Oracle® Cloud Extending Oracle Cloud Applications with Visual Builder Studio





Oracle Cloud Extending Oracle Cloud Applications with Visual Builder Studio, Release 25.07

G25890-02

Copyright © 2022, 2025, 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 The Essentials

What Can You Do with Oracle Visual Builder Studio?	
Which Tool Do I Use to Customize?	
What Is an Extension?	
What Is a Workspace?	
What Is the Designer?	
What Are Dependencies?	
What Are Dynamic Components?	
What Are Fragments?	
How Do I Use My Sandbox in Visual Builder Studio?	
Visual Builder and Visual Builder Studio: What's the Difference?	
What's the Extension Lifecycle?	
How Are Extensions Applied at Runtime?	
11	
Extension Best Practices Get Started	
Extension Best Practices Get Started	
Extension Best Practices Get Started What Do You Want To Do in VB Studio?	
Extension Best Practices Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application	
Extension Best Practices Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application	
Extension Best Practices Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application Create an Extension Collaborate on an Extension	
Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application Create an Extension Collaborate on an Extension Clone an Existing Repository	
Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application Create an Extension Collaborate on an Extension	
Extension Best Practices Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application Create an Extension Collaborate on an Extension Clone an Existing Repository Import an Extension Archive Use a Scratch Repository	
Extension Best Practices Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application Create an Extension Collaborate on an Extension Clone an Existing Repository Import an Extension Archive Use a Scratch Repository Add a New Page to an Oracle Cloud Application	
Extension Best Practices Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application Create an Extension Collaborate on an Extension Clone an Existing Repository Import an Extension Archive Use a Scratch Repository	
Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application Create an Extension Collaborate on an Extension Clone an Existing Repository Import an Extension Archive Use a Scratch Repository Add a New Page to an Oracle Cloud Application Add an Application to the Oracle Cloud Applications Ecosystem	
Get Started What Do You Want To Do in VB Studio? Configure an Oracle Cloud Application Create an Extension Collaborate on an Extension Clone an Existing Repository Import an Extension Archive Use a Scratch Repository Add a New Page to an Oracle Cloud Application Add an Application to the Oracle Cloud Applications Ecosystem Add a Resource to the Oracle Cloud Applications Ecosystem	



	The Properties Pane	2-32
	Variables Picker	2-33
	Expression Editor	2-34
Pa