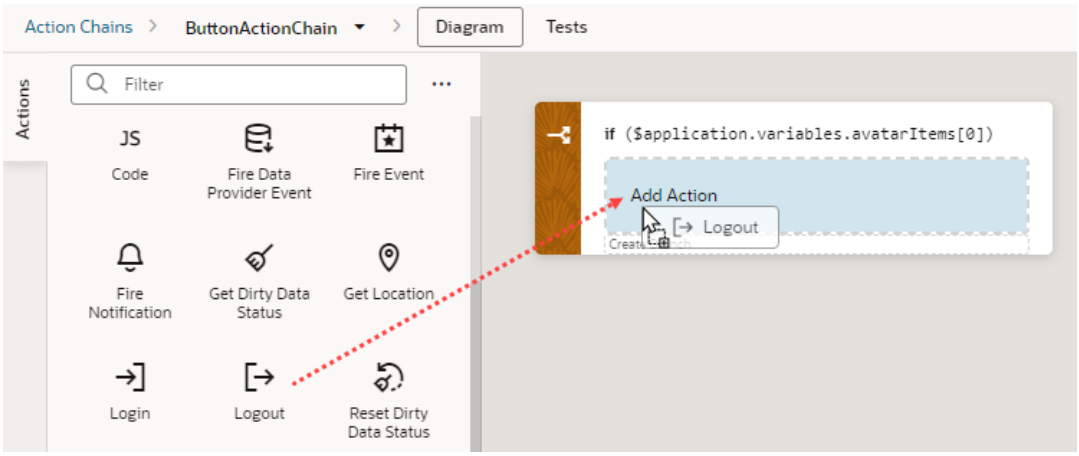
> **Note:**
>
> The **Sign Out** option doesn't appear for apps enabled as PWAs, but you can enable the same functionality for these apps by calling the Logout action from any page component.

To add a Logout action to a page component:

1. Open your application in the Navigator.
2. In the Structure view, select the component you want to add the Logout action to, then click the **Events** tab in the Properties pane, click **+ Event Listener**, and select **On 'ojAction'**. To enable logout for a web app's default Sign-In option:
   a. Open the app's `shell-header` fragment under **Fragments**.
   b. Click the Structure view and locate the Avatar's hidden **Button** component. You can also switch to Live mode and select the Sign Out option to see the correct Button selected in Structure view; switch back to Design mode.
   c. Click the **Events** tab in the component's Properties pane, then click **+ Event Listener** and select **On 'ojAction'**.



3. In the Action Chain editor, drag and drop the **If** action from the Actions palette onto the canvas, then in the action's Properties pane, set the **Condition** field's value as `[[ $application.variables.avatarItems[0].id ]]`.
4. Drag and drop the **Logout** action onto the **Add Action** area of the If action to indicate the action you want to follow:



If are using an external identity provider, enter the provider's logout endpoint in the **Logout URL** field in the action's Properties pane, something like `https:***/oam/server/logout?end_url=https://****/oamwebsso/logout-success.jsp`. If you are using IDCS for authentication, you don't need to specify the logout URL.

When you are done, run your app to check whether you're being logged out of all active sessions (in the same browser) associated with the same identity domain.

> **Note:**
>
> The logout action won't work when you preview the app in Live mode (to avoid logging you out during development). You'll need to stage or publish your app to make sure logout works as expected.

Redirect URL After Logout

A post-logout URL always points back to the deployed app (because the server runtime logout code isn't aware of changes made in the IDCS client app). One option is to use the IDCS logout directly (instead of the Visual Builder logout URL) and specify your post-logout URL in a query parameter, for example:

```
https://servicename-cloudaccount.builder.ocp.oraclecloud.com/
mycompany/logout.html?postlogouturl=https://servicename-
cloudaccount.ocp.oraclecloud.com%3A443%2Fic%2Fbuilder%2Frt%2F<AppName>
%2F<Version>%2FwebApps%2F<WebAppName>%2F
```

Create a Custom Lock Page

You can create a custom page that displays when someone tries to access an application that you have locked for maintenance.

You can apply the custom page to either a visual application or an individual web application.

- To apply the page to the entire visual application, import the file to the visual application at the root level.
- To apply the page to an individual web application, import the file to the web application resources.

You can create more than one `app-locked.html` page and import each one to a different location. Any custom pages applied at the web application level will override the setting at the visual application level.

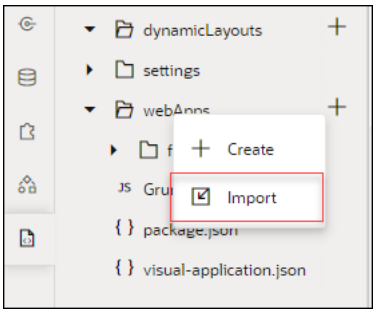
> **Note:**
>
> You will need to create the custom page and add it to the visual application or web application *before* locking the application.

Apply a Custom Lock Page to a Visual Application

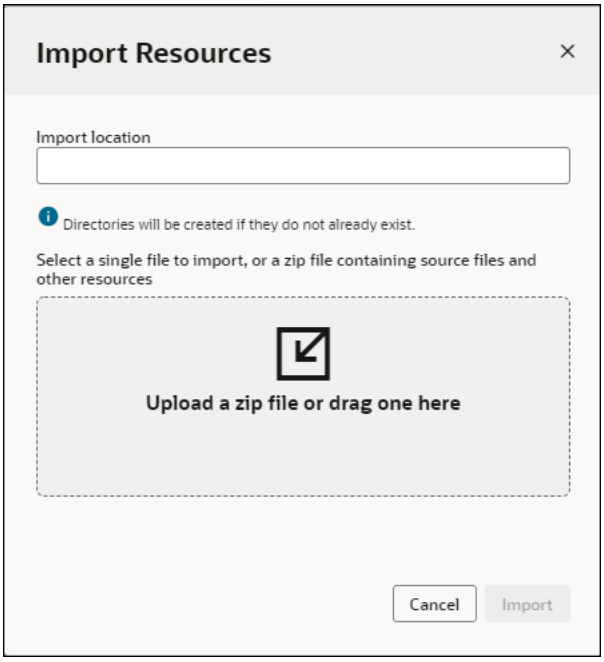
You can create a custom lock page and add it to the root level of a visual application. When someone tries to access the locked application, the custom page will display.

Before locking the application, create a custom `app-locked.html` page, then import the file to the root of your visual application.

1. Create your custom page and save it as `app-locked.html`.
2. Add the custom page to the root of the visual application.
   a. Open your web application in the Navigator.
   b. Click **Source View**.
   c. Right-click the **webApps** directory and choose **Import** in the popup menu:



   Alternatively, drag the file from your local file system onto the **webApps** directory in the Navigator.
   d. In the **Import Resources** dialog box, remove `webApps` from the **Import location** field, then click the drop target area and navigate to the file on your local system.



e. Click **Import** to import the file.

The `app-locked.html` file is added to the root of your visual application.

3. Apply the lock to your application (See).

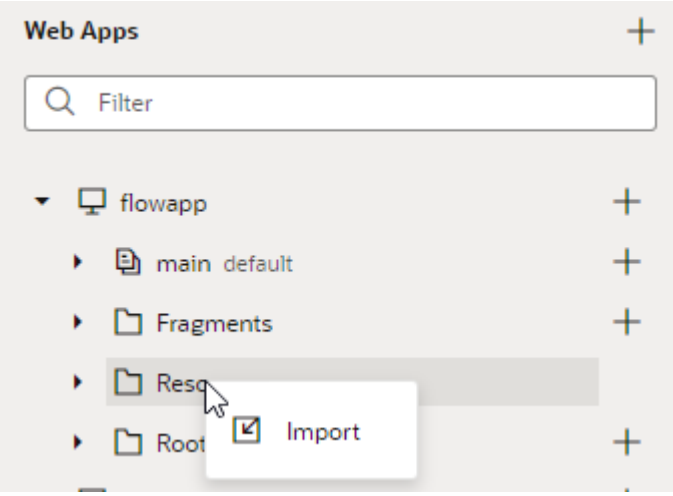
When someone tries to access the locked application, the custom `app-locked.html` page that you added to the visual application root displays, unless unique `app-locked.html` pages have been applied to individual web applications. The page at the web application level will override the page at the visual application root level.

Apply a Custom Lock Page to a Web Application

You can create a custom lock page and add it to a web application's resources. When someone tries to access the application, the custom lock page displays, overriding any page applied to the visual application level.

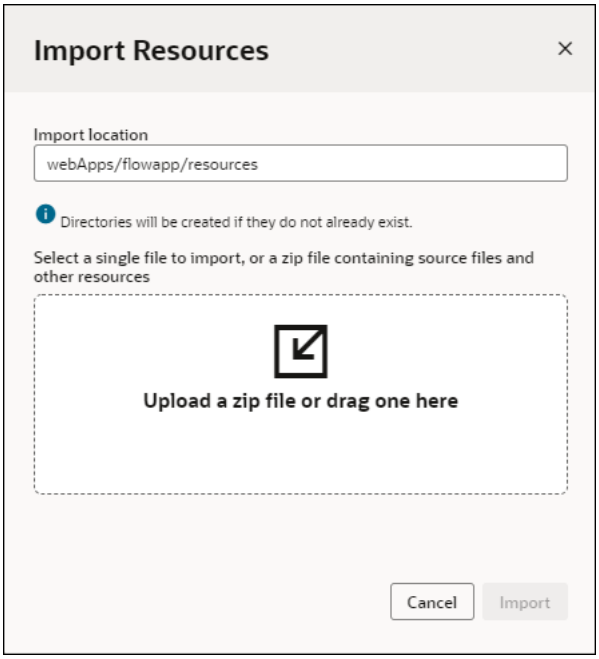
Before locking the application, first create the custom `app-locked.html` page, then import it to the resources section of your web application.

1. Create your custom page and save it as `app-locked.html`.
2. Add the custom page to the root of the visual application.
   a. Open your web application in the Navigator and locate the **Resources** folder.
   b. Right-click the **Resources** folder and choose **Import** in the popup menu.



Alternatively, drag a file from your local file system onto the **Resources** folder in the Navigator to open the **Import Resources** dialog box.

c. In the **Import Resources** dialog box, click the drop target area and navigate to the file on your local system.



d. Click **Import** to import the file.

The `app-locked.html` file is added to the resources of your web application.

3. Apply the lock to your application (See).

When a user tries to access the locked application, the custom `app-locked.html` page displays for this web application, overriding the page applied visual application root level.

Use a SOAP Web Service With Visual Builder

Visual Builder is designed to consume REST natively. We recommend that you use a SOAP to REST transformation on the server. You may want to perform this transformation using Oracle Integration.

If you prefer to handle the SOAP translation on the client as part of your VB application, you can follow these steps to communicate with the SOAP service:

1. Define a service connection to a SOAP endpoint.
2. Write JavaScript code to call the SOAP service using the restHelper method. See REST Helper in the *Oracle Visual Builder Page Model Reference*.
3. Parse the XML messages to and from the endpoint in your JavaScript code.

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 backend.
- 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 How Does the Fixed Credentials Authentication Method Work? 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 and Audit Your Code Using Grunt 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.

> **Note:**
>
> While these options make it possible to host your Visual Builder app anywhere, developing your app using Visual Builder is only available through the Cloud, so the REST services you want to connect to must be exposed for Cloud access. To achieve this for on-premise systems, one option is to use the Connectivity Agent in Oracle Integration. See Manage the Agent Group and the On-Premises Connectivity Agent in *Using Integrations in Oracle Integration 3*.

Embed a Web App in an Oracle Cloud Application

You can edit an Oracle Cloud Application to embed your web app using Page Composer or Application Composer. For your embedded web app to work in an Oracle Cloud Application, you'll want to confirm that:

- Your Visual Builder instance is associated with Oracle Cloud Applications;
- Single Sign-On (SSO) is enabled;
- The authentication for your web app is using an Oracle Cloud Account.

> **Note:**
>
> Contact your service administrator or project owner if you are unsure about how your Visual Builder instance is configured.
>
> Specifically, the **Allow only secure applications to be created** setting in the instance must be unchecked to prevent users from seeing this message: `Refused to display 'https://idcs-domain.identity.oraclecloud.com/ui/v1/signin' in a frame because it set 'X-Frame-Options' to 'deny'.`

When a user is logged in to an Oracle Cloud Application, the application provides the user's Oracle Cloud Account details for authentication when accessing the embedded web app, so the web app will appear in the application's page without requiring any additional login. This authentication only occurs when the user first accesses the embedded web app and may mean that the web app takes a long time to load. Subsequent accesses will be faster.

To add your web app to an Oracle Cloud Application, you'll need to:

1. Prepare your web app in Visual Builder;
2. Publish your web app;
3. Embed your web app using Page Composer or Application Composer.
   For more information on which one you will use with your Oracle Cloud Application, see Differences Between Using Page Composer and Application Composer in *Configuring and Extending Applications*.

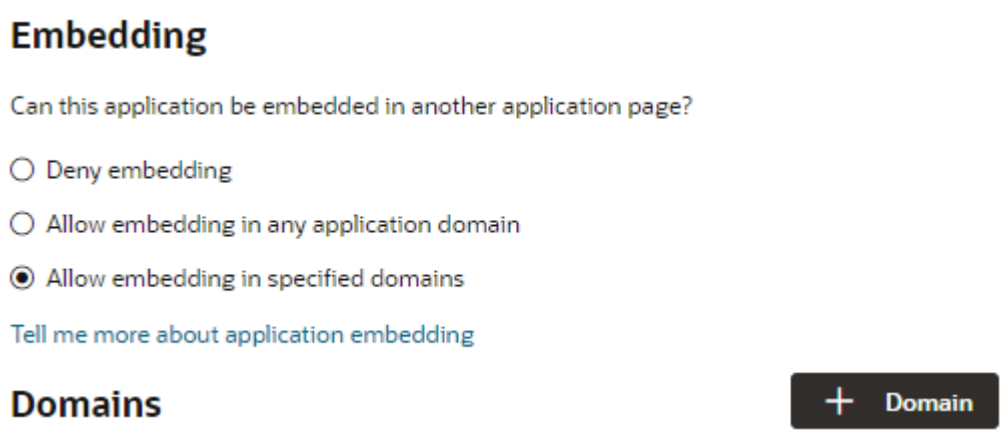
Make Your Web App Ready for Embedding

Before embedding your app in an Oracle Cloud Application, you'll want to configure your web app settings to allow embedding, and modify your shell page to remove the app's default header and footer. You might also want to choose a theme for your application that matches the look and feel of the application where you are embedding it.

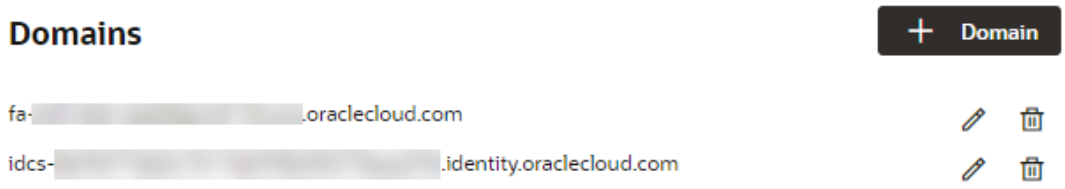
When embedding a web app, it is quite common to set up your web app so that the Cloud Application passes some values to the web app as parameters in the URL. The input parameters are usually assigned to app-scoped variables defined in your app. The variables need to have the **Pass on URL** option set in the Properties pane.

To get your app ready for embedding:

1. In your web app's Settings editor, open the Security tab and select **Allow embedding in any application domain** or **Allow embedding in specified domains**.

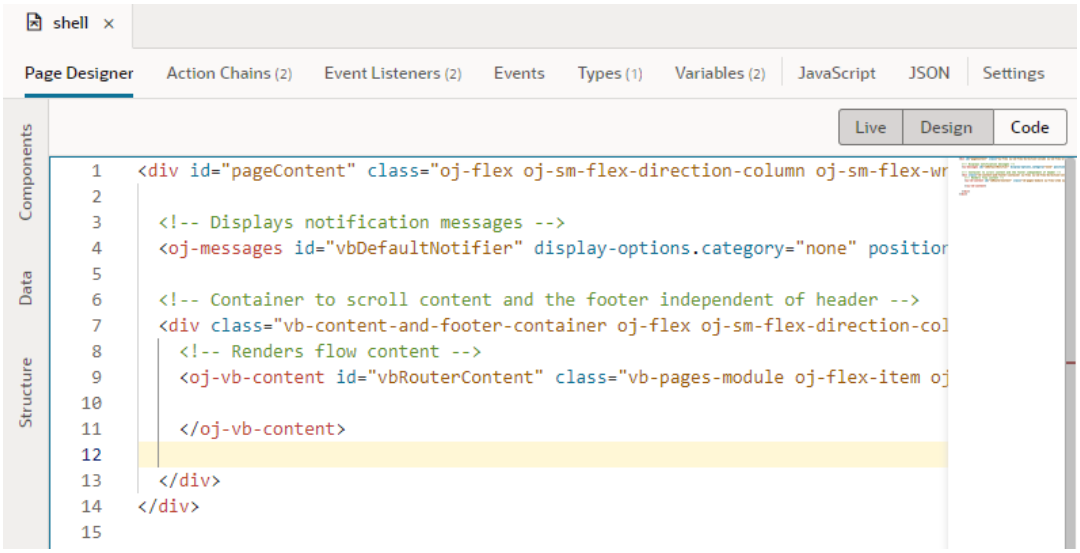


2. If you chose **Allow embedding in specified domains**, enter your Oracle Cloud Application instance's domain (for example, `fa-`*`identifier`*`.fa.ocs.oraclecloud.com`. Make sure you also add the domain's IDCS host name (for example, `idcs-`*`identifier`*`.identity.oraclecloud.com`).



   You can add multiple domain names where embedding is allowed, but typically only one IDCS host name is required.

3. If you want to change your app's theme, open the General tab and select the theme from the drop-down list in the General tab.
4. In the Navigator, expand Root Pages and open the `shell` page in the Code editor.
5. Delete the `<header>` and `<footer>` elements in the code.



Your shell page should now only have `<div>` elements for the page, message notifications, and the content. In Design view, you can see that the app now only contains the core content.

6. When your app is ready to be embedded, stage and publish the app so that the app is accessible at a public URL.
7. Open the live app in your browser to confirm that the page renders correctly at the URL.

   When testing the URL, you might also want to test that passing your app parameter in the URL works correctly, for example, by including the variable name and some value in the URL (`https://vbinstanceurl/.../appname/?VariableName=Value`)
8. Copy the URL, and make a note of the app parameters you are using.

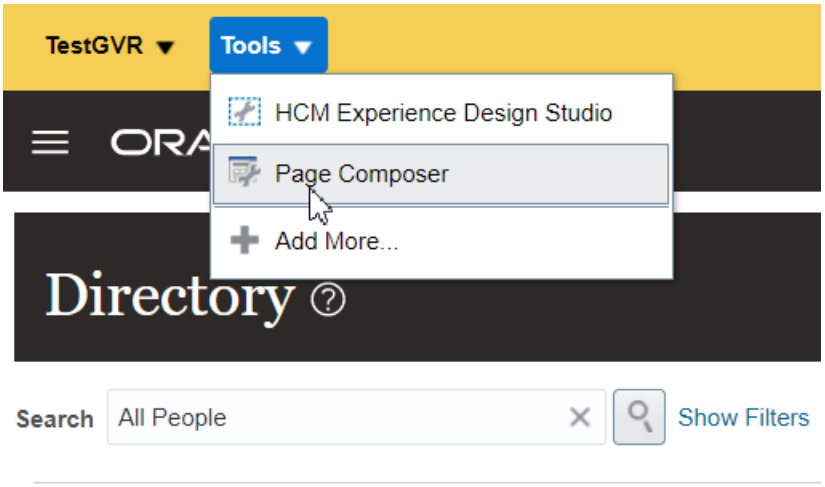
   You'll need to know the URL and the parameters when you embed the app using Application Composer or Page Composer.

Embed a Web App Using Page Composer

For Oracle Cloud Applications that you edit using Page Composer, you embed your web app by adding a Web Page component to a page and then specifying the app's URL and parameters. The following steps are high-level and are presented here to help you embed your web app in a page using Page Composer. For additional details, see Guideslines for Using Page Composer in *Configuring and Extending Applications*.

To embed an app using Page Composer:

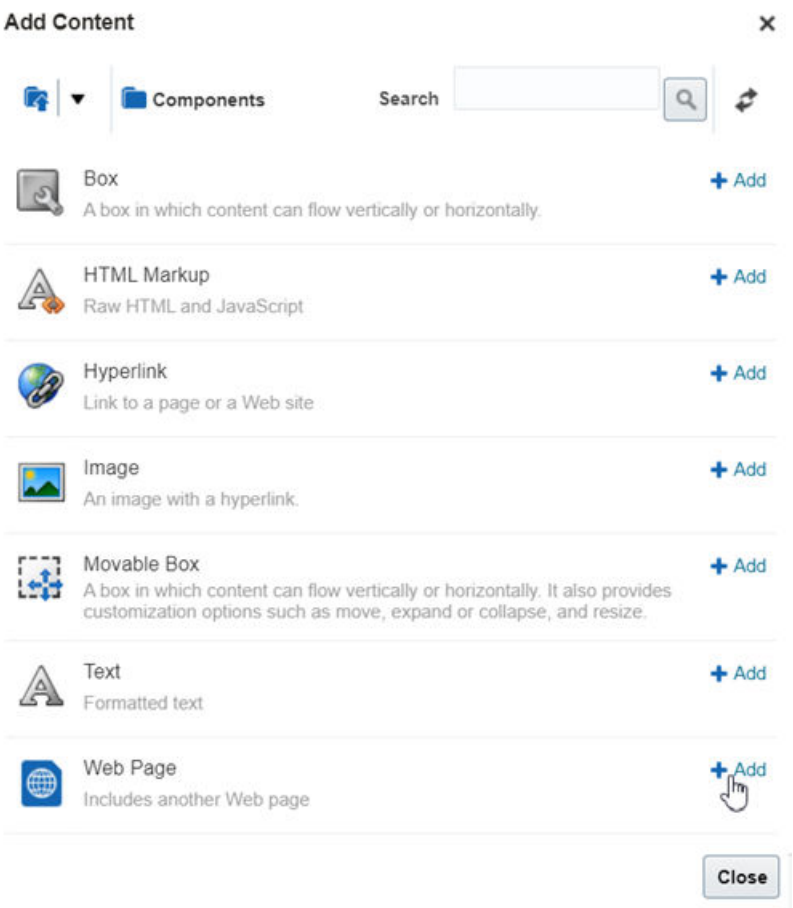
1. Log in to the Oracle Cloud Application where you want to embed your app.
2. Open a sandbox with Page Composer enabled, if you haven't already.
3. In your Oracle Cloud Application, navigate to the page where you want to embed your web app content.
4. Open the Tools menu and select **Page Composer**.



Page Composer will appear in the menu if you're in sandbox which is configured to be edited using Page Composer.

If you are using Application Composer, see the next topic.

5. In the Source view, open the Selector panel and select the area in the page where you want to embed your app. Click **Edit** in the popup menu to enter Edit mode.
6. Make sure the area is selected in the Selector panel, then click **Add** (+) in the panel's toolbar and add a Web Page component in the Add Content dialog. Click **Close**.

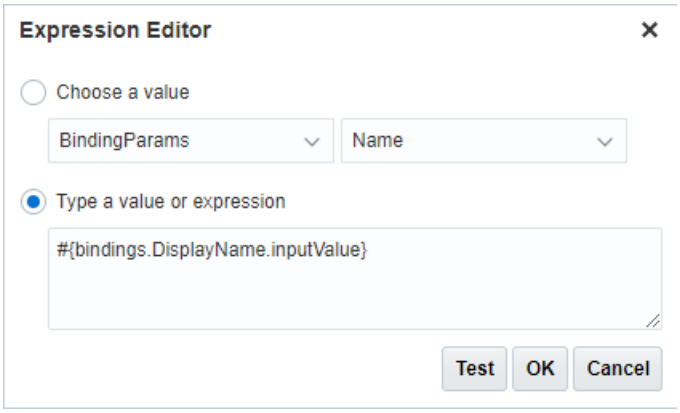


The area now contains the Web Page component.

7. In the Selector, right-click the new Web Page component and select **Edit** in the popup menu to open the Component Properties: Web Page dialog box.
8. Add a name and description for the new component.
9. In the Source field, open the dropdown list and select **Expression Builder**.

   If you are passing a variable to your web app as part of the URL, you'll need to use the Expression Editor to construct the URL.
   If you aren't going to pass a variable, you can paste the URL for your web app in the Source field.
10. In the Expression Editor dialog, select **Choose a Value**, and then select **Binding Parameter** in the dropdown list.
11. Select the binding parameter you want to use from the dropdown list.
12. Select **Type a value or expression**, then edit the expression in the text area to define the URL of your web app.

    The text area contains an expression fragment (<*generated-expression*>) generated for you based on the binding parameter you selected. It might look something like `#{bindings.DisplayName.inputValue}`. This is the value that will be passed to your web app.



13. *Prepend* the URL of your web app to this expression in the text area.

    Your expression might look something like `https://vbinstanceurl/.../appname/?VariableName=<generated-expression>`. When you paste in the URL, make sure it includes `?VariableName=` before the generated expression, where *VariableName* is the name your web app expects in the URL (for example, "OwnerName").
14. Click **OK** to close the Expression Editor dialog, then click **OK** to close the Component Properties dialog box.
15. Click **Done** to finish the Page Composer editing session.
16. Publish your sandbox after you've added the web page.

After closing the editing session, your web app content appears in the area in the page in the Oracle Cloud Application. The app is embedded in the page, and you can navigate to pages within the web app without leaving the Oracle Cloud Application page containing the web app. You can use your web app to display web app data in a page, and for example, to show a graph that rendered in the web app.

Embed a Web App Using Application Composer

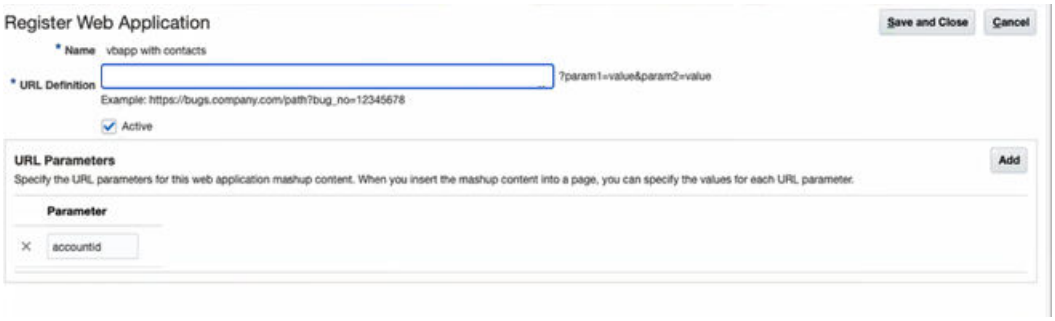
For Oracle Cloud Applications that you edit using Application Composer, you embed your web app by adding a *mashup* to the application, for example, in a tab or a page. The following steps are high-level and are presented here to help you embed your web app in a page using Application Composer. For a detailed description of how to embed a web app, see the Overview of Mashups section in *Configuring Applications Using Application Composer*.

To embed a web app using Application Composer:

1. Log in to the Oracle Cloud Application where you want to embed your app.
2. Open a sandbox with Application Composer enabled, if you haven't already, and then open Application Composer.
3. Create a **parameter-based mashup** for the web application.

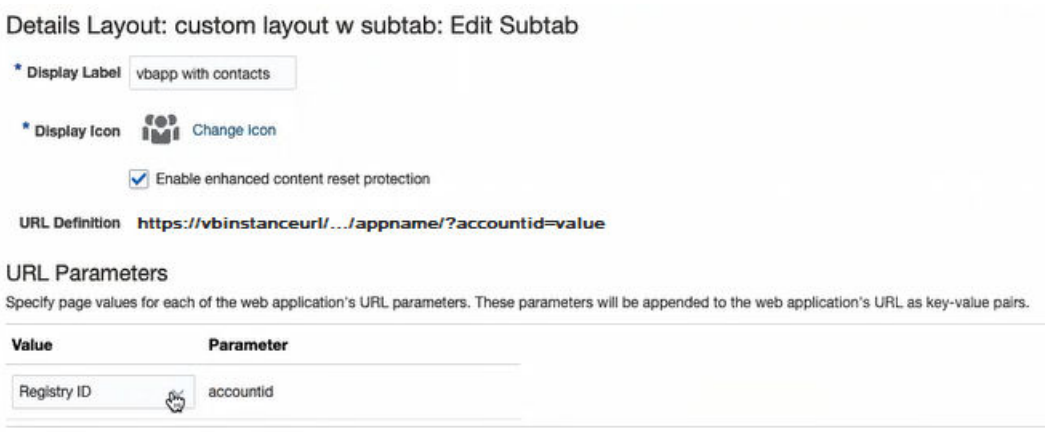
   If you want to pass any input parameters to your web app, you'll need to define the names of the parameters when you create the mashup. For example, if your web app is using an input variable named `accountid` (`https://vbinstanceurl/.../appname/?accountid=SomeValue`), you'll want to add `accountid` to the mashup.

   For more details on creating a parameter-based mashup, see Register Your Web Application in *Configuring Applications Using Application Composer*



4. In Application Composer, locate the application page (for example, the Details page) where you want to embed the web app.

   If you specified any parameters in your mashup, you'll need to map the parameters to fields in your application. For example, you would map the Registry ID field to the `accountid` parameter you defined in the mashup if you wanted to use the Registry ID value as your web app's input parameter. For more on adding a mashup to a page, see Embed a Registered Web Application into Your Application Page in *Configuring Applications Using Application Composer*.



5. Publish your sandbox after you've added the mashup to your application.

Call Server-side Functionality from Visual Builder

Visual Builder apps run in the client's browser. If you want to invoke external code residing on some other server, create a REST wrapper for that code and use Visual Builder to call REST services and invoke them.

This also applies to Node.js code on the server which is in JavaScript.

Add the Oracle Digital Assistant to Your Web App

You can integrate the Oracle Digital Assistant (ODA) into your Visual Builder web app simply by importing the Oracle Web SDK into your project and adding just a few lines of code.

Before you begin:

- Set up the Oracle Web SDK channel in your Digital Assistant (DA) instance and associate it with a skill. See Overview of the new Oracle Web SDK and its customization features in Oracle Digital Assistant 19.10 and later
  Record the following Oracle Web channel parameter settings: `URI` (your chat server URL) and `Channel ID` (the Web channel Id). You'll need to add these values to your code when you integrate the digital assistant.
- Download the Oracle Web SDK 1.0 from the Oracle Digital Assistant (ODA) and Oracle Mobile Cloud (OMC) Downloads page.

To import the Oracle Web SDK and add required code:

1. Import the Oracle Web SDK into your project:
   a. Extract the `oda-native-client-sdk-js-1.0.0.zip` file downloaded from the ODA download page.
   b. Compress `native-client-sdk-js` folder where the `web-sdk.js` file is saved.
   c. From your project, right-click the resources node in the navigation tree and click **Import**.
   d. Import the compressed `native-client-sdk-js.zip` and click **Import**.

      Your imported files appear in the resources branch.

2. Add code to a page of your web app to invoke the Oracle Web SDK to connect and invoke the web-widget.
   a. Select index from the navigation tree and select HTML (</>) to display the index page in HTML.
   b. Add the following code before closing the **</head>** tag, including your `URI` and `channelID` values where indicated:

```
<script src="resources/native-client-sdk-js/web-sdk.js"></script>
<script>
    var chatWidgetSettings = {
        URI: 'YOUR_URI',
        channelId: 'YOUR_CHANNELID'
  };
        setTimeout(() => {
            window.Bots = new WebSDK(chatWidgetSettings);
            Bots.connect().then(() => {
                console.log("Connection Successful");
            }, (reason) => {
                console.log("Connection failed");
                console.log(reason);
            });
        }, 2000);
</script>
```

   Line 1 points to the `web-sdk.js file` stored in the `resources/native-client-sdk-js` folder.

   Line 3-6 sets the `URI` and `channelID` which are passed as a parameter.

   Line 7 calls the functionality after 2 seconds. This is done to ensure that the page gets loaded.

   Line 8 initializes the library with the configuration.

   Line 9 establishes the connection with the server.

   Line 15 defines the delay between the rendering of a page and the display of the messenger icon or widget. The default setting is 2 seconds.

3. Run the project to test the integration:
   a. Click **Run** to start the web application in a separate tab.
   b. Click the Chat widget icon to start your chat bot.

Abort Pending REST Calls in Visual Builder

When a REST call to your application takes too long, you might want to let your users cancel the call midway. You do this by adding an `AbortController`, a browser-based interface that lets you abort a web request.

Here's a sample scenario that shows how you can use an `AbortController` in a Visual Builder application. For demo purposes, assume the app has a page with two buttons: a **Call REST** button and a **Cancel REST** button.

- When users click the **Call REST** button, an `ojAction` event triggers an action chain to call the `Get/Employee/{Employee_Id}` endpoint and displays a notification to the user.

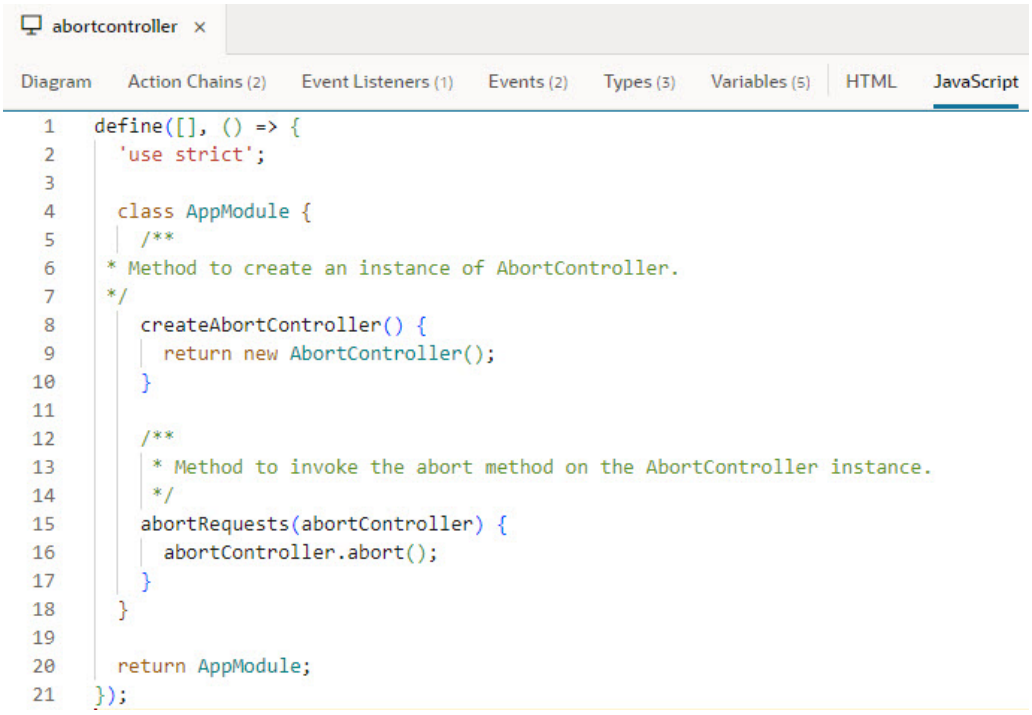
- When users click the **Cancel REST** button, another `ojAction` event triggers an action chain to abort the REST request and display a notification to the user.

> **Note:**
>
> The `AbortController` API is *not* supported for Visual Builder applications that are enabled as PWAs or configured for offline capabilities.

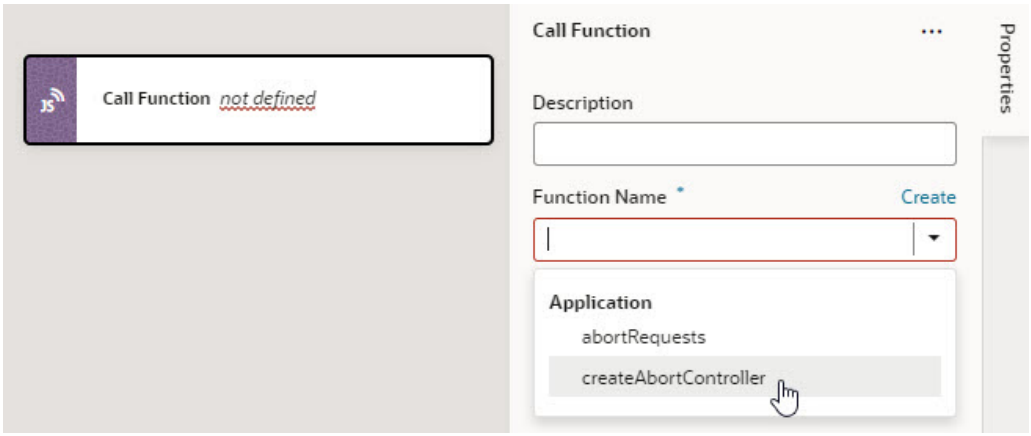
1. To use the `AbortController` API, you first need to create an `AbortController` instance. You also need to call the `abort` method on the `AbortController` instance that's created. We'll do this by adding two app-level JavaScript functions that can be used across your application's pages.
   a. Select your application node, then click **JavaScript** to open the app-level JavaScript editor.
   b. Add this JavaScript snippet to the editor:

```
/**
 * Method to create an instance of AbortController.
 */
createAbortController() {
  return new AbortController();
}

/**
 * Method to invoke the abort method on the AbortController instance.
 */
abortRequests(abortController) {
  abortController.abort();
}
```

   Your JS editor might look something like this:

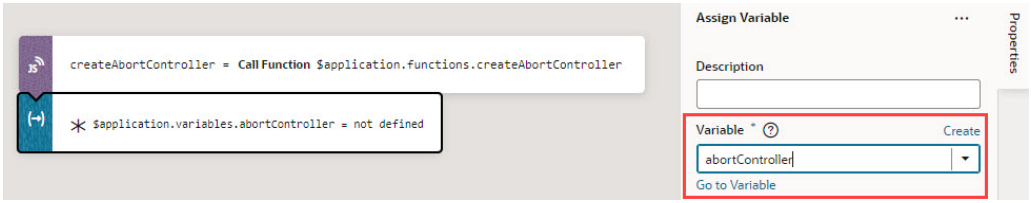


2. Create a variable to track the `AbortController` instance that will be created.
   a. Click **Variables** to open the app-level Variables editor.
   b. Click **+ Variable** and create a variable with ID `abortController` (for example) and type **Any**.
3. To initialize the `abortController` variable when the app loads, build an action chain that's triggered in response to a `vbEnter` event for the application.
   a. Click **Event Listeners** at the app level.
   b. Click **+ Event Listener**.
   c. Select **vbEnter** under Lifecycle events and click **Next**.
   d. Select **Create Application Action Chain** to create a new app-level action chain. Click **Finish**.
   e. Click **Go to Action Chain** next to the newly created `vbEnterListener` action chain to open the Action Chains editor.
   f. From the Actions palette, drag a Call Function action onto the canvas.
   g. In the action's **Function Name** property, select `createAbortController` under Application to call the JS function that creates an `AbortController` instance.



h. Now drag and drop an Assign Variable action to follow the Call Function action.

i. In the Assign Variable action's Properties pane, select `abortController` under Application from the **Variable** list.



j. Hover over the **Value** property and click ***(x)*** to open the variables picker, then select `createAbortController` under Local to assign the value returned by the `createAbortController` function to your `abortController` variable.

4. Associate the `AbortController` with the REST call you want to abort. You do this by attaching the `AbortSignal` of an `AbortController` instance to the REST request via the `signal` option. For example, to abort the `Get/Employee/{Employee_Id}` endpoint request, you attach the `AbortSignal` to the Call REST action in the action chain underlying the button component (which is `ButtonActionChain` in our example).

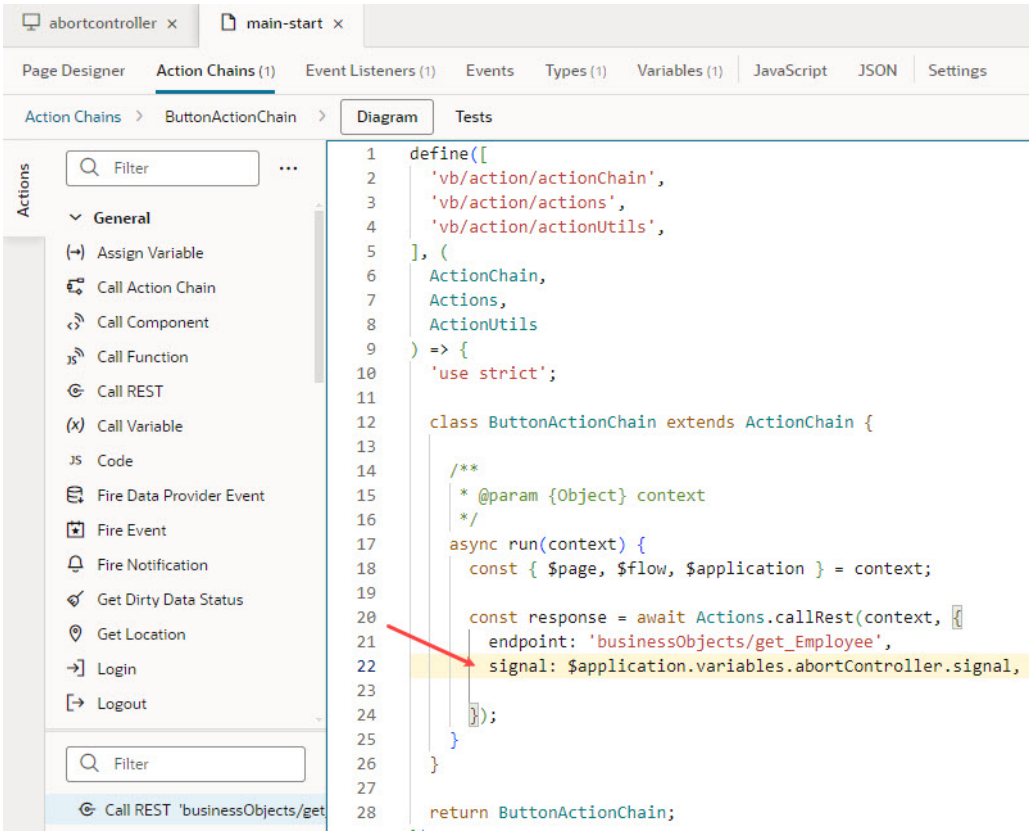
   a. Open the action chain that uses the Call REST action in the Action Chains editor. Because our example assumes a Call REST button on a page, this Call REST action exists in the `ButtonActionChain` action chain defined at the page level.

   b. Click **Code** to switch to code view.

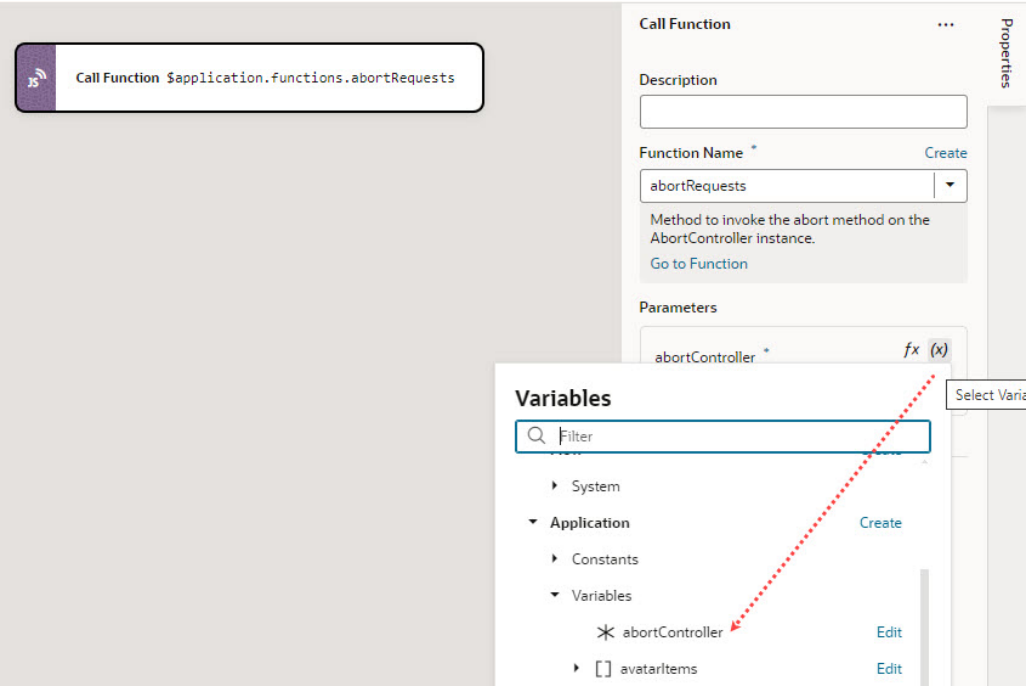
   c. Locate the code snippet for the particular Call REST action and add this line after the `endpoint` constructor:

   ```
   signal: $application.variables.abortController.signal,
   ```

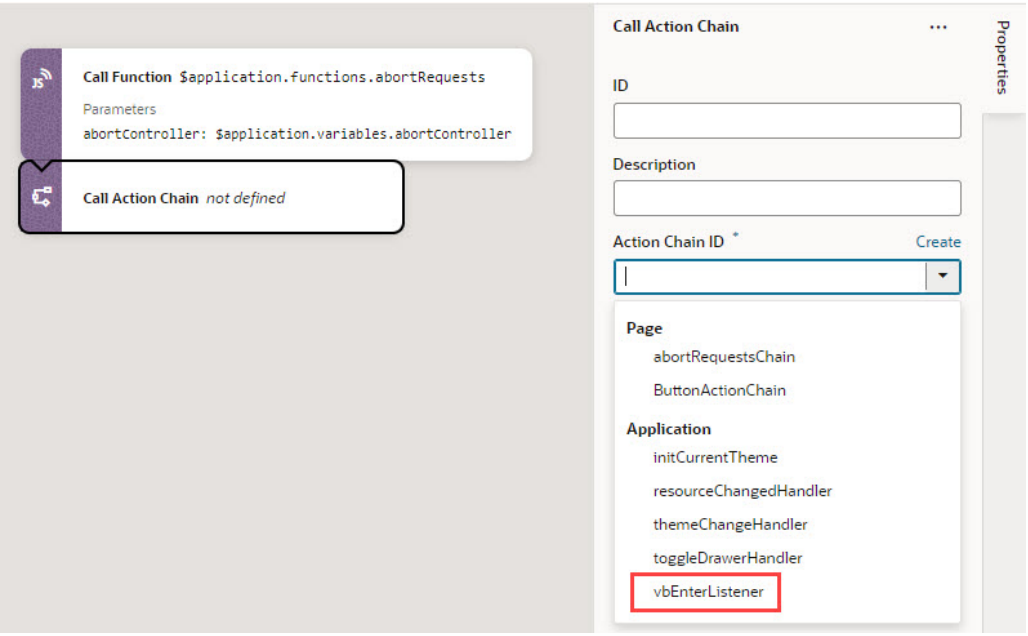
   In this example, the `AbortSignal` is accessed via `$application.variables.abortController.signal` and attached to the `get_Employee` REST request:



5. To abort REST requests with the `AbortSignal` attached, build an action chain to call the `abort` method on the `AbortController` instance. Because our example assumes a Cancel button that users click to abort the `Get/Employee/{Employee_Id}` request on a page, we define the abort action chain at the page level (but you can also define it at the app level).
   a. Click **Action Chains** to open the Action Chains editor.
   b. Click **+ Action Chain** and create a new action chain with `abortRequestsChain` as its ID.
   c. From the Actions palette, drag a Call Function action onto the canvas.
   d. In the action's **Function Name** property, select `abortRequests` to specify the JS function that calls the `abort` method on the `AbortController` instance for the specific REST request.
   e. To pass your `abortController` variable as an input parameter to the `abortRequests` function, locate the `abortController` under Parameters, hover over the parameter to open the variables picker, then select `abortController` under Application.



f. Calling the `abort` method on an `AbortController` instance permanently aborts any future requests, so you need to create a new `AbortController` instance and assign it to the `abortController` variable. To do this, drag and drop a Call Action Chain action and select the `vbEnterListener` action chain (which was created for you in step 3 to do both) in the **Action Chain ID** list.



g. Add other actions as needed to handle the Cancel operation. For example, you might want to add actions to notify the user and reset anything that's required for your app's typical flow.

Forms

Common use cases relating to forms:

- Enable Client-Side Validation for a Form
- Validate Dates in Forms

Enable Client-Side Validation for a Form

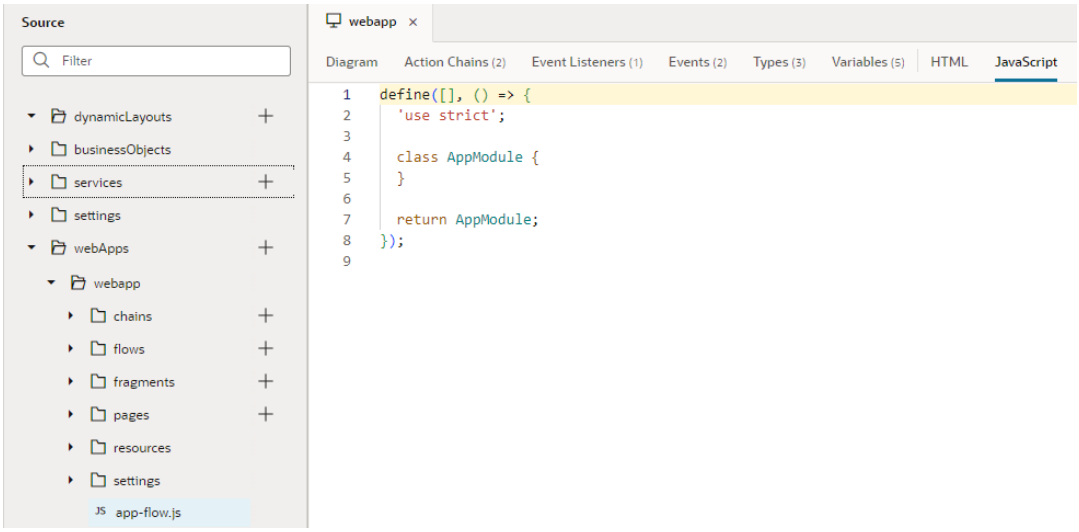
You may want to set up a form so that your application can check the validity of its contents before the user submits it.

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. If it isn't already, 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 **Source View** in the Navigator, then find your application's `app-flow.js` file:



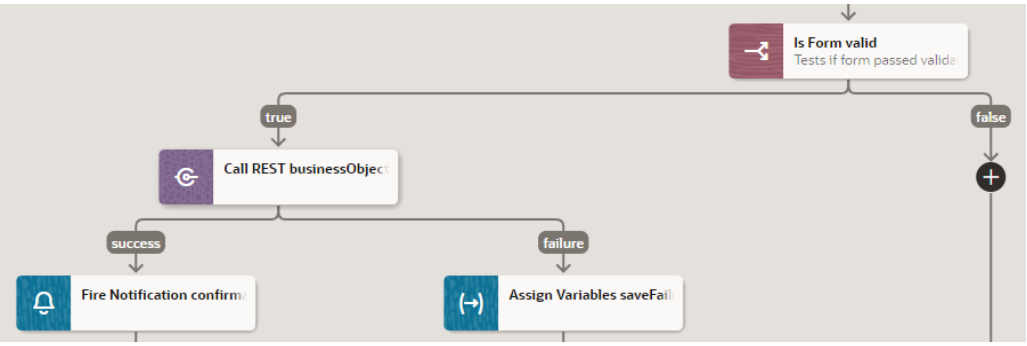
5. Add the `isFormValid` function as shown here (the function code appears in bold):

```
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;
    }
};

return AppModule;
```

6. Go back to the page with the form. Click the **Save** button, then select the **Action Chains** tab for the button and click **createExpenseReportChain**.
7. If an **If** action does not exist:
   a. Drag an **If** action after **Start**.
   b. In the **Condition** field, enter:

      ```
      {{ $application.functions.isFormValid("CreateForm") }}
      ```

      The argument to the `isFormValid` function is the `id` value for the `oj-validation-group` element.
   c. Move the **Call REST businessObjects/...** node to the **true** branch of the **If** action. For example:



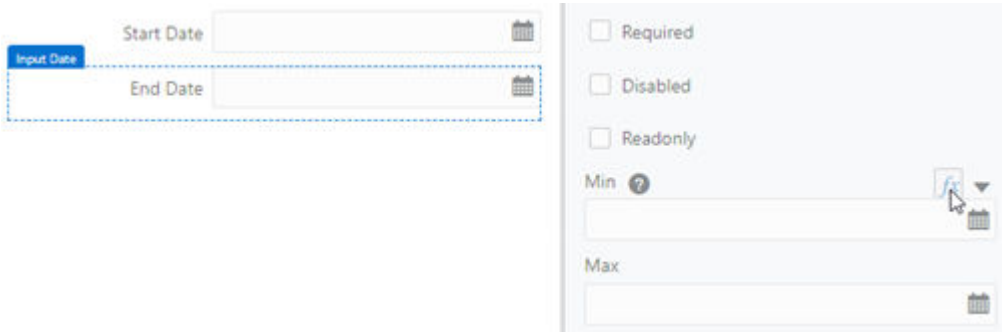
You can now test the form validation.

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 Properties pane, click the **fx** icon for the 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 Format a Date Field for information on how to format a date field of a business object.

Tables

Common use cases relating to tables. It's worth noting that UI guidelines recommend lists, instead of tables, to display data records for complex data sets. While tables tend to be more

utilitarian, lists are ideal for small-screen displays and are increasingly common in web as well as mobile applications.

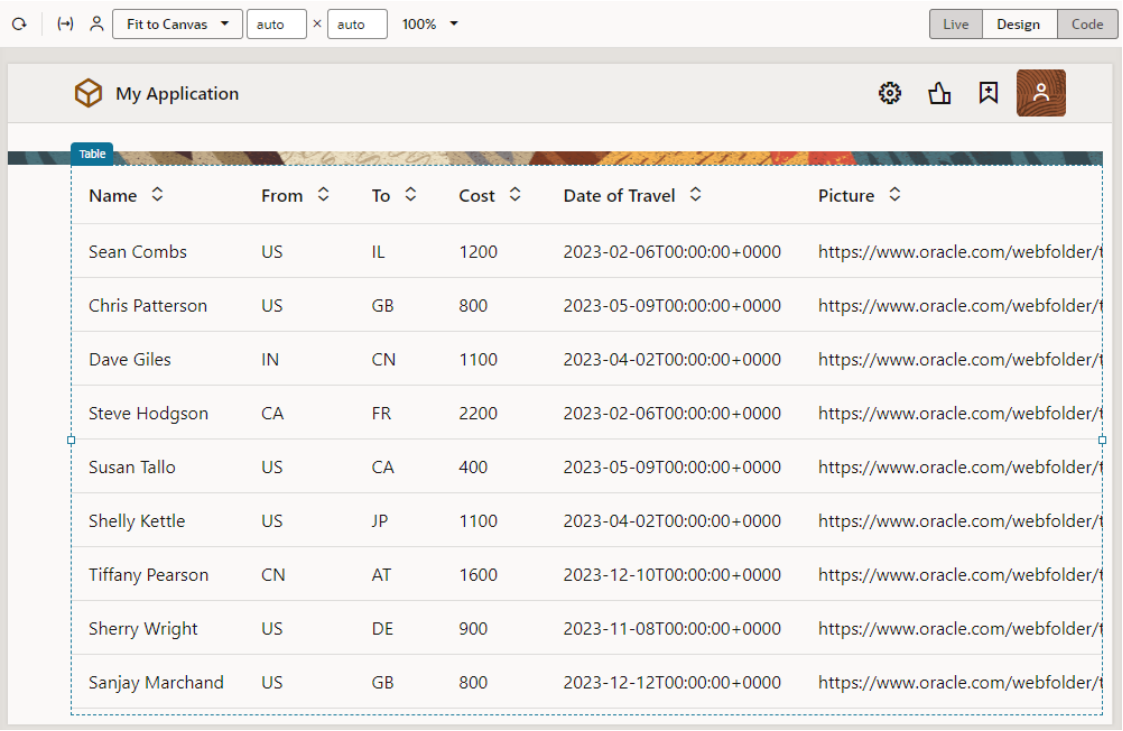
If lists don't provide the functionality you need, here are some advanced techniques you can use in table components:

- Modify a Table's Default Display
- Reorder Columns in a Table
- Sort Data in Table Columns
- Enable Resizing of a Table Column
- Wrap Table Text
- Add Columns to an Existing Table
- Format Row Values in a Table Conditionally
- Create a Search Filter for a Table
- Create an Editable Table
- Update Pagination Behavior for a Table
- Enable Text Selection in a Table

Modify a Table's Default Display

You can modify the default look and feel of tables to provide a more visually appealing display in your application's pages.

Here's how a table mapped to data in a business object (done using the Add Data quick start) displays by default:

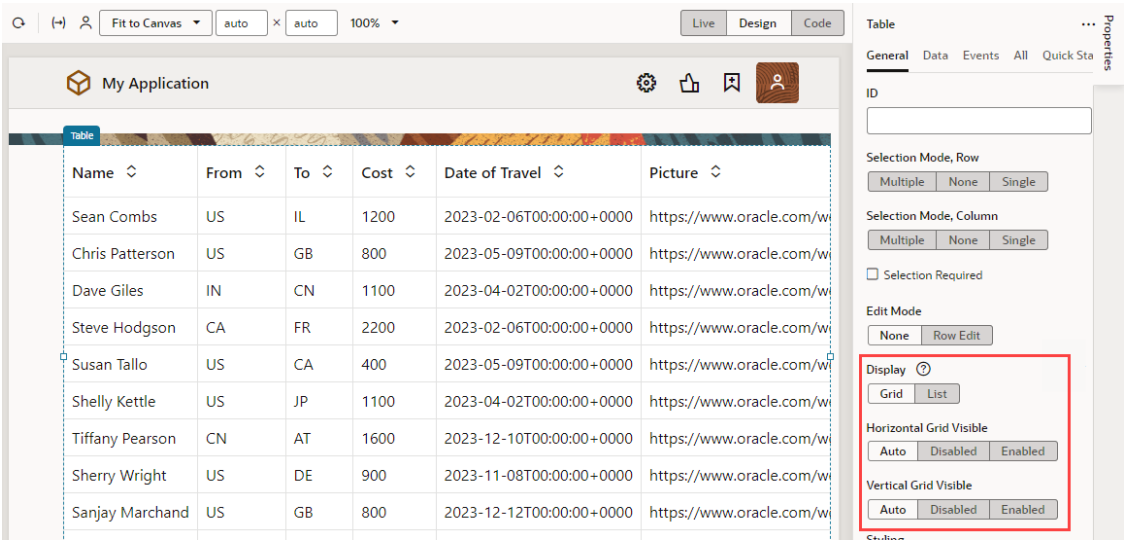


Display table as a grid

By default, tables display as a list. To switch the list view to a more compact grid view:

1. Select the table on the page to view its properties.
2. Select **Grid** under **Display** in the Properties pane's General tab.
3. If you want to hide the horizontal and vertical gridlines, select **Enabled** for the **Horizontal Grid Visible** and **Vertical Grid Visible** properties.

Here's how the table looks in grid mode (with the horizontal and vertical gridlines still visible):



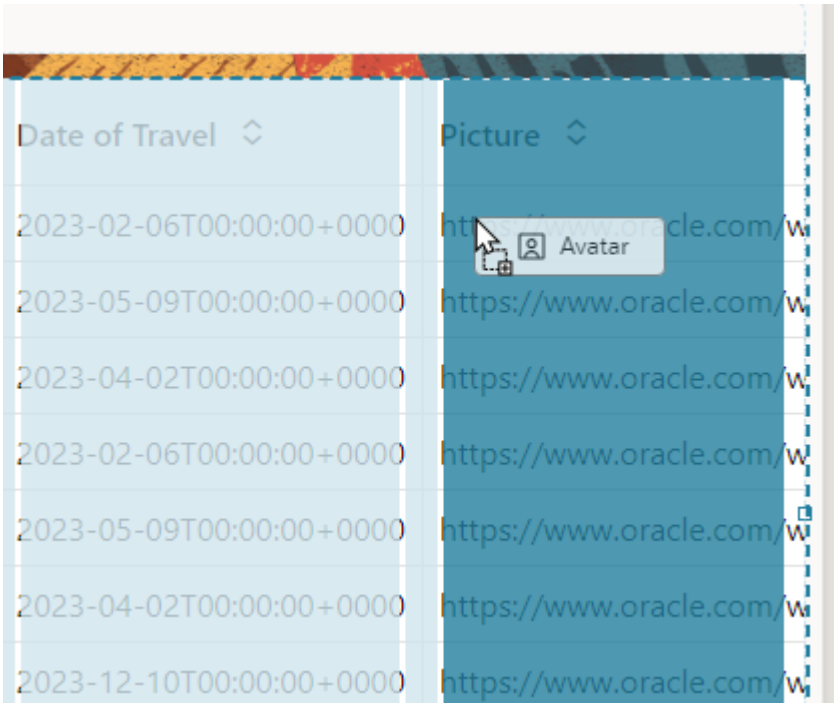
Reformat content in table columns

It's possible to reformat the content of your data columns to provide a cleaner and more visual display. For example, you might want to show images or visualize data using gauges.

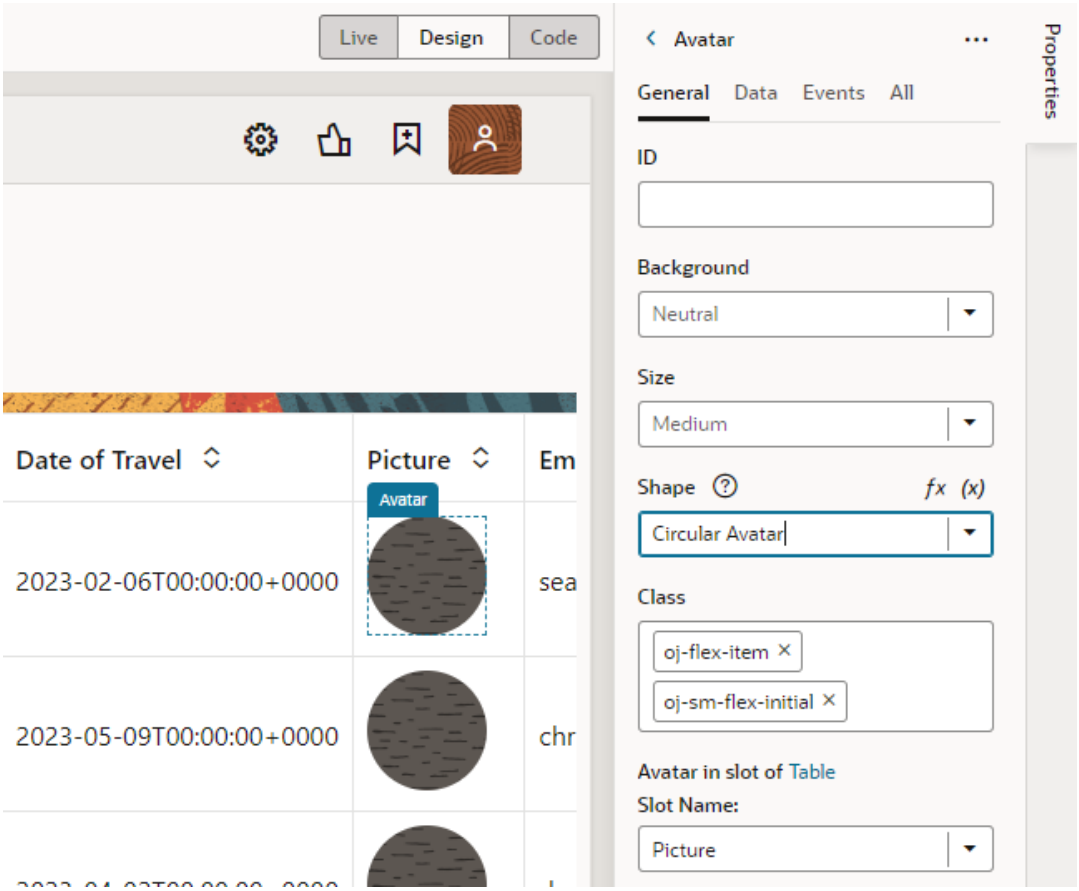
Display images in a column

To display images in a table column:

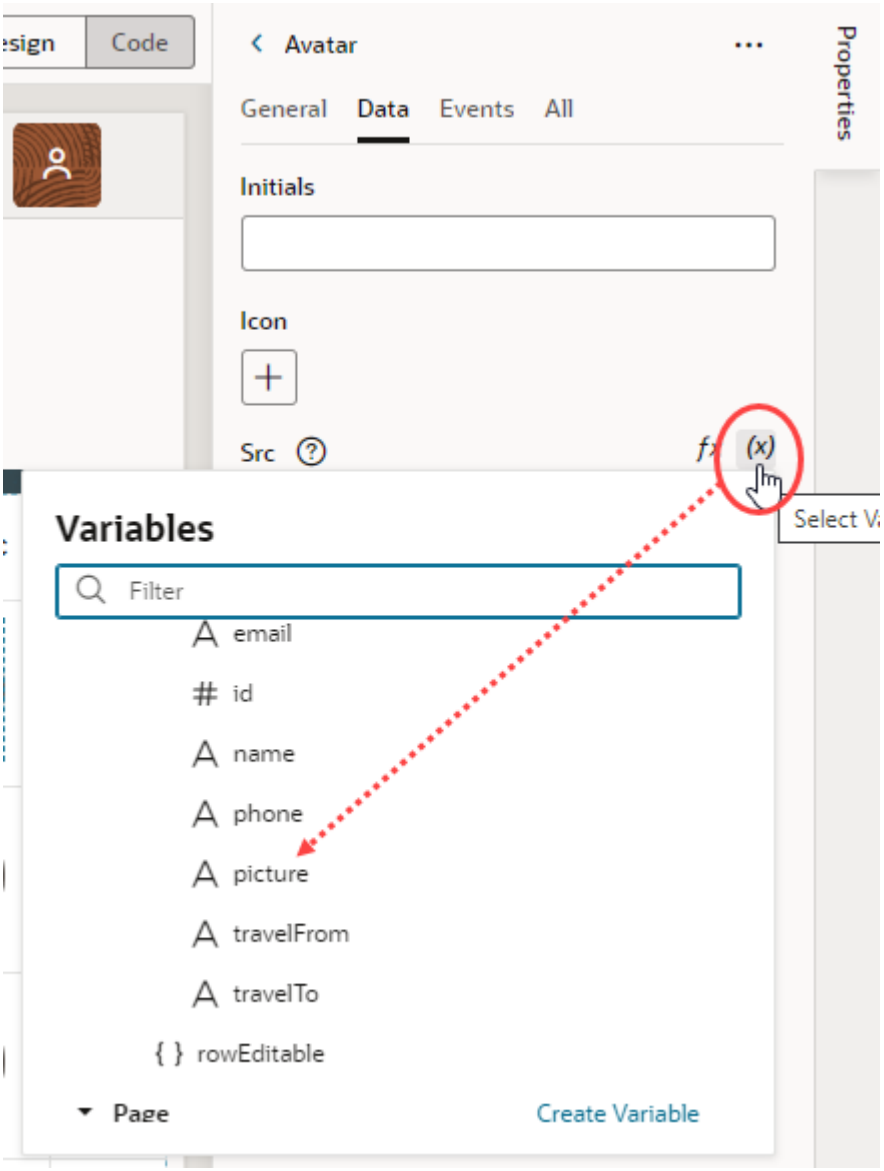
1. Drag an avatar component from the Components palette and drop it directly onto any row in a particular column on the table (Picture, in our example):



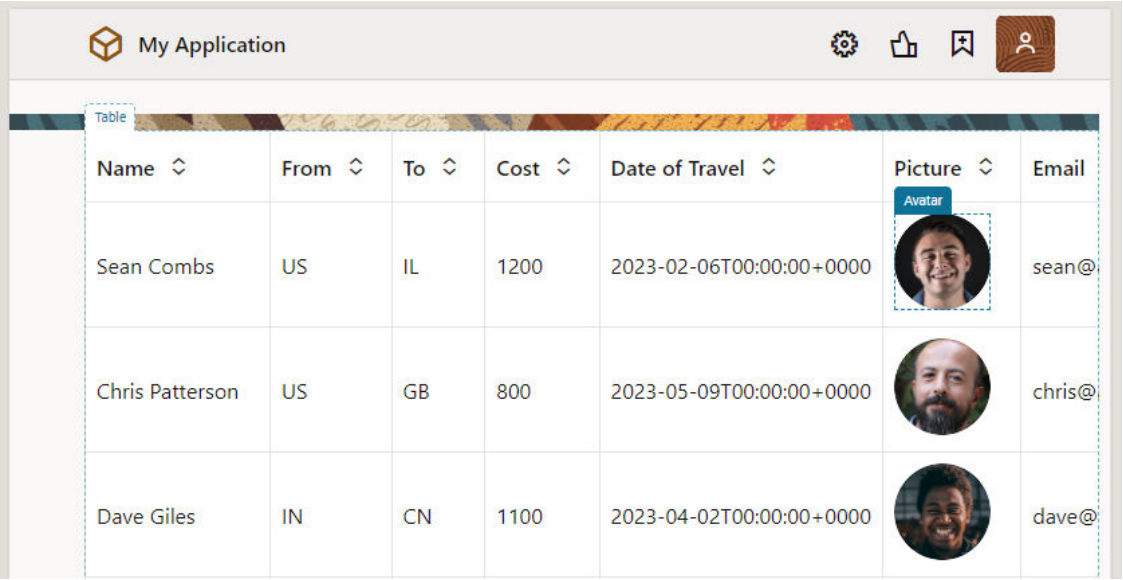
2. Update the avatar component's properties as desired, for example, you can change its size and shape as well as give it a background:



3. Switch to the **Data** tab in the Properties pane. Hover over the **Src** field and click ***(x)*** to open the variables picker. Select **picture** under **Current** and **Row** to bind the avatar to the value of that row in the column:



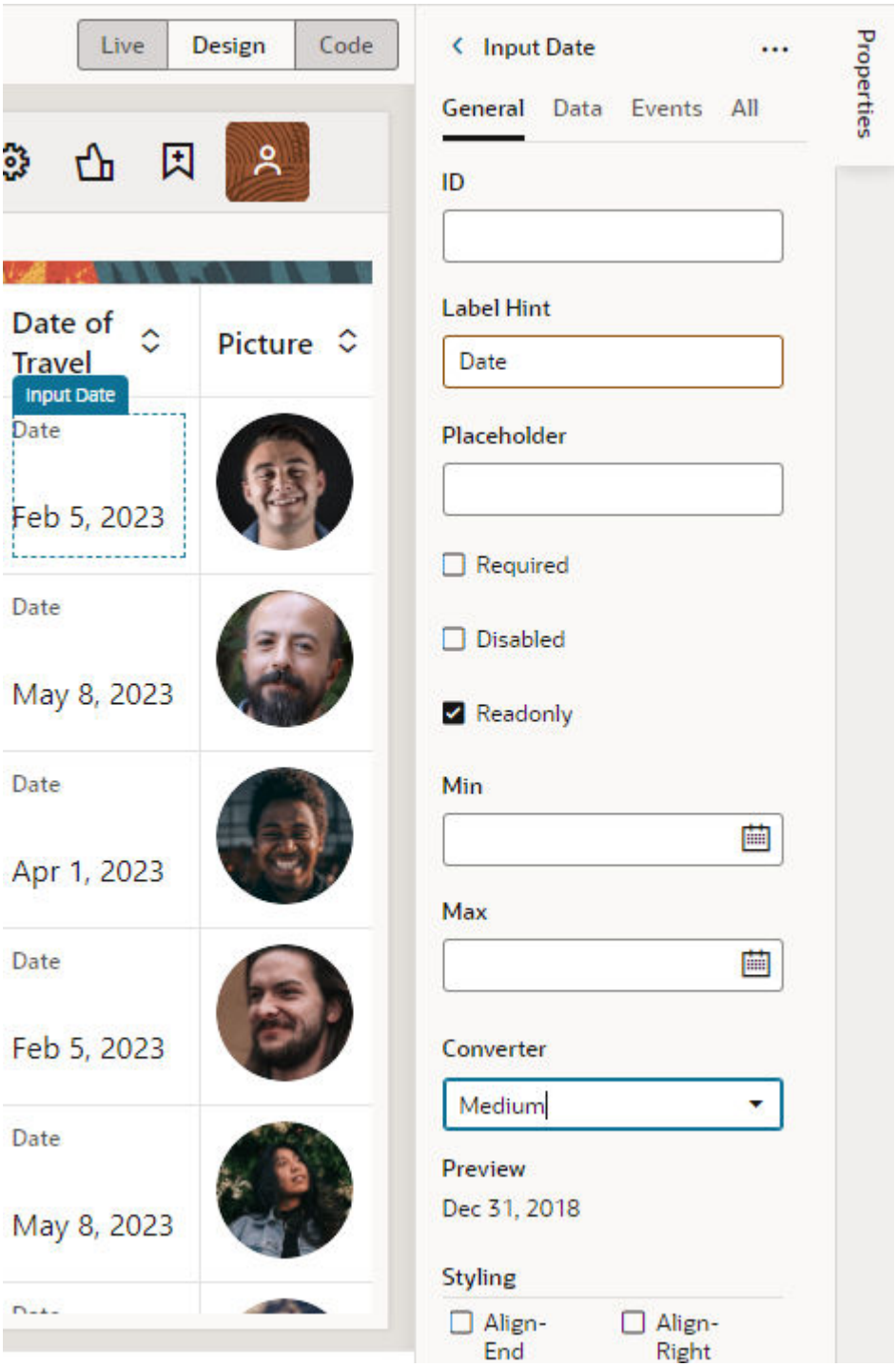
Your data now shows as images:



Reformat a date column

To change the format used by a date column in a table:

1. Drag an input date component from the Components palette and drop it directly onto a date-based column (Date of Travel, in our example). The component automatically picks up the existing value.
2. Update the component's properties, for example, you can use the Converter property to set the format of the date:



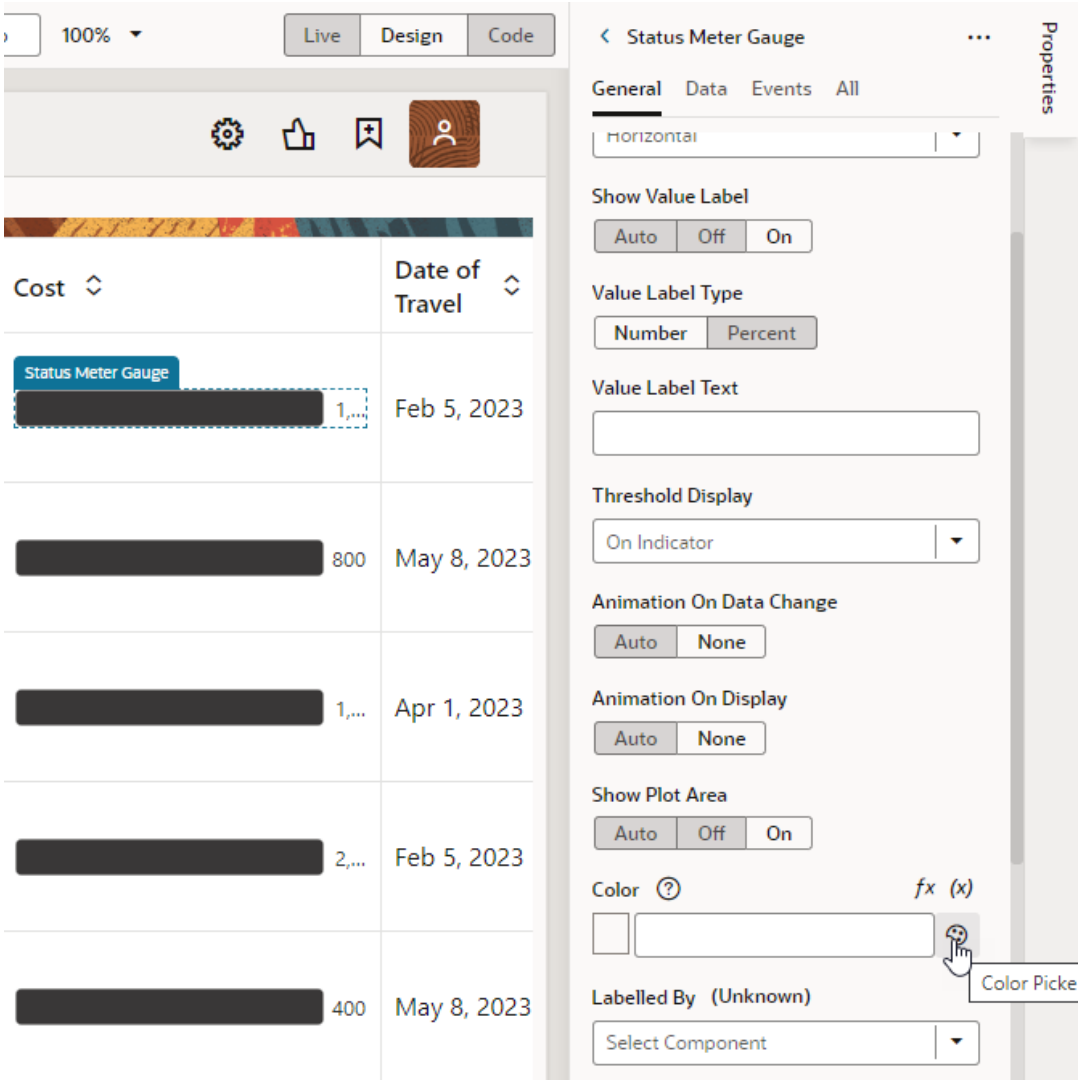
You might also want to clear the component's **Label Hint** field, so `Date` does not appear in the column.

Visualize data in a column

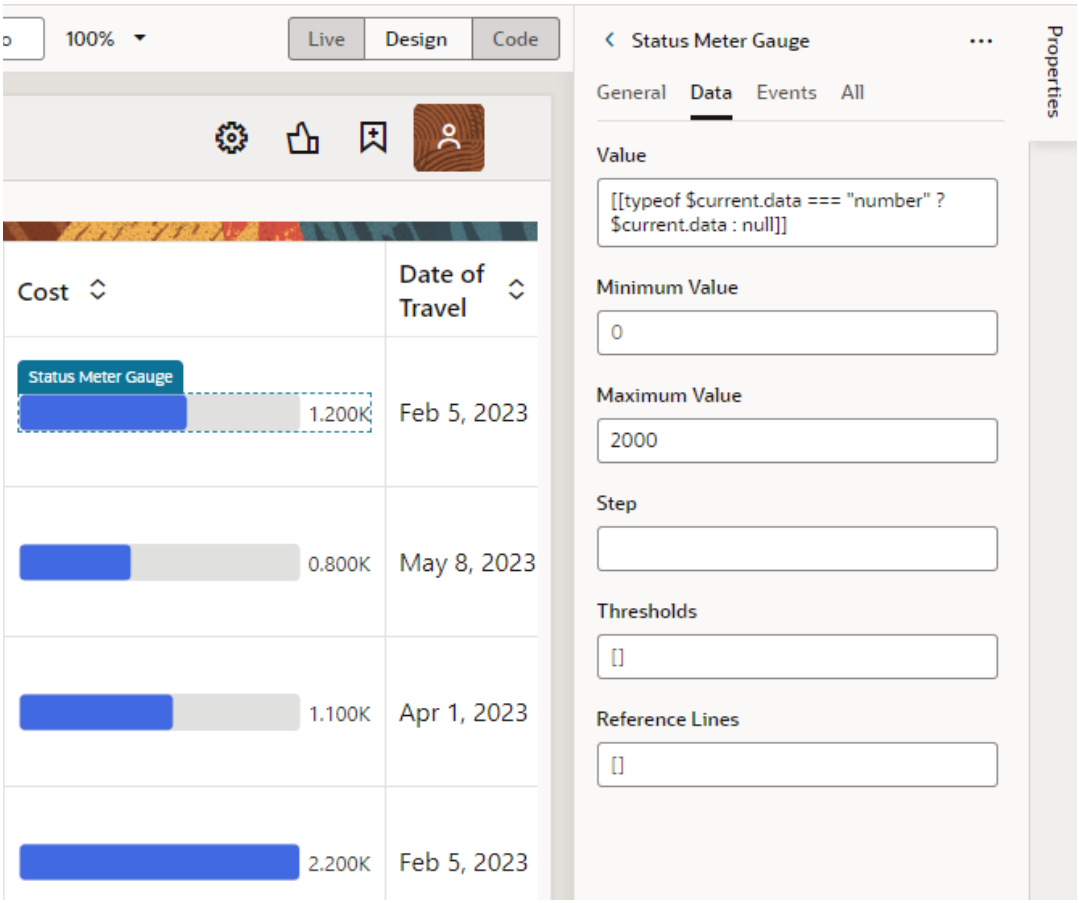
To visually display data in a table column:

1. Drag a visualization component (for example, the Linear Status Meter under Gauges) and drop it directly onto a particular column in the table (Cost, in our example). The component automatically picks up the existing value.

2. Update the component's properties as desired, for example, you might change the color of the gauge:



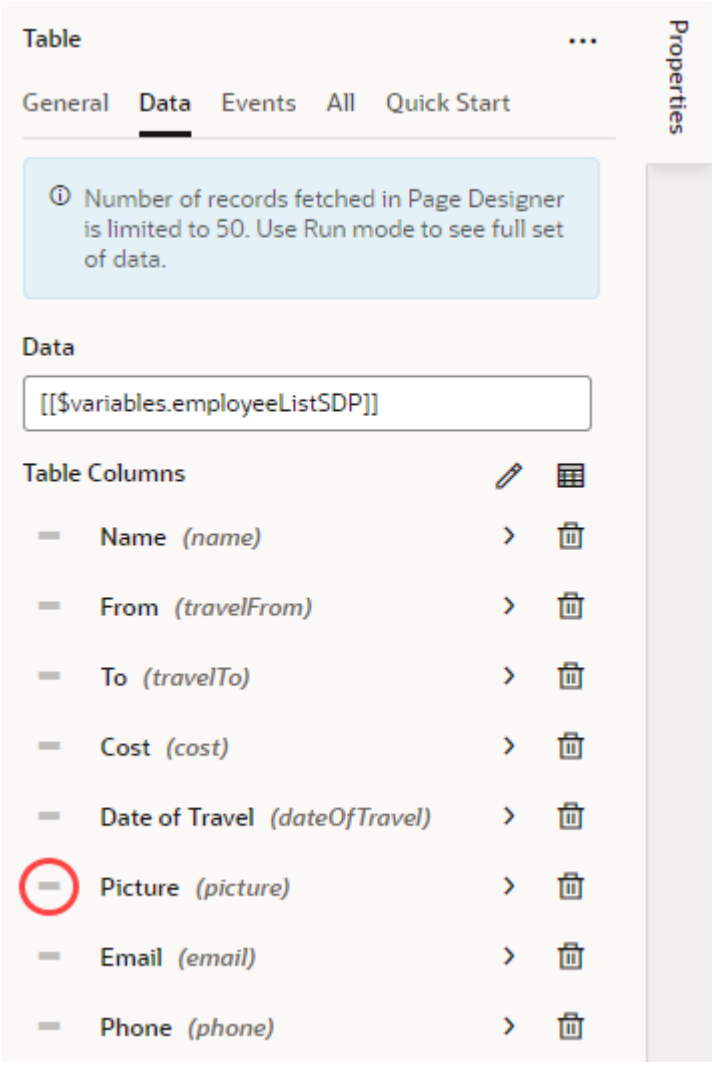
3. If your data falls outside the default data range for the gauge component, switch to the **Data** tab and change the **Maximum Value**, so your data renders in a more meaningful way:



Reorder Columns in a Table

When you use the Add Data quick start to map data to your table, you can specify the order in which you want columns to display in the table. If you want to change this order after the table is set up, you can do it from the table's **Data** tab in the Properties pane.

1. Select the table on the page to view its properties.
2. Click the **Data** tab.
3. Under Table Columns, drag a column's handle ( ) and drop it where you want the column to be:

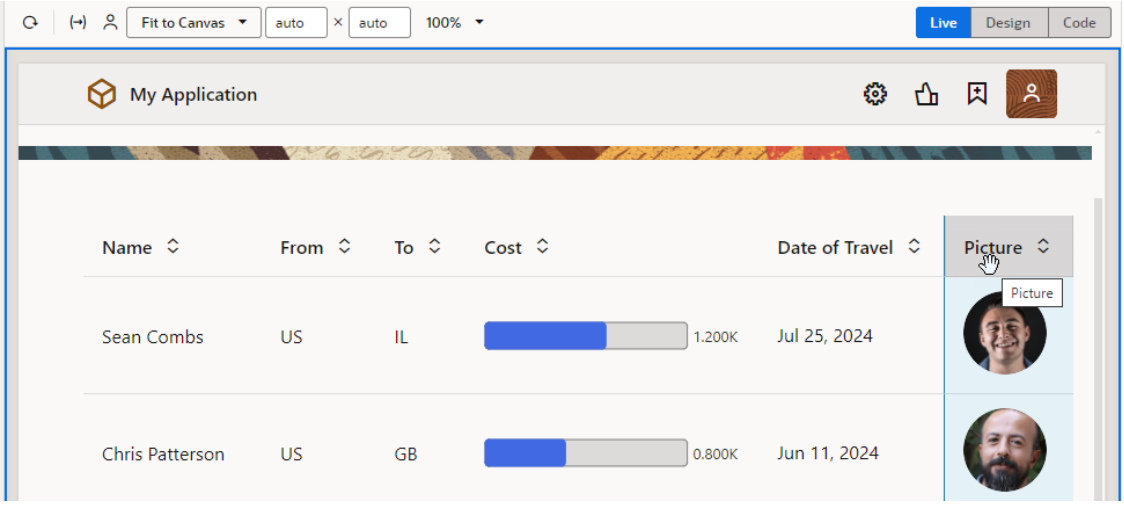


Enable drag-and-drop reordering

You have the option to enable drag-and-drop functionality to reorder table columns. To do this:

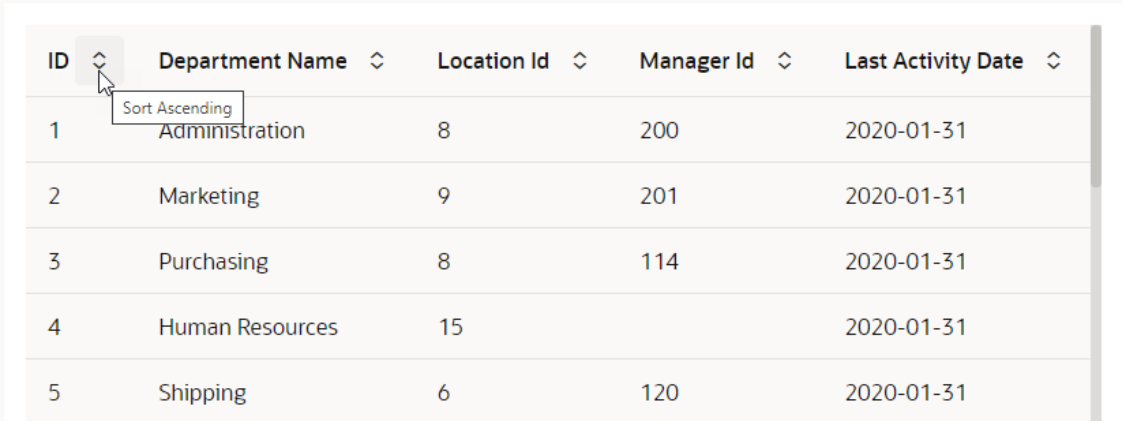
1. Select the table on the page to view its properties.
2. Set **Selection Mode, Column** in the General tab to either **Single** or **Multiple**, allowing either single or multiple columns to be selected.
3. Switch to the **All** tab.
4. Hover over the **DnD** property, then select **Show sub-properties** ( ).
5. Hover over the **Reorder** property and select **Show sub-properties** ( ).
6. Select **Enabled** from the **Columns** list.

With this setting enabled, you should be able to select a column in **Live** mode, then drag the move icon that appears in the column header and drop it where you want it:



Sort Data in Table Columns

Sorting is enabled by default on all table columns. Simply use the up and down icons on the table headers in Live mode to arrange the column's data in either ascending or descending order:

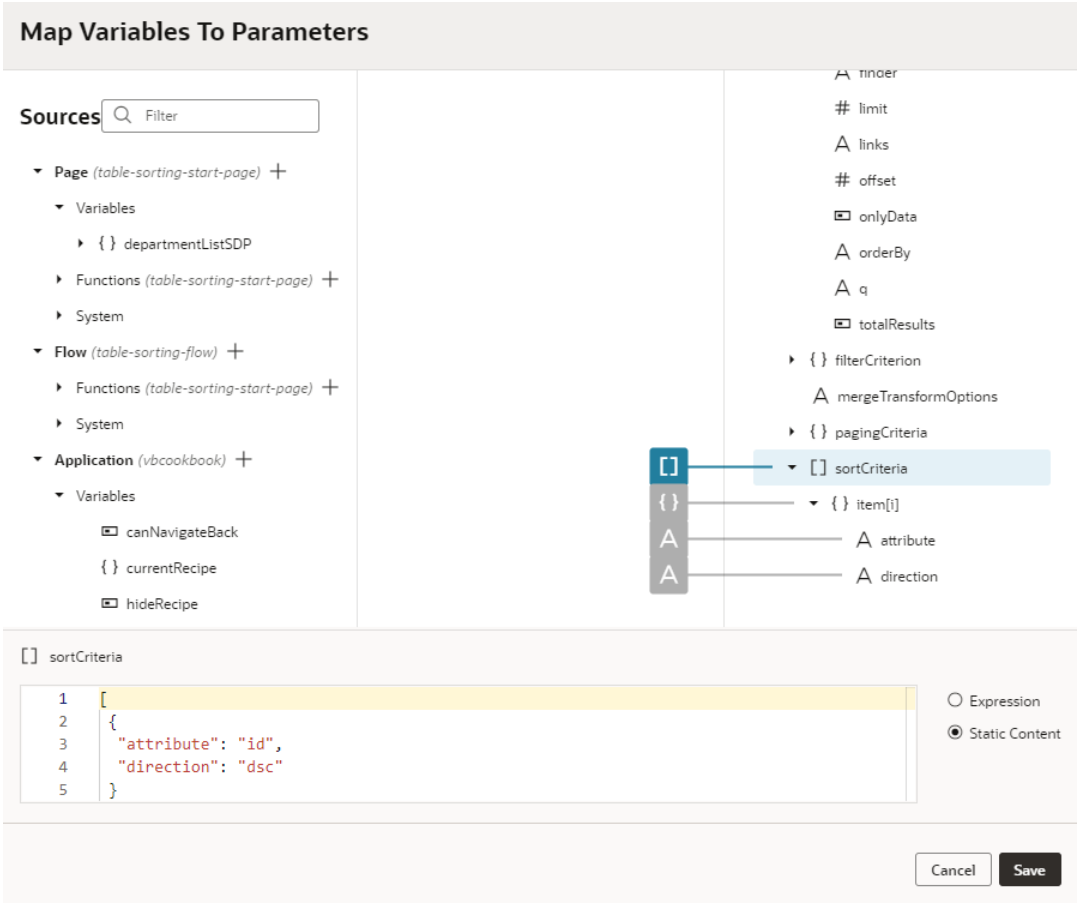


Sort table data by a particular column

If you want your table to always sort data based on a particular column (for example, ID), you can use the `sortCriteria` parameter in the Service Data Provider (SDP) used to retrieve the table's data.

1. In the **Variables** tab, locate the SDP variable underlying the table, for example, `departmentListSDP`.
2. In the variable's Properties pane, select **sortCriteria** under Parameters.
3. On the Target pane of the Map Variables to Parameters dialog, expand **sortCriteria** and **item[i]** to view the **attribute** and **direction** options.
   a. Select **attribute**, then enter the name of the field you want to sort in the expression builder; for example, `id`.

b. Select **direction**, then enter the direction of the sort, either `asc` for ascending or `dsc` for descending. If you don't specify a value, the default direction (`asc`) is used for sorting.

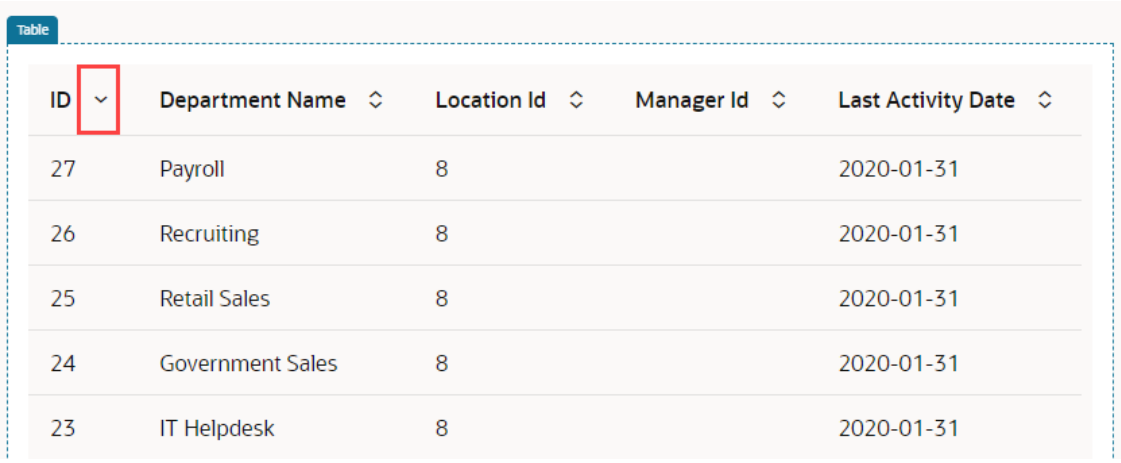


You can add several sort criteria in the array to define sorting by multiple columns. For example, to sort by location first and then by department, you'd use:

```
[
 {
  "attribute": "location",
  "direction": "dsc"
 },
 {
  "attribute": "departmentName",
  "direction": "asc"
 }
]
```

4. Click **Save**.

With `sortCriteria` specified on the ID column in the descending direction, your table sorts data by default on the ID column in the descending order, as shown here:



Disable sorting on a column

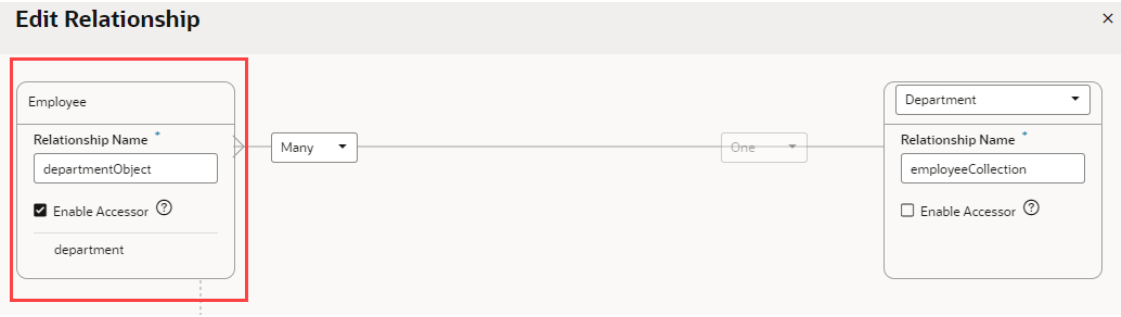
If you want to remove sorting capabilities on a particular column:

1. Select the table on the page to view its properties.
2. Click the **Data** tab.
3. Under Table Columns, click **Column Detail** ( ) for the column on which you want to disable sorting.
4. Set **Columns, Sortable** to **Disabled**.

Sorting in Parent-Child Relationship Tables

When working with tables in a parent-child hierarchy, sorting will work as expected on the parent table's columns. To sort records on the child table's columns, some additional steps are required.

Let's say you have a table that displays data for departments and the department's employees in a Departments table (parent table). When end-users select a department in the Department table, the selection determines the list of employees that appears in the Employee table (child table). To sort records by the child table's columns, the table must be based on the child objects and fetch parent details for each child (employees, in our example). The accessor in the relationship between the Department and Employee business objects must also be enabled. In the case of employees in a department, this would be the accessor on the Employee business object that allows access to Department information:



See Edit a Business Object Relationship for steps on editing the relationship to enable the accessor.

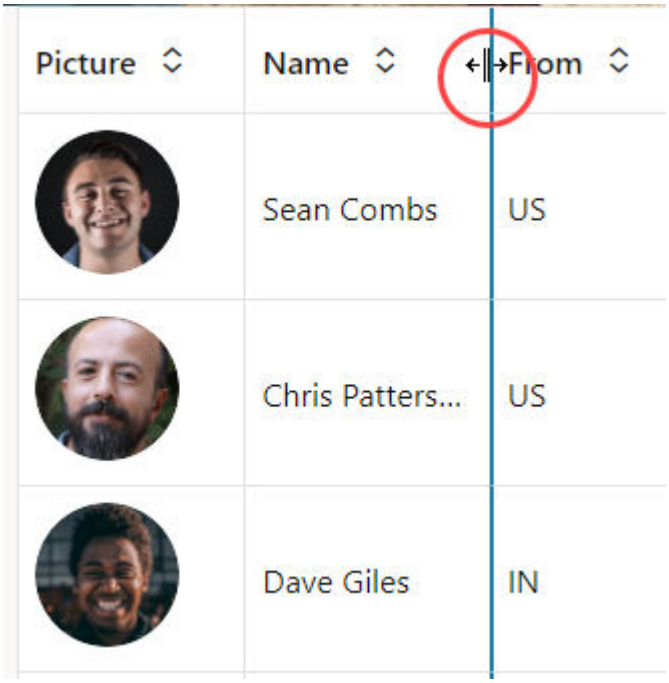
Enable Resizing of a Table Column

You can enable columns in a table to be resized, functionality that is particularly useful if a column's values are too long.

1. Select the table on the page to view its properties.
2. Click the **Data** tab.
3. Click **View Detail** ( ) next to Table Columns to view the table's column definition.
4. Locate the column that you want to be resizable. For example, when you want the **Name** column to be resizable, add `"resizable": "enabled"` to the column definition:

```
{
  "headerText": "Name",
  "field": "name",
  "resizable": "enabled"
},
```

If you now switch to Live mode, you should be able to resize the Name column:



If your data is cut off, you might want to enable text wrapping. See Wrap Table Text.

Wrap Table Text

If the header text for table columns doesn't display completely when a column's width is reduced, you can enable text wrapping for the header. To do this:

1. Select the table on the page to view its properties.
2. Click the **Data** tab.

3. Under Table Columns, find the column whose header text you want to wrap and click **Column Detail** ( ).
4. Add `white-space:normal; word-wrap:break-word;` to the **Columns, Style** property.

You can also update the column definition in a page's Code view. Update the `style` attribute as shown here on the column definition to enable wrapping if the header does not fit:

```
"white-space:normal; word-wrap:break-word;"
```

You can enable text wrapping for the content in table columns as well. To do this:

1. Under Table Columns in the table's Data tab, find the column whose content you want to wrap and click **Column Detail** ( ).
2. Add `white-space:normal` to the **Columns, Style** property.

Add Columns to an Existing Table

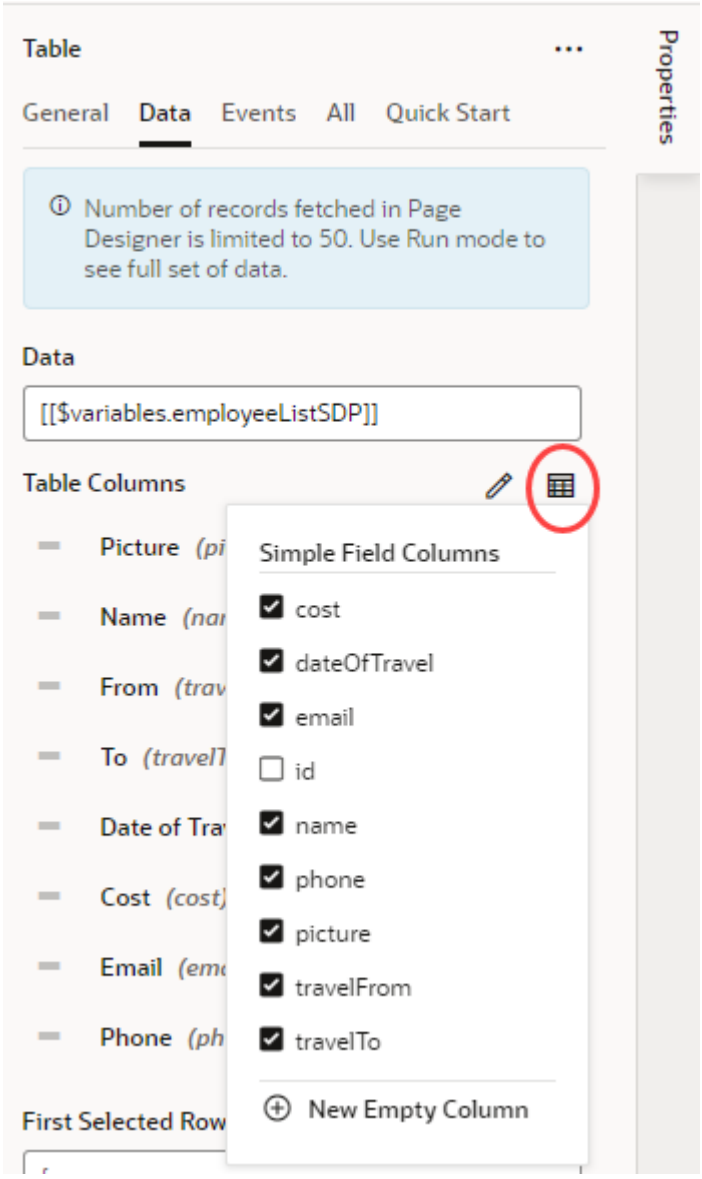
If you used a quick start to set up your table, it's possible to add new or existing fields from your data source, even if you chose not to show them in your table during initial setup.

Tables that are generated by quick starts are typically based on *SDP* or *Service Data Provider*, which is a specific data type designed for variables that are used to send a request to an endpoint. It's the endpoint's attributes that display as columns in your table. To add an endpoint attribute that you didn't include originally, you'll need to modify the type being used by your table. For example, if your table uses `[[$variables.employeeListSDP]]` to store its data (as displayed on the **Data** field of the table's properties in the Data tab):

1. Switch to the **Variables** tab and look for the **employeeListSDP** variable.
2. In the variable's Properties pane, look for the **Type** field and click **Go to Type**.
3. In the Types tab, click **Edit from Endpoint** next to your table's type.
4. When the Edit Type From Endpoint dialog opens, select the attributes you want to add and click **Finish**.

Modify your table to display unused endpoint attributes as columns:

1. Select the table on the page to view its properties.
2. Click the **Data** tab.
3. Click **Edit Columns** ( ) next to Table Columns.
4. Select the field you want to display as a column.



If you want to add an empty column, click **New Empty Column**, then click **Column Detail** to set the **Columns, Header Text**. You can then design this column to display some data.

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:

```
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.

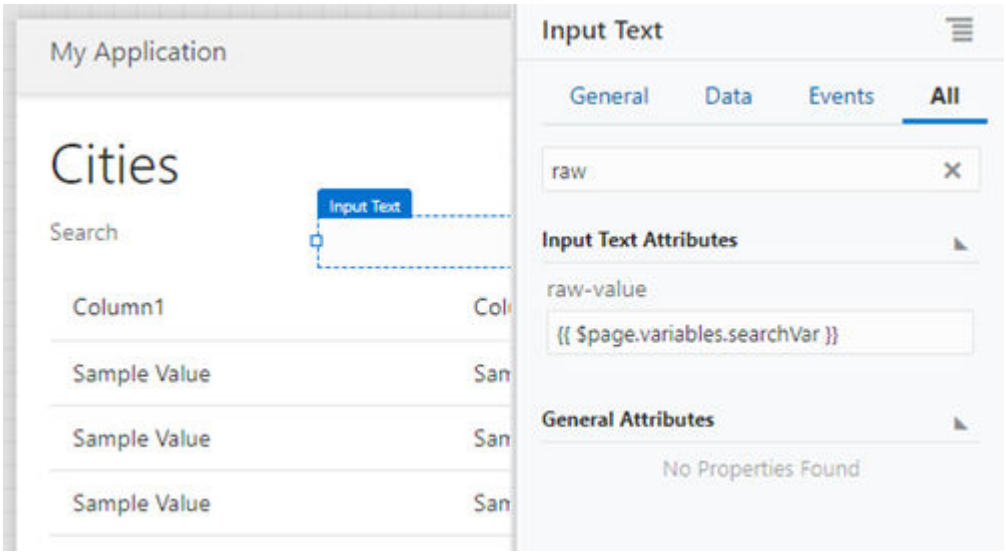
```
<template slot="Salary">
  <span :style.font-weight="{{$page.functions.weight($current.data)}}">
    <oj-bind-text value="[[$current.data]]">
    </oj-bind-text>
  </span>
</template>
```

When the page is displayed, all salary values above 2000 appear in bold.

Create a Search Filter for a Table

You can use an input text component to filter a table column to search for text.

1. Drag an Input Text component onto your page.
2. Click the **All** tab of the component and locate the **raw-value** attribute, which tracks what is typed into the text field.
3. Click the **Select Variable** icon for the attribute to create a new page variable (call it `searchVar`, for example).



After you set the **raw-value** attribute, when you type in the Input Text field, the value of the `searchVar` variable will change.

4. Click the table, then click the **Add Data** quick start to populate the table.
5. On the **Add Data** page of the quick start, specify the business object you want to use and click **Next**.

6. On the **Bind Data** page of the quick start, bind the fields you want to display and click **Next**.
7. On the **Define Query** page of the quick start, click the **filterCriterion** builder icon on the **Target** side.
8. Select the table column you want to filter, the operation you want to use (**$co** for Contains, or **$sw** for Starts With, for example), and the value (`$variables.searchVar`).

Once you have bound the raw value of the input text to a variable and then used that variable as the filter criterion for a table column, you have your search filter. Use the **Run** icon or the **Live** button to test the behavior.

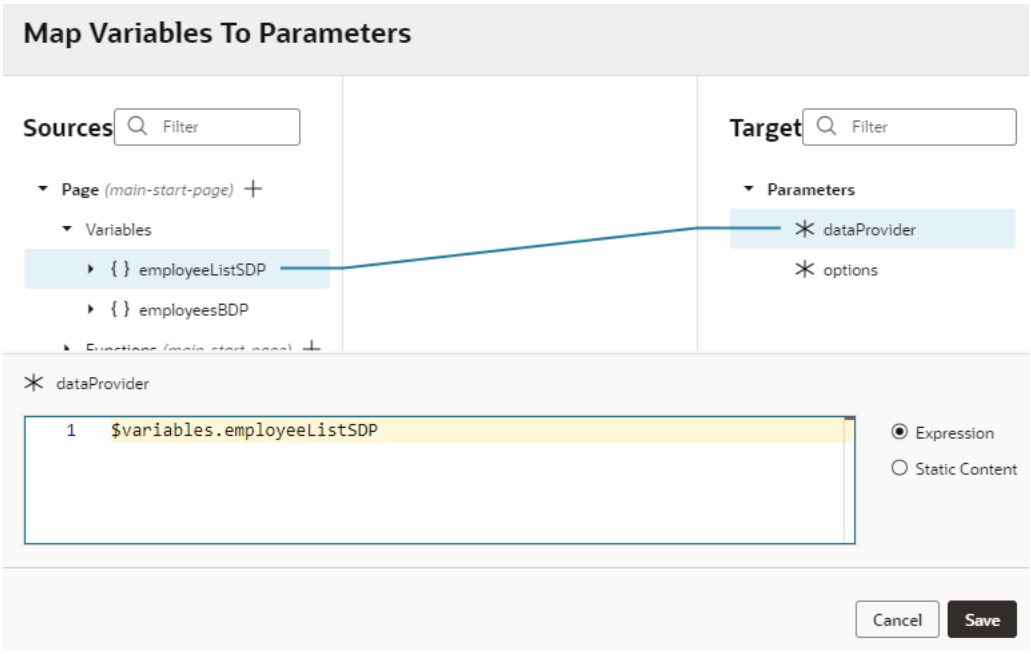
Create an Editable Table

You can create an editable table that allows users to edit multiple existing rows before they submit all their changes in one transaction to the backend service.

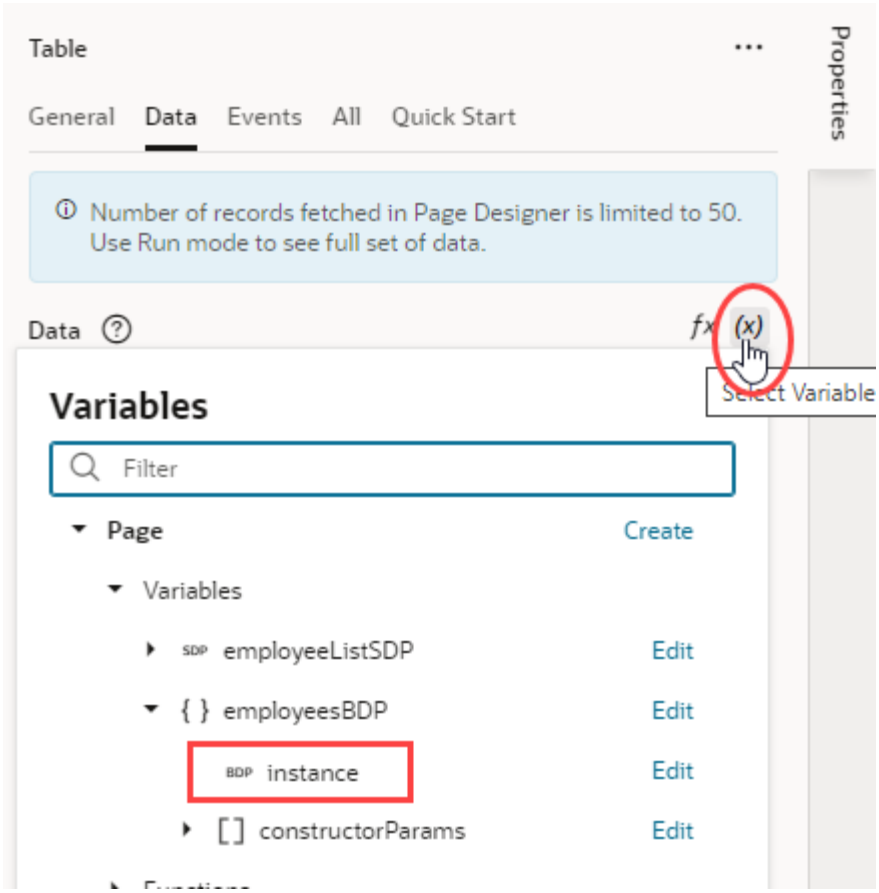
This implementation of an editable table uses the Buffering Data Provider with a Service Data Provider (SDP). The Buffering Data Provider is a wrapper that provides buffering for an underlying data provider, so that edits can be committed to the data source later on. The underlying data provider (Service Data Provider, in this case) is responsible for fetching data, and the Buffering Data Provider merges any buffered edits with the underlying data so they appear as a unified set of data.

Here's how you can create a table that displays editable rows of employees. All changes are stored in a buffer until the user clicks a **Save** button, at which time a REST call posts the changes to the backend service. The example used here follows the Batch Editable Table (BDP) recipe in the Visual Builder Cookbook, which demonstrates additional functionality, such as how to implement validation and create new rows.

1. Add a table to your page and connect it to data using the Add Data quick start. The quick start creates a variable that is automatically bound to the table component. Because this variable is based on the Service Data Provider type, you need to create a variable based on the Buffering Data Provider type and set up your table to reference the new variable.
   a. Drag and drop a table component to your page, then add data to it using the Add Data quick start.
   b. Create a variable (for example, `employeesBDP`) based on the Buffering Data Provider type. In the newly created variable's Properties pane, map the `dataProvider` Constructor Parameter to the `employeeListSDP`:



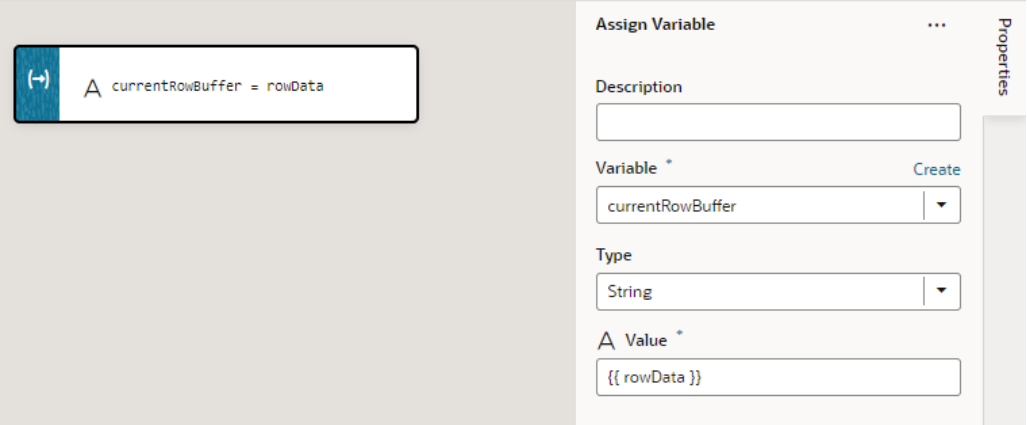
c. In the table's **Data** tab, change the `data` property to point to `[[ $variables.employeesBDP.instance ]]`, instead of `employeeListSDP`.



2. Create event listeners and action chains to handle the table's edit events and make the table editable.

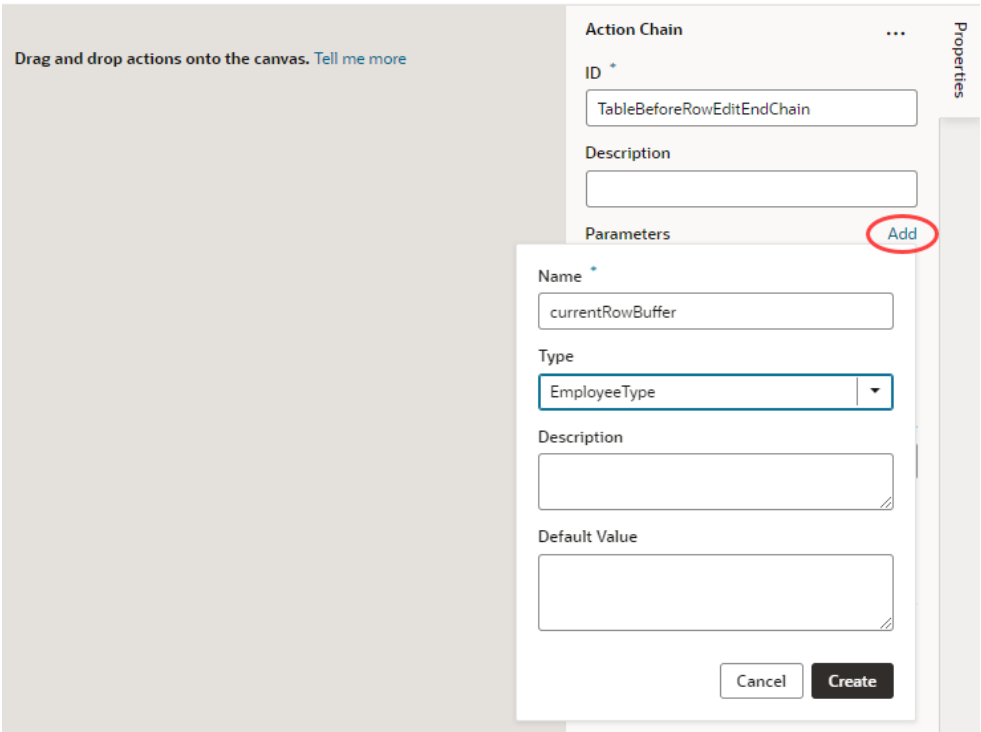
   a. Create a custom type from your endpoint (for example, the `EmployeeType` type based on the `GET /Employee/{Employee_Id}` endpoint), then create a page variable named `currentRowBuffer` that is assigned to the custom type.

   b. Now add an `ojBeforeRowEdit` event listener to the table, then in the `TableBeforeRowEditChain`, add an Assign Variable action with the current row data (`rowData`) assigned to the `currentRowBuffer` page variable.



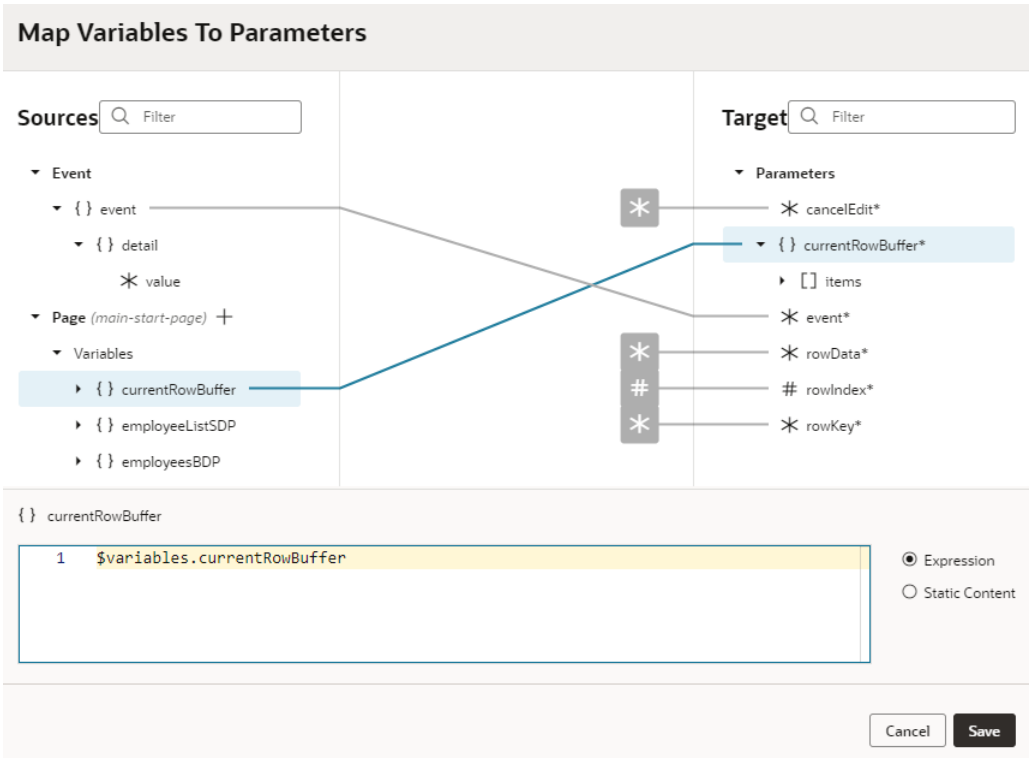
   c. Add an `ojBeforeRowEditEnd` event listener to the table, then in the `TableBeforeRowEditEnd` action chain, add two new Parameters as input parameters:

      - `currentRowBuffer` of type `EmployeeType`



- `event` of type `Any`

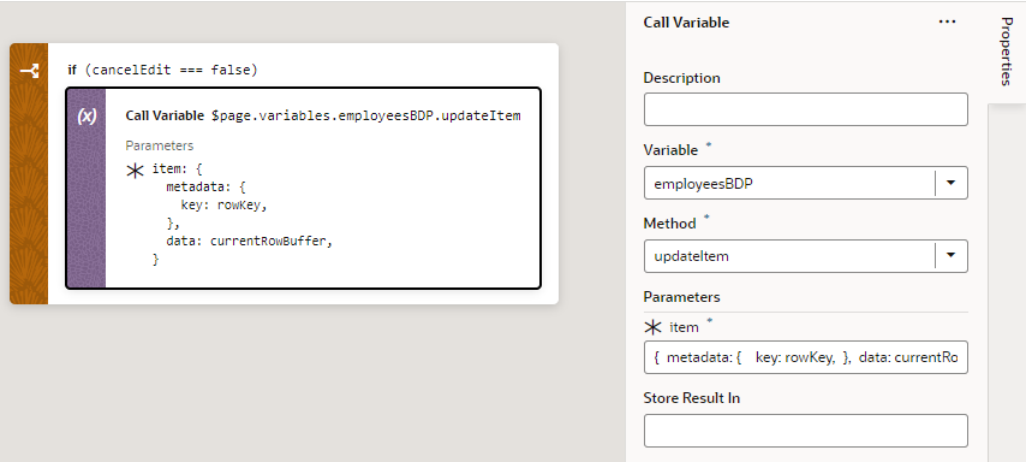
d. To map these two new parameters, open the **Event Listeners** tab, select the **TableBeforeRowEditEnd** action chain under the **tableBeforeRowEditEnd** event. In the action chain's Properties pane, map the new parameters to `$variables.currentRowBuffer` and `$event`, respectively.



e. In the `TableBeforeRowEditEnd` action chain, add an If action and test if the `cancelEdit` parameter is `false`. If `false`, changes need to be stored into the `employeesBDP`.

Storing changes is implemented using a Call Variable action for variable `employeesBDP` and function `updateItem` with parameters:

```
{ metadata: { key: rowKey}, data: currentRowBuffer}
```



f. Both the `ojBeforeRowEdit` and `ojBeforeRowEditEnd` events must have asynchronous behaviour enabled for editable table functionality to work. To do this, switch to the page's JSON view, locate `tableBeforeRowEdit` and `tableBeforeRowEditEnd` under `eventListeners`, and add `"asyncBehavior": "enabled"` for both events:

```
"eventListeners": {
  "tableBeforeRowEdit": {
    "asyncBehavior": "enabled",
    "chains": [{ ... }]
  },
  "tableBeforeRowEditEnd": {
    "asyncBehavior": "enabled",
    "chains": [{ ... }]
  }
```

g. In the table's properties, set **Edit Mode** to **Row Edit** (if the option isn't already selected).

h. Enhance table cells to make them editable:

   i. Drag and drop a Text component onto the first table column which needs to be editable. This generates a cell template for the column.

   ii. In the Structure view, right-click the **Bind Text** node and select **Surround** and **If** to generate a condition for when to render the text component. Set the Bind If condition to `[[$current.mode === 'navigation']]`. When the table row is in read-only mode navigation, the text component will render the column value.

   iii. Drop an Input Text (or any other component) into the same cell template as the Bind Text node and wrap it again in an If condition, but this time set the condition to `[[$current.mode === 'edit']]` to show an editable input text component when the table row is in edit mode.

   iv. Remove the Input Text component's **Label Hint** and **ReadOnly** properties. Bind the **Value** property in the Data tab to the corresponding property from `currentRowBuffer`. For example, `{{ $variables.currentRowBuffer.firstName }}`.

   Use similar steps to make the other table columns editable.

3. Track and save changes to the table. Saving changes involves collecting them from the Buffering Data Provider instance, marking changes as being submitted, creating a single

batch REST Call payload to store all changes in one transaction, and finally marking changed items in the Buffering Data Provider as submitted.

a. Drag a button and drop it above the table on the page. Change the **Label** to `Save` in the button's properties.

b. In the button's **Events** tab, add an event listener for the **On 'ojAction'** event and rename the corresponding action chain as `SubmitChanges`.

c. In the SubmitChanges action chain, add supporting JavaScript functions:

   - The `createBatchPayload` function to generate the payload with all changes. All changes stored in the Buffering Data Provider are accessed using the API call `getSubmittableItems`.
   - The `setStatusTo` function to set Buffering Data Provider items to `submitting`, `submitted`, or `unsubmitted` states. The Buffering Data Provider requires an edit item's status to be set, via the `setItemStatus` API call, before the change is committed to the data source.

   See the batch editable table recipe in the cookbook for sample code.

d. Using these functions, you can build your action chain to change state, save data, and handle error states. See the batch editable table recipe in the cookbook for sample code.

Update Pagination Behavior for a Table

You can improve the performance and usability for a table by controlling the number of records that appear on a page and the total number of records displayed.

Here are some things to consider before adjusting the total number of records and pagination for your table:

- You can either choose to load more records initially with slight delays later in scrolling or have an initial delay when you load records, which will allow for smoother scrolling. The `scroll-policy-options.fetch-size` property controls the number of records feteched initially. For example, if you set this property high to a high number, it will fetch many records at the start and scrolling through them will be smooth, but the initial fetch might be a bit slower and you will consume more memory on the client.
- You might also want to limit the number of total records a user can fetch to control the amount of network traffic and memory consumption on the client. The default for a JET table is set to 500 but you can modify this with the `scroll-policy-options.max-count` property of the table.

> **Note:**
>
> To further improve the table's usability, consider adding filtering controls to the table to allow users to define search criteria to display the records they want listed.

To adjust pagination and total records displayed, you can:

- Add pagination controls to your table to limit the number of records shown on each page and control the total number of records shown.
- Set the table to load all records to fetch all records when the table loads on the page.

Add Pagination Control to Your Table

While users can use the browser's scroll bar to scroll through the rows displayed in table, in most cases you'll want to limit the table's height, so that only a certain number of records are shown at a time.

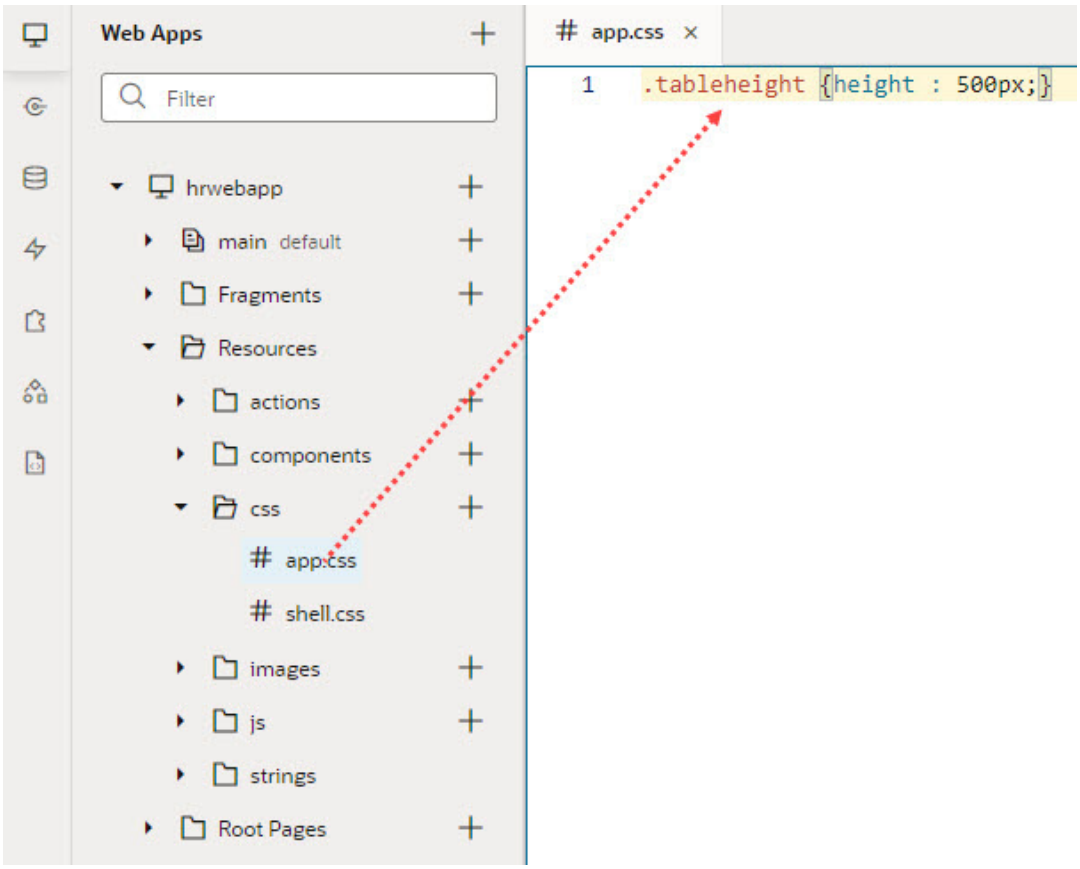
When you set the table's height, paging is automatically implemented and a fixed number of records is fetched at a time (25 is the default).

> **Note:**
>
> For pagination to work efficiently, your backend server needs to support pagination – Visual Builder's business objects and SaaS services support this out of the box. For other REST services that support pagination, you'll need to define a transform function as part of the service definition.

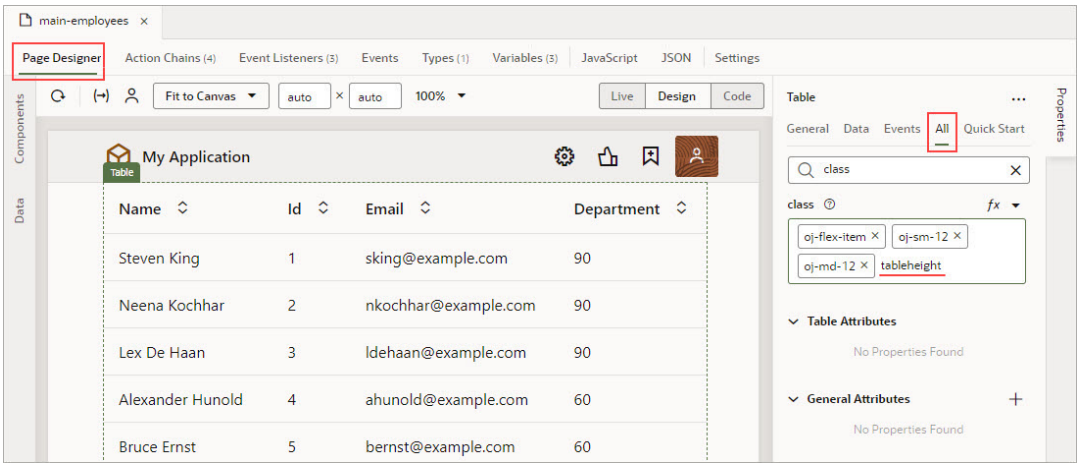
To add pagination control to your table:

1. Define a new CSS class in `app.css`. Open your application in the Designer, then expand the **Resources** and **css** nodes, as shown below. Paste this line into `app.css` and change the height from 500px to the height that would show the number of rows you want:

```
.tableheight {height : 500px;}
```

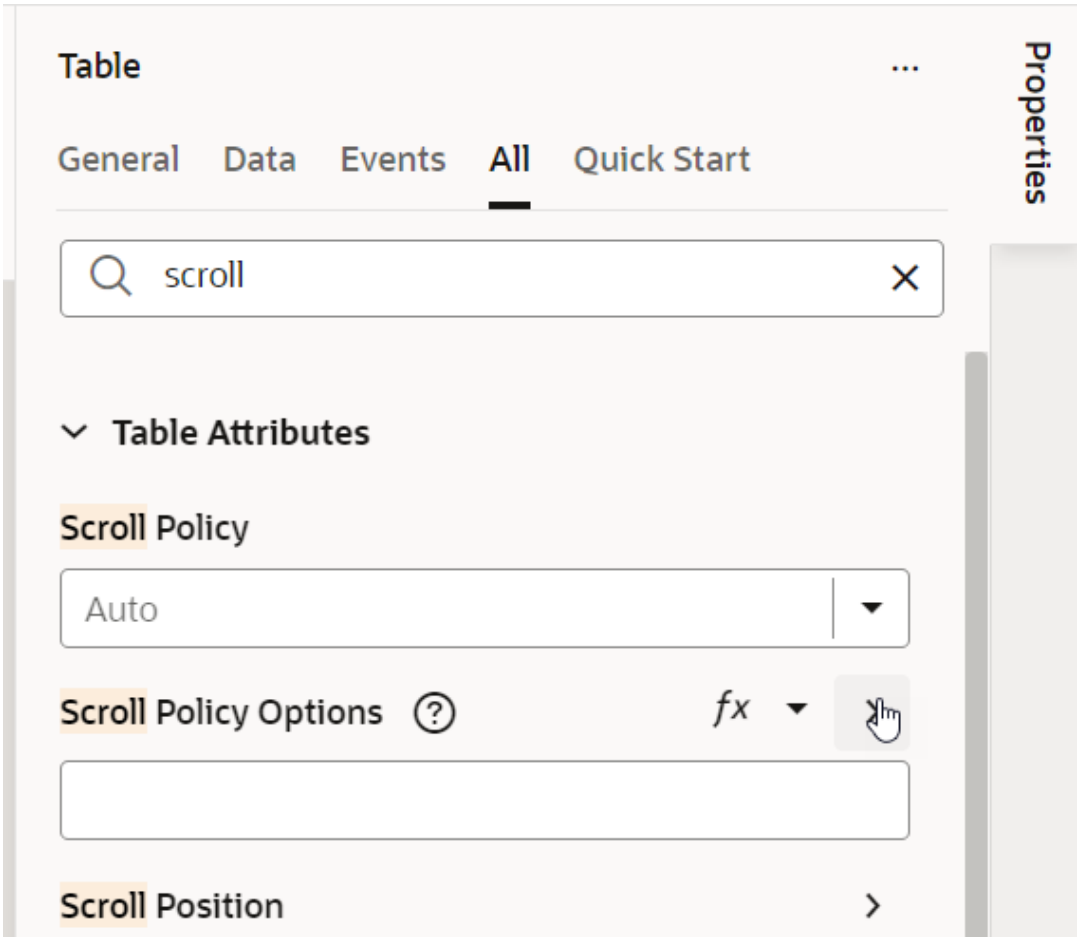


2. Go to the **Page Designer** tab for the page, select the table on the canvas, then select the **All** tab in the Properties pane. Enter `class` in the search box to bring up the **class** property for the table, then enter `tableheight` in the **class** property:

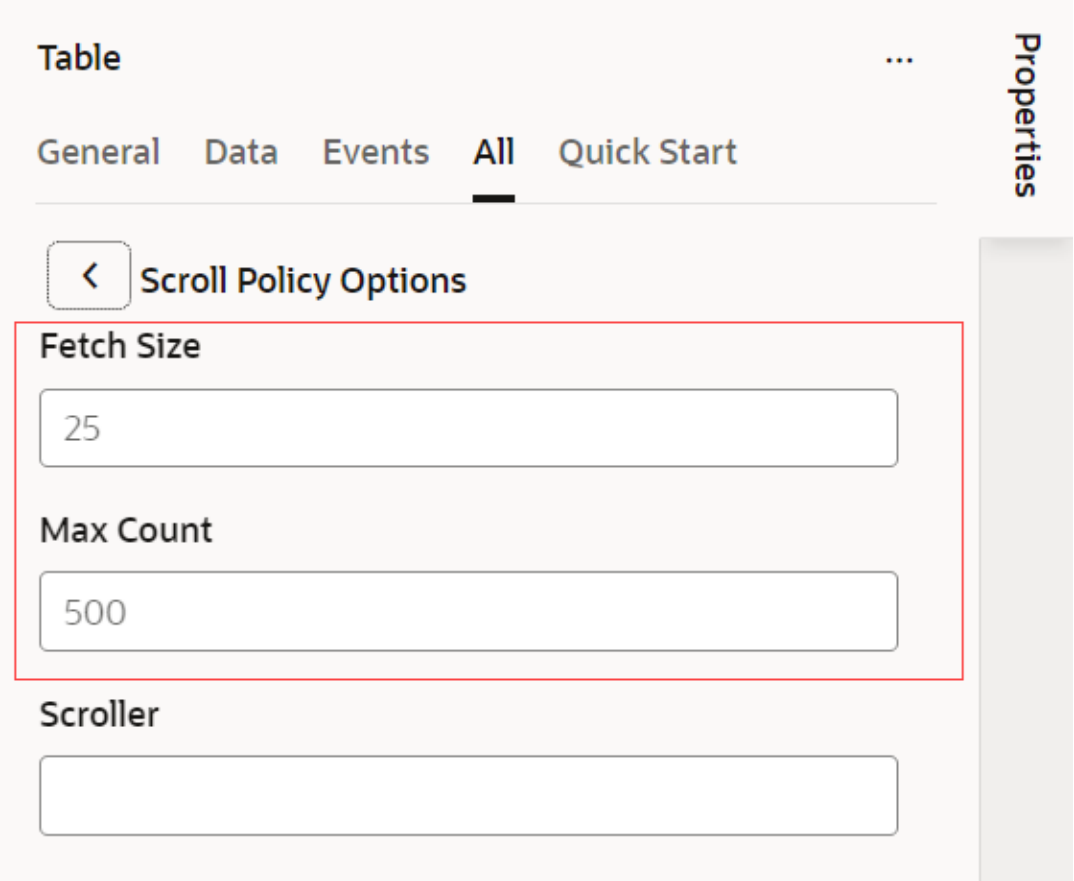


To fine-tune how many total rows are displayed and how many rows are fetched at a time:

1. With the table selected on the canvas, click **All** on the Properties pane and type `scroll` in the search box.
2. Click the arrow on the far right of the **Scroll Policy Options** property:



3. Adjust the **Scroll Policy Options**:
   - For the **Fetch Size** property, enter the optimal fetch size.
   - For the **Max Count** property, enter the total number of records displayed in the table.

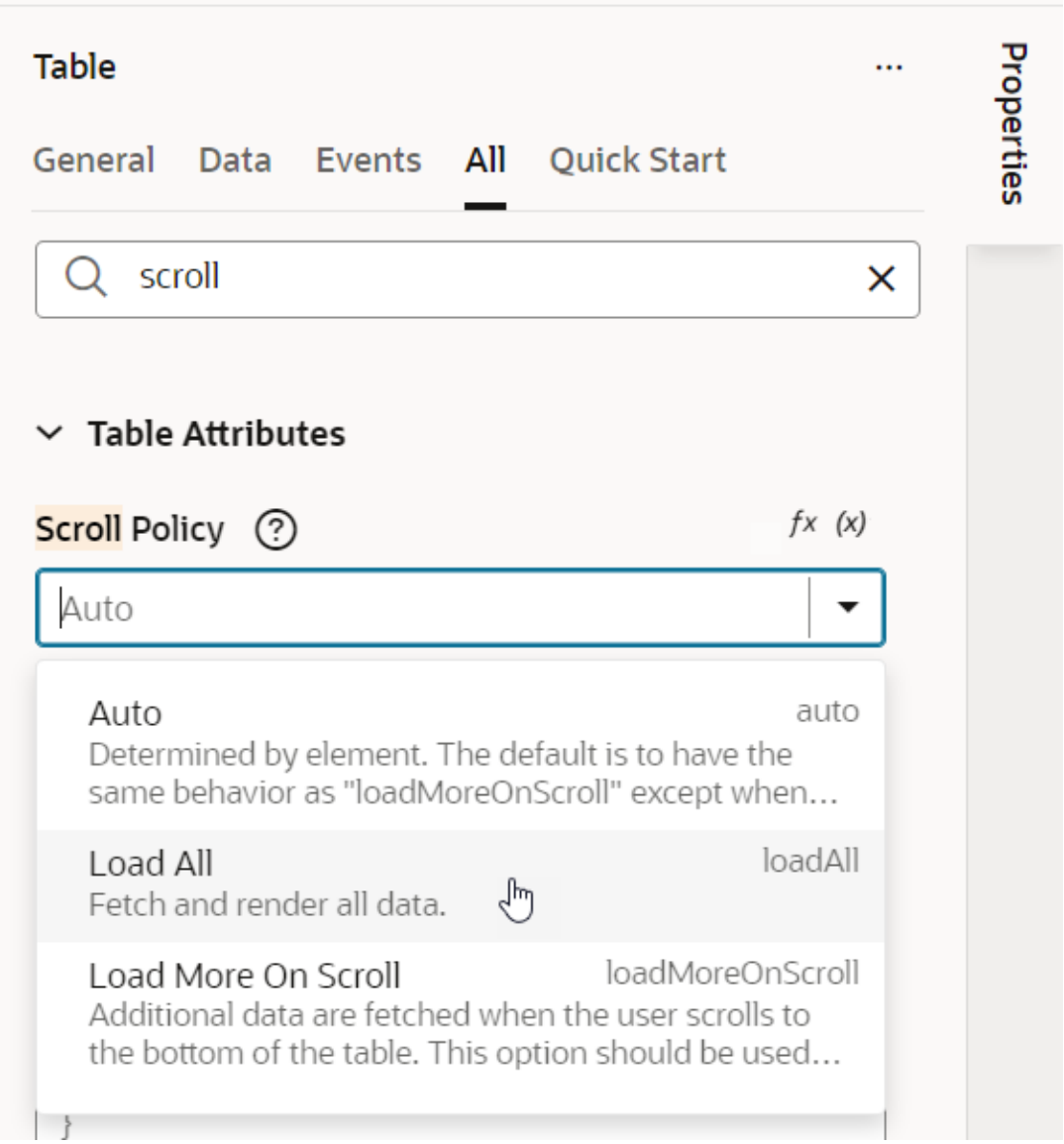


> **Note:**
>
> To keep the table and data source in sync, it is recommended that you set the `#maxSize` property for the SDP `pagingCriteria` to match the `scroll-policy-options.max-count` table setting.

Load All Records

You can load all of the table data onto the page at one time by setting the scroll policy to **Load All**.

1. With the table selected on the canvas, click **All** on the Properties pane and type `scroll` in the search box.
2. From the **Scroll Policy** drop-down menu, select **Load All**:



All table data is fetched and rendered when the table is displayed in the application.

For more details about the scroll properties of the oj-table component, see Oracle Jet - oj.table Element.

Enable Text Selection in a Table

To enable users to select and copy text in a table, add the `user-select:text` style to a row or cell template.

> **Note:**
>
> If you enable text selection, you cannot also enable drag-and-drop functionality for the table.

1. In the Page Designer, select the table and switch to **Code** mode.

2. Add the `user-select:text` style to a row or cell template. To select text in an entire row, add `user-select:text` to the `<tr>` element, for example:

```
<template slot="rowTemplate">
  <tr style="user-select:text">  data-oj-as="row"
     <td>
        <oj-bind-text value="[[row.data.EmployeeId]]"></oj-bind-
text>
     </td>
     ...
  </tr>
</template>
```

> **Note:**
>
> Inline styles are not recommended and might be flagged in audits. If possible, implement this styling as a custom class in your application's CSS file.

To select text in individual cells, add the `user-select:text` style to the cell template. If your cell template uses an `<oj-bind-text>` element, wrap that binding with a `<span>` element and add the style class to that span, for example:

```
<template slot="departmentCellTemplate4">
  <span style="user-select:text">
      <oj-bind-text
value="[[ $current.data.items[0] ? $current.data.items[0].department :
undefined ]]"></oj-bind-text>
  </span>
</template>
```

You can now select text in the table to copy it.

Pages and Flows

Common use cases relating to pages and flows in an application:

- Restrict User Access to an Application, Flow, or Page

Restrict User Access to an Application, Flow, or Page

When you want to limit user access to your application, you can set up user roles at the application level, then restrict access only to those roles. You can use this approach to restrict access to your app, even a page or flow in your app.

> **Note:**
>
> In addition to security settings that are set on the UI layer, you should also secure your backend REST services since client-side restrictions can potentially be hacked. If your backend is developed with business objects in Visual Builder, see Access and Secure Business Objects.

To restrict access to your application:

1. Create a user role (for example, MyAdminRole), then associate it with specific IDCS groups or users, as described in Manage User Roles and Access.

   When you set up user roles for your visual application, the roles map to groups in your IDCS account. These roles act as additional roles on top of the built-in Authenticated User role.

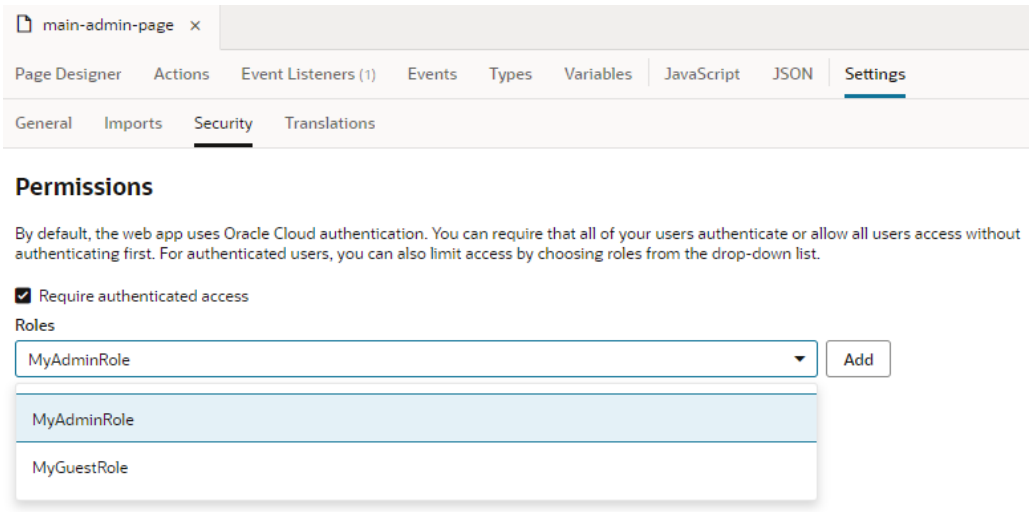
   When access to the app requires authentication (default), all users who sign in to the app with their Oracle Cloud credentials are assigned the Authentication User role. By default, an authenticated user can see and manage all business object data. To change this, update the business object's security settings.

2. Set role-based permissions at the app, flow, or page level:

   a. In the Web Apps pane, click the node for the artifact. To restrict access to the page, for example, you'd click the page node.

   b. Click **Settings**, then **Security**.

   c. From the **Roles** drop-down list, select the role you want to use to restrict access. Here's an example of a page-level configuration:



   d. Click **Add**.

3. Test the application to preview it in different roles, as described in Test Role-Based Access.
4. When you are ready, stage and publish your application, as described in Stage and Publish Visual Applications.

Components

Common use cases that involve the use of specific components:

- Conditionally Show or Hide UI Components
- Enable Time Zone Specification
- Validate the Length of an Entry in an Input Text Field
- Filter Multiple Attributes in a Search
- Set an Initial Value for the Select (Single) Component

Conditionally Show or Hide UI Components

You can dynamically display UI components in your application by surrounding the component with an `oj-bind-if` and setting conditions to either show or hide the component.

See Use Conditions to Show or Hide Components.

Enable Time Zone Specification

Different time zones have different standards for formatting date and time fields. You can implement the appropriate format for a given time zone by customizing the **Input Date Time** component.

You will first need to edit your visual application's JavaScript file to enable the arbitrary time zone setting. You can then customize the **Input Date Time** component to display time zone data.

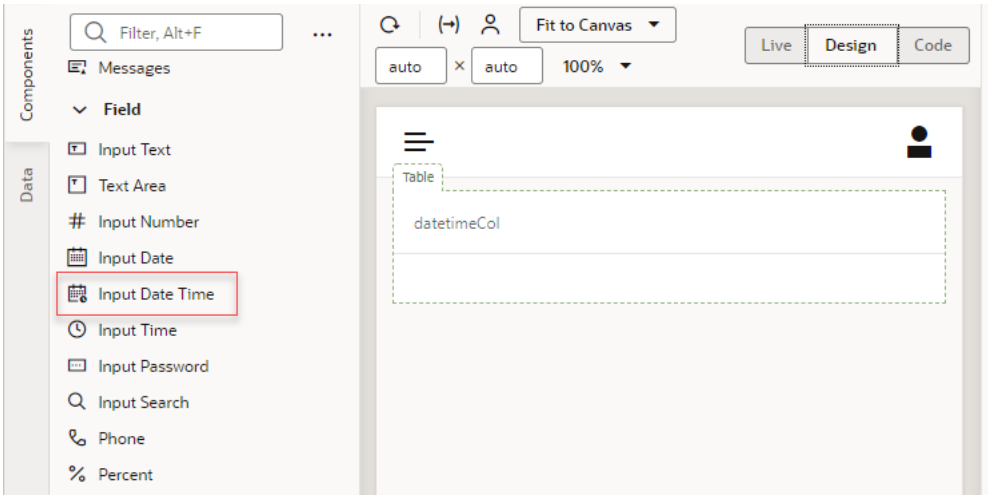
In this example, the Input Date Time component is added to the table with a `datetimeCol` template applied.

1. To enable the time zone specification, add the definition of the `ojs/ojtimezonedata` RequireJS module to your visual application's JavaScript file.

   Here is an example of `ojs/ojtimezonedata` added to a page's JavaScript file:

   ```
   define(['ojs/ojtimezonedata'], function() {
   'use strict'; var PageModule = function PageModule() {};
   return PageModule;
   });
   ```

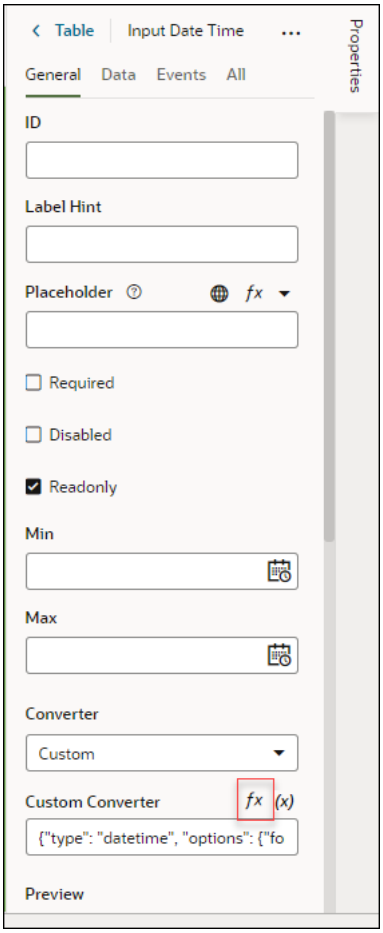
   You can specify this definition for any unit of the page, flow, or application.
2. In the Page Designer, drag and drop the **Input Date Time** from the Field components into the target column.



Here is an example of page code with a cell template that has been applied to a column named `datetimeCol`:

```
<div class="oj-flex">
<oj-table scroll-policy="loadMoreOnScroll" class="oj-flex-item oj-sm-12 oj-
md-12"
data="[[$page.variables.mySampleBOListSDP]]"
columns='[{"headerText":"datetimeCol","field":"datetimeCol","template":"dat
etimeCol"}]'>
<template slot="datetimeCol">
<oj-input-date-time  value='[[typeof $current.data ===
"string" ? $current.data : null]]' readonly="true"></oj-input-date-time>
</template>
</oj-table>
</div>
```

3. In the Structure view, select the **Input Date Time** component that you added.
4. In the Properties pane, choose **Custom** from the **Converter** drop-down list, then click **fx** to open the Expression Editor.



5. In the Expression Editor, set the `timeZone` property to a time zone in ISO 8601 format, and save.

   Here is an example with the Asia/Tokyo time zone specified.

```
{"type": "datetime", "options":
      {"formatType":"datetime", "pattern": "MM dd, yyyy h:mm:ss a Z",
"timeZone":
      "Asia/Tokyo"}}
```

   When the custom converter is applied, the page code described above will look like this:

```
<div class="oj-flex"><oj-table
      scroll-policy="loadMoreOnScroll" class="oj-flex-item oj-sm-12
      oj-
md-12"data="[[$page.variables.mySampleBOListSDP]]"columns='[{"headerText":"
datetimeCol","field":"datetimeCol","template":"datetimeCol"}]'><template
      slot="datetimeCol"><oj-input-date-time value='[[typeof $current.data
=== "string" ?
      $current.data : null]]' readonly="true"converter='{{  {"type":
"datetime",
      "options": {"formatType":"datetime", "pattern":  "MM dd, yyyy
h:mm:ss a Z",
```

```
"timeZone": "Asia/Tokyo"}}
}}'></oj-input-date-time></template></oj-table></div>
```

Validate the Length of an Entry in an Input Text Field

You can add code to an input field to validate the length of the text you enter in a form.

Suppose you have a form for creating a business object instance that has an input text field for capturing a numerical customer ID. You want to be sure that the number entered is between 3 and 7 characters. To do this, add this code to the input text field:

```
<oj-input-text id="oj-input-text--285061062-2" class="oj-flex-item oj-sm-12
oj-md-4" validators='[{
    "type"           : "regExp",
    "options"        : {
    "pattern"        : "[a-zA-Z0-9]{3,7}",
    "messageSummary" : "wrong length",
    "messageDetail"  : "need 3-7 chars"}}]' >
```

> **Tip:**
>
> You can also add a field validator to the input text field to enforce the same rule. See Field Validators for Business Objects.

Any time you add validation for an input text field and want to customize the message displayed, simply add or edit the `messageDetail` attribute.

Filter Multiple Attributes in a Search

You can apply a single search input term to multiple attributes of a service endpoint by setting up a `filterCriterion` with many conditions.

You can use `filterCriterion` to filter data displayed in a Table, List, or another collection component in Visual Builder. For information on how to create a search filter in a Table, see Create a Search Filter for a Table. You can use this configuration for a List View component as well. For detailed information on filtering data, see Filter Data Displayed in a Component.

Once you have your search filter, simply extend the filter criteria. For example, to filter data in the Traveler and Destination table columns, your filter criteria might look as shown here:

```
{
"criteria": [
{
"value": "{{ $page.variables.filterVar }}",
"op": "{{ \"$eq\"\n }}",
"attribute": "{{ \"traveler\"\n }}"
},
{
"value": "{{ $page.variables.filterVar }}",
"op": "{{ \"$eq\"\n }}",
"attribute": "{{ \"destination\"\n }}"
}
],
```

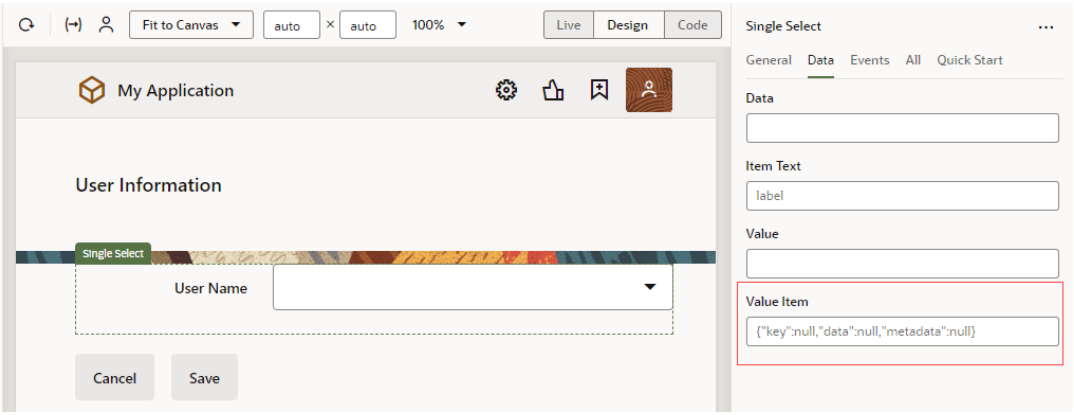
```
"op": "{{ \"$or\"\n }}"
}
```

Set an Initial Value for the Select (Single) Component

You can set an initial value for the **Select (Single)** component of a visual application using the **Value Item** attribute.

> **Note:**
>
> The **Select (Single)** component from the Component palette is labeled **Single Select** in the Page Designer.

1. In the Page Designer, select the **Single Select** component.
2. Select the **Data** tab in the Properties pane.



3. Set the initial value for the **Value Item** attribute.

   The default for this attribute is:

   ```
   { "key": null, "data": null, "metadata": null }
   ```

   You can assign set values to the attribute. For example:

   ```
   {"key":2,"data":{"userName":"user002"},"metadata":null}
   ```

   You can also use variables to specify the attribute:

   ```
   {{ {"key":$variables.initialID,"data":
   {"userName":$variables.initialuserName},"metadata":null} }}
   ```

Business Objects

Some common use cases relating to business objects in a visual application:

- Format a Date Field

- Apply an Aggregate Function to a Calculated Field From a Child Business Object

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 Components palette and drop it on top of the date field.
2. Set that field's converter property to match the format you need.

   For details about how to use the converter property to format a date, see Use Converters to Change Date and Time Formats and Time Zones.

Apply an Aggregate Function to a Calculated Field From a Child Business Object

When you want to aggregate a field in a parent business object based on a calculated field in a child business object, the calculated field won't show up as a **Field to Aggregate** field with the **Aggregate from related business object data** option in a field's Properties pane or the **Create Aggregate Field** option in the Fields tab. Using a calculated field with declarative field aggregation isn't supported, but you can get around this by storing the calculated value in the business object.

Consider a sample Shopping Cart scenario, where you have a `Shopping Cart` business object with `Cart Item` as its child business object. Assuming that `Cart Item` has a field `cartItemAmt` (calculated as `Qty` * `Unit Price`), you're trying to aggregate `cartItemAmt` as the `TotalAmount` in the `Shopping Cart` business object.

The recommended approach for this requirement is to store the calculated value in the business object. You can do this by adding a trigger to the detail item that will catch any transaction and update the calculated field in the parent field (see Field Triggers).

This approach has the added advantage of speed; because the calculation isn't done at runtime, your page will load faster.

30

Troubleshooting & FAQs

Look here for possible solutions to some common issues when using Visual Builder:

Topics:

- How Do I Find the URL of My Visual Builder Instance?
- How Do I Find My Application's Runtime Version?
- How Do I Clear My App's Resource Cache?
- How Do I View Details of Client Apps in IDCS?
- How Do I Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null?
- How Do I Resolve Web Component Loader Issues?
- How Do I Resolve a 'Method Not Allowed' Error?
- How Do I Resolve a 'No Such File' Error for the URI.js File?
- How Do I Fix a Missing Scroll Bar in a Table?
- How Do I Access Components After Upgrading?
- How Do I Set a Custom Content-Security-Policy Header?
- Troubleshooting Access Issues
- Troubleshooting Service Connections
- Troubleshooting Business Objects

How Do I Find the URL of My Visual Builder Instance?

When you file a service request for Visual Builder, you'll need to provide the URL of the Visual Builder service instance that was provisioned for you. Here's how to locate your Visual Builder service instance's URL:

1. Launch the Oracle Cloud Console at https://www.oracle.com/cloud/sign-in.html.
2. Provide your Cloud Account Name (the account that you chose when you signed up) and click **Next**.
3. Sign in with the user credentials that you received when you set up your account.
4. Click the menu at the top of the page, then select **Developer Services** and **Integration** in the navigation tree.
5. From the **Compartment** list, select the compartment where your Integration instance is hosted.
6. On the instances page, click the instance link in the **Name** column.
7. Click **Service Console** on the instance's details page.
8. When the instance opens in a new browser tab, click **Show/Hide Navigation menu** ☰ in the top corner of the Home page.

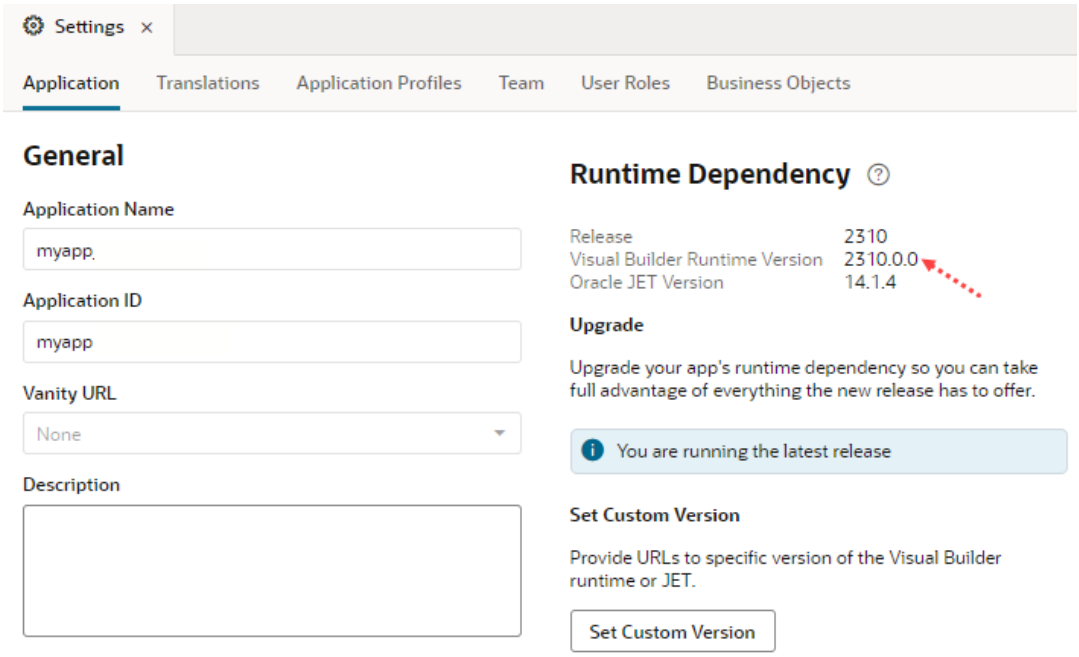
9. In the navigation pane, click **Visual Builder**.
10. When the instance opens in a new browser tab, make a note of the URL.
11. Provide this URL when filing a service request with Oracle Support.

How Do I Find My Application's Runtime Version?

You may need to locate your runtime version if you contact Oracle Support with an issue or if you've received a message that your app needs to be upgraded to a newer runtime version.

To find the application runtime:

1. Click **Menu** in the upper right corner and select **Settings**.
2. Locate the **Visual Builder Runtime Version** in the Runtime Dependency section:



How Do I Clear My App's Resource Cache?

Oracle Support might sometimes ask you to clear your visual application's resource cache, so you can refetch its contents from the server. To do this:

1. Click **Menu** in the upper right corner and select **Settings**.
2. Click **Clear Client Caches** under Troubleshooting.
3. If prompted to confirm, click **Yes**.

Clearing the cache removes persistent data stored in your browser and reloads the page.

How Do I View Details of Client Apps in IDCS?

Each time a visual application is created on a Visual Builder instance, a companion *OAuth client application* is automatically created in IDCS. As your app progresses to its deployment phase, another client application is created for each application profile used to deploy the app

to a development, staging, or production instance. This means that a single visual application may have several client apps running on IDCS for the lifespan of the app.

When troubleshooting issues with your app, especially those relating to identity propagation, it might be helpful to check whether a client app exists and examine it for configuration issues—which you can do by directly accessing the client app from the visual application's Settings editor.

1. Click the **Menu** in the upper-right corner of your visual application, then select **Settings**.
2. Click **Application Profiles** to view the application profiles used by your app.
3. Select the profile whose details you want to view, then click **IDCS Details**. For example, here's what you might see for the Base configuration profile, used by default for development, test, and production deployments:

4. Use the links to access your IDCS client app and examine its details. If you don't have permissions, you can provide the client app ID to your administrator or Oracle Support.

Field	Description
Base IDCS URL	Base URL of the Oracle Identity Cloud Service instance used by your Visual Builder instance. Only IDCS administrators can access the instance at this URL.
OAuth Client Application ID	ID of the OAuth client application used by a visual app to request an OAuth2 authorization token (whose value is asked once and kept for the duration of a session).
Link to IDCS application	Link to the actual client app used by your visual app's application profile. If you have the required permissions, you can click this link to open the client app on the IDCS console and view its details.
Added Scopes	OAuth scopes in IDCS that the visual app has access to and which can be used in token-based flows for service connections.

How Do I Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null?

To write efficient expressions that handle situations where a referenced field might not be available or the field's value could be null, use the JavaScript optional chaining operator (`?.`) and the nullish coalescing operator (`??`). These operators are supported in standalone JS files as well as in HTML/JSON file expressions.

To avoid exceptions that might occur because of a missing field, which can happen when a field is optional, use the optional chaining operator (`?.`). The optional chaining operator (`?.`) enables you to read the value of a property located deep within a chain of connected objects

without having to check that each reference in the chain is valid. The `?.` operator is like the `.` chaining operator, except that instead of causing an error if a reference is nullish (`null` or `undefined`), the expression short-circuits with a return value of `undefined`. When used with function calls, it returns `undefined` if the given function does not exist.

For example, when your expression is `$fields.USMType_c?.value()`, JavaScript will check to make sure that `$fields.USMType_c` is not `null` or `undefined` before trying to access `$fields.USMType_c?.value()`. If `$fields.USMType_c` is `null` or `undefined`, the expression automatically short-circuits, returning `undefined`. See optional chaining operator.

To avoid exceptions that might occur because of a missing field value, use the optional nullish coalescing operator (`??`). Using the `$fields.USMType_c?.value() ?? 42` example, if the value is nullish, 42 will be returned. This sets a default value when no value is found. See Nullish coalescing operator.

How Do I Resolve Web Component Loader Issues?

If creating a custom web component from an application's Resources node returns the `Unable to load CCA loader: top-navigation/loader: SyntaxError: Unexpected token < in JSON at position 0` error, your web component loader was most likely created incorrectly.

Your loader must be created to use the order that matches paths and parameters, for example:

```
define([
  './viewModel',
  'ojs/ojcomposite',
  'text!./component.json',
  'text!./view.html',
  'css!./styles'
], function (
  viewModel,
  Composite,
  metadata,
  view) {
...
```

If you still run into the `Unable to load CCA loader` error, make some dummy changes to the web component loader and revert your changes. These actions will force a reload of the web component loader and flush out old dependencies.

How Do I Resolve a 'Method Not Allowed' Error?

If you try to call a web application entry point using POST, you'll get the error message "Method Not allowed". Due to restrictions imposed by IDCS, Visual Builder can accept the GET method only.

How Do I Resolve a 'No Such File' Error for the URI.js File?

If you get a `no such file` error for the `URI.js` file when building or deploying your application using the grunt command line vb-build command, this means that the `requirejs` optimizer can't find a reference to the `urijs` library:

```
Running "_vb-optimize-prepare-ojL10n-plugin:xxxxx" (_vb-optimize-prepare-ojL10n-plugin)
task
downloading ojL10n plugin from https://static.oracle.com/cdn/jet/v7.1.1/default/js/debug/
ojL10n.js
to build/ojL10n.js

Running "requirejs:xxxx" (requirejs) task
Error: ENOENT: no such file or directory, open
'C:/xxxx/VBCS/1.0/build/optimized/webApps/test/version_5244468526385398434/urijs/URI.js'
In module tree:
    services/daily_impact_list/transforms

[Error: Error: ENOENT: no such file or directory, open
'C:/xxx/VBCS/1.0/build/optimized/webApps/test/version_5244468526385398434/urijs/URI.js'
In module tree: services/daily_impact_list/transforms
    at xxxxxxx\1.0\node_modules\requirejs\bin\r.js:28332:19
    at xxxxxxx\1.0\node_modules\requirejs\bin\r.js:3059:39
    at xxxxxxxxx\1.0\node_modules\requirejs\bin\r.js:2999:25
    at Function.prim.nextTick (xxxxxxxxx\1.0\node_modules\requirejs\bin\r.js:28083:9)
    at Object.errback (xxxxxxxxx\1.0\node_modules\requirejs\bin\r.js:2998:26)
    at Object.callback (xxxxxxxxxxx\1.0\node_modules\requirejs\bin\r.js:2984:23)
    at Object.then (xxxxxxxxx\1.0\node_modules\requirejs\bin\r.js:3038:23)
```

To resolve this problem, empty the path to this library for the optimizer using one of these methods:

- (Recommended) Add the following code to the `Gruntfile.js` file in the root of the application sources:

  ```
  {code:java}
  grunt.initConfig({
      "vb-require-bundle": {
      crosswalk: {
      options: {
      emptyPaths: ["urijs"]
      }
      }
      }
      });
  {code}
  ```

  Or

- Add the `{{--emptyPaths=urijs}}` option to the grunt vb-build command.

How Do I Exclude a JavaScript Library from a Packaging Job

When building your application or extension, the packaging job might fail if you try to include a JavaScript library in the bundle.

To resolve this problem, you can configure the `vb-require-bundle` Grunt task to exclude the library from the bundle. For example, the Grunt file to exclude the `exceljs.min` JavaScript library might look like this:

```
'use strict';

/**
 * Visual Builder application build script.
 * For details about the application build and Visual Builder-specific grunt
tasks
 * provided by the grunt-vb-build npm dependency, please refer to
 * https://www.oracle.com/pls/topic/lookup?ctx=en/cloud/paas/app-builder-
cloud&id=visual-application-build
 */
module.exports = (grunt) => {
    require('load-grunt-tasks')(grunt);

    grunt.config('vb-require-bundle', {
        "options": {
            "emptyPaths": ["xlsx"],
            "bundles": {
                "vb-app-bundle": {
                    "modules": {
                        "find": [
                            ".*(\\.(js|json|html|css))$",
                            "!resources/js/exceljs",
                        ]
                    }
                }
            }
        }
    });

};
```

How Do I Fix a Missing Scroll Bar in a Table?

There is an issue with the Cascading Style Sheet (CSS) classes which results in the scroll bar not appearing in the table. Because the default behavior is to fetch 25 records and only load more records on scroll, you can't view more than the initial 25 records.

To show the scroll bar and allow you to scroll more records, you need to set a specific height to the table using the style property. Select the table in the Designer and set the height of the table; for example, to 300 pixels.

When you run your app, the table now displays the scroll bar.

For more tips for adjusting the behavior of your table, see the Table Pagination and Scrolling in Visual Builder Explained blog.

How Do I Access Components After Upgrading?

If you can't find the JET components you need after an upgrade, make sure that your component exchange connection is set up correctly.

For more information, see Manage Your Component Exchange.

How Do I Set a Custom Content-Security-Policy Header?

The `Content-Security-Policy` header is a HTTP response header that allows you to restrict resources (such as JavaScript, CSS, and images) that can be loaded in your app and from where. By default, Visual Builder sets an appropriate value for the header, but you can choose to override it for your app.

The default value denies embedding (or allows it if configured in the Security tab of the app-level Settings editor). It also allows the use of scripts and styles imported from HTTPS sources alone, in addition to inline scripts and styles. If this isn't suitable for your app, you can set your own header value by adding the `contentSecurityPolicy` property to the `userConfig` element in your application's `app-flow.json` file.

1. On the **Web Apps** tab in the Navigator, select your web app, then click the **JSON** tab to open the `app-flow.json` file.
2. Configure the `userConfig` element in the file, which by default is defined as:

```
"userConfig": {
   "type": "vb/DefaultSecurityProvider",
   "configuration": {},
   "embedding": "deny"
    },
```

Add the `contentSecurityPolicy` property and define its value to the exact directives you want for the header in your server responses (refer to the CSP Reference for details):

```
  "contentSecurityPolicy": "<your-value>",
```

If you want to allow embedding, make sure you define the directive as part of your header; otherwise, the embedding configuration specified in the app-level Security settings takes effect. This setting by default denies embedding (`"embedding": "deny"`).

When the `contentSecurityPolicy` property isn't specified, the following default configuration takes effect:

```
   "contentSecurityPolicy": "frame-ancestors 'self', script-src 'self'
'unsafe-eval' 'unsafe-inline'
      https:, style-src 'self' 'unsafe-inline' https:"
```

Troubleshooting Access Issues

Topics:

- How Do I Control the Session Duration For My Visual App?

- Why Does a Live App That Allowed Anonymous Access Prompt for Login?
- How Can I Recover Apps Linked to a Deleted User Account?

How Do I Control the Session Duration For My Visual App?

The session duration for your visual app is controlled in Identity Cloud Service (IDCS). See Change Session Settings for information on changing those settings.

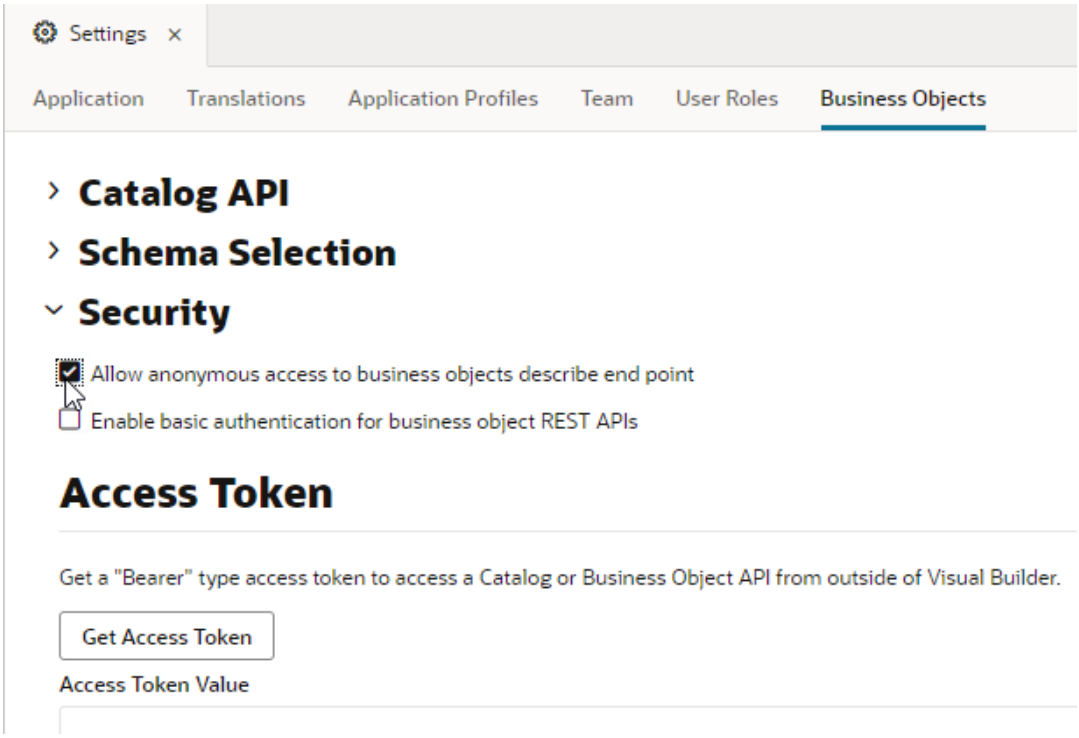
If you have single sign-on (SSO) set up with other identity providers (for example, Oracle Cloud Applications, Okta, Azure, etc), IDCS will use the minimum session duration of all the identity providers that are involved with the session. Therefore, if you need to adjust the session timings for a visual app in which IDCS has an SSO setup, you need to tweak the session timings of all identity providers.

Why Does a Live App That Allowed Anonymous Access Prompt for Login?

Applications that allow anonymous access and have business objects with anonymous access must explicitly allow anonymous access to the Describe endpoint for business objects in your visual application. If your app is already published, you might need to enable this option and publish your application again.

To allow anonymous access to the Describe endpoint for business objects in your visual application:

1. Create a new version of your live application. See Create a New Version of an Application.
2. Open the **Business Objects** tab of the visual application's Settings editor.
3. Select **Allow anonymous access to business objects describe end point**.



4. Stage and publish the new version of your application. See Stage and Publish Visual Applications.

How Can I Recover Apps Linked to a Deleted User Account?

In Oracle Integration Cloud, Visual Builder apps are tied to the application creator. If the creator's user account is deleted and you want to remove or recover apps associated with that user account, you'll need to assign a new user to the app from the Oracle Identity Cloud Services (IDCS) console.

1. On the IDCS Admin Console, click the menu in the top-left corner and select **Applications**.
2. Search for your Visual Builder environment using its unique name. Get the details from the Visual Builder instance on the Dashboard.
3. Click **Application Roles**, then assign a new user with the Service Administrator role.
4. Log in to Visual Builder using the user account associated with the Service Administrator role.
5. On the Visual Applications page, select the **Administered by Me** checkbox and find your application.

Troubleshooting Service Connections

Topics:

- Why Was a Certificate in the Remote Path Reported as Invalid?
- How Do I Resolve an "Unknown Host" Error?
- How Do I Resolve a "Cannot Process Service Scope" Error?
- How Do Timeout Settings Affect External REST Calls in a Visual App?
- How Do I Resolve a CORS Error?

Why Was a Certificate in the Remote Path Reported as Invalid?

When a connection to an Oracle Integration service returns the `A certificate in the remote path is reported as invalid because of Certificate chaining` error, it usually means the SSL certificate in the keystore is either missing or is corrupt. This issue can cause 400 and 502 HTTP errors.

Follow these steps to upload the Visual Builder certificate to the keystore:

1. Get the certificate. There are many ways to get the Visual Builder certificate and the steps may be different for your browser version. These steps are for Firefox on Windows:
    a. Open the Visual Builder Home page in your browser (for example, https://xxx.integration.ocp.oraclecloud.com/ic/builder/).
    b. Click the **Secure** icon on the left of the URL in the address bar.
    c. In the pop-up window, click the **Show Connection Details** arrow for the Secure Connection entry, then click **More Information**.
    d. Click **View Certificate** in the Page Info dialog.
    e. For the first certificate entry, click the **PEM (cert)** link to download the certificate.
    f. Click **Save** and **OK**.

g. Enter a file name and directory, then click **Save** to download the certificate as a PEM file to your local system.

2. Upload the certificate to the Visual Builder keystore:
   a. Log in to Visual Builder as an admin user.
   b. Click the menu in the top-left corner and select **Certificates**.
   c. Click **Upload**.
   d. In the **Upload Certificate** dialog, enter an alias for the certificate and drag and drop the certificate that you saved previously.
   e. Click **Upload**.

How Do I Resolve an "Unknown Host" Error?

If you're not able to invoke a REST service URL from Visual Builder in your network because of an `Unknown host` error, you'll need to make sure that the REST endpoint is publicly accessible.

Here's an example of an `Unknown host` error:

```
{    "type":
      "HTTP://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.1",
      "status": 400,
      "title": "Bad Request",
      "detail":
      "{\"type\":\"abcs://proxy_auth_problem/unknown_host\",\"title\":\"Unknown
      Host\",\"detail\":\"Authentication server xxx: Name or service not known is
      unknown\",\"status\":400}"
}
```

Try pinging the domain listed in the endpoint URL. If you continue to get the `Unknown host` error, take steps to expose the endpoint to the public internet, so Visual Builder can invoke the service.

How Do I Resolve a "Cannot Process Service Scope" Error?

If you see a `Cannot process service scope` error when you're trying to connect to an external service using identity propagation authentication, you probably haven't associated the service and the Visual Builder application in Oracle Identity Cloud Service (IDCS), or the service isn't represented in IDCS as a Resource application.

Using identity propagation authentication mechanisms, such as Oracle Cloud Account, Delegate Authentication, or OAuth 2.0 User Assertion, to call a REST service requires the service endpoint to be hosted in the IDCS Identity Domain URL. In the following example, a HTTP 400 error occurs because the `https://`*`servicename-cloudaccount`*`.integration.ocp.oraclecloud.com` endpoint isn't associated with the IDCS Identity Domain:

```
{
"type":"HTTP://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.1",
"status": 400,
"title": "Bad Request",
"detail": "{\"type\":\"abcs://proxy_problem/jwt/
uri\",\"title\":\"InvalidURI\",\"detail\":\"Cannot process
\\\"https://servicename-cloudaccount.integration.ocp.oraclecloud.com/XxAdfRESTAppTest4-
RESTWebService-context-root/resources/lookups\\\"\",\"status\":400,\
"o:errorDetails\":[{\"type\":\"abcs://proxy_problem/auth/scope/update\",
```

```
\"title\":\"Invalid service scope\",\"detail\":\"Cannot process service scope
\\\"https://servicename-cloudaccount.integration.ocp.oraclecloud.com/\\\" in IDCS, for
URI
\\\"https://servicename-cloudaccount.integration.ocp.oraclecloud.com/XxAdfRESTAppTest4-
RESTWebService-context-root/resources/lookups\\\"\",\"status\":400}]}
"
}
```

Because Delegate Authentication assumes co-location of resources or default established trust relations, follow these steps to create the necessary association:

1. Configure the OAuth layer for the endpoint (`https://servicename-cloudaccount.integration.ocp.oraclecloud.com`) to accept the IDCS Identity Domain URL (`https://idcs-xxxxxxxxxxx.identity.oraclecloud.com`) as a Trust issuer. See Manage Oracle Identity Cloud Service Identity Providers.
2. From the IDCS Admin console, create a "Resource" application that exposes the primary audience (`https://servicename-cloudaccount.integration.ocp.oraclecloud.com`) and scope (`/`).

How Do Timeout Settings Affect External REST Calls in a Visual App?

Timeout settings vary depending on how a service connection's backend is configured. Depending on the backend's connection type and authentication method, web apps can call external REST services directly or through the Visual Builder proxy.

If the REST service is:

- Called directly from the app, then you only need to worry about the browser timeout. Most browsers time out at around 4-5 minutes.
- Routed through the proxy, then in addition to the browser timeout, you should also be aware of the proxy timeout. The proxy times out if the REST API doesn't respond within 234 seconds.

To see how different combinations of authentication and connection types impact the way REST calls are routed, see Use an Appropriate Connection Type to Handle CORS for REST Services.

How Do I Resolve a CORS Error?

A service connection and the REST API it's connecting to must have identical CORS settings. For example, if you're connecting to an Oracle REST Data Services (ORDS) endpoint, either both Visual Builder and ORDS must have CORS enabled, or neither must have CORS enabled.

If there's a mismatch, then the service connection won't work and you'll get an error: `The REST API that you are connecting to doesn't seem to have Cross Origin Request sharing (CORS) for the Visual Builder domain. Try again after adding necessary CORS settings for the REST API you are connecting to.`

To fix this, you can either change the service connection's connection type or enable the service to support CORS for the Visual Builder domain.

Troubleshooting Business Objects

Topics:

- What Is The Maximum Data Limit for Business Objects?
- How Do I Resolve Database Connection Problems?
- How Do I Resolve a "Failed to verify the target database" Error When Switching the Tenant's Database
- How Can I Access Business Object Data?
- How Do I Resolve a "Maximum Number of Sessions Exceeded" Error?

What Is The Maximum Data Limit for Business Objects?

Refer to these frequently asked questions and answers for help retrieving and posting large volumes of data to and from Visual Builder.

1. *Can Visual Builder business objects (BO) hold 220K to 250K records?*
   Yes, they can hold that number of records.
2. *Are there performance challenges in accessing data from BOs of this size?*
   Depends on what performance you are hoping to achieve. A query on a table with that number of records will, of course, be slower than a query on a table with fewer records. This may still be fast enough for your needs. The way to verify it is to test the performance with your specific data and scenario.
3. *Can I load data of about 250K in one single load using REST API PUT/POST operations?*
   You can use the batch approach to load this much data. However, you might also want to look into loading the data directly using the data import APIs. See Import Data from the Command Line.
4. *Can I retrieve data of this size using the GET operation? Is there a better way to extract 250K of data?*
   Get operations will get you the latest data from your BO. If you need to export all the data to another system, consider using the data export APIs. See Export Data from the Command Line.
5. *What is the maximum volume that individual BOs can hold, given a size of ten columns? What would Oracle suggest to limit the maximum volume in a business object?*
   This depends on the amount of data in each column. The database is limited to 5GB so the calculation would be something like 5GB/the amount of data in a row.

How Do I Resolve Database Connection Problems?

If you see messages like `Error with code: 500 occurred while performing request to : /ic/builder/resources/application/applist` and `Problem Processing Request A internal problem processing the request with identified with hash` in your browser's log files, it's likely that the database schema is locked. Ask your database administrator to unlock it using SQL/Plus or SQL Developer.

How Do I Resolve a "Failed to verify the target database" Error When Switching the Tenant's Database

If you are a tenant Administrator and you get an error `Failed to verify the target database.` in the Change Tenant Database dialog when switching the database used by Visual Builder, it might be because the database is not reachable, or because you don't have the required privileges.

To resolve the error, try the following:

1. Confirm the target database is publicly accessible. Visual Builder cannot reach databases in private subnets.
2. Create the ADMIN user (`adminuser`) and grant the user the required roles:

```
CREATE USER [adminuser] IDENTIFIED BY [password];
GRANT CONNECT, RESOURCE, DBA TO [adminuser];
GRANT SELECT ON SYS.DBA_PROFILES TO [adminuser] WITH GRANT OPTION;
GRANT SELECT ON SYS.DBA_USERS TO [adminuser] WITH GRANT OPTION;
GRANT SELECT ON SYS.DBA_DATA_FILES TO [adminuser] WITH GRANT OPTION;
GRANT SELECT ON SYS.DBA_SEGMENTS TO [adminuser] WITH GRANT OPTION;
```

3. Assign the SYSOPER and SYSDBA roles to the ADMIN user (`adminuser`):

```
GRANT SYSOPER, SYSDBA TO [adminuser];
```

You can run the following query to confirm the ADMIN user has the necessary privileges:

```
select * from v$pwfile_users;
```

For details on switching the tenant's database, see Switch to Your Own Oracle DB Instance in *Administering Oracle Visual Builder in Oracle Integration*.

How Can I Access Business Object Data?

If you have trouble accessing business object data, it might be because you need to add the service URL to the list of domains that are allowed to exchange data with your applications.

You might see any of the following issues when trying to access business object data:

- When trying to access the business object **Data** tab, an error message similar to this pops up:

```
[Client UUID: xxxxxxx][ECID Headers: Not present] : error
```

- Status code `403` is displayed when tracking the API call from network tab in the developers console.
- You aren't able to select business objects from any of the action chains.

To resolve this, an Admin needs to add `https://<SERVICE_NAME>-<IDENTITY_DOMAIN>.developer.ocp.oraclecloud.com` to the list of domains that are allowed to exchange data with applications in your instance. See Allow Other Domains Access to Services.

How Do I Resolve a "Maximum Number of Sessions Exceeded" Error?

If you see this error when connecting to an Always Free Autonomous Database, then you've reached the maximum limit of simultaneous database sessions.

To see how many sessions are associated with the database, check:

```
# select status, count(*) from v$session group by status;
```

You can avoid this error by upgrading to the paid service to obtain more resources for your Autonomous Database. See Upgrade an Always Free Autonomous Database to a Paid Instance in *Using Oracle Autonomous Database Serverless*.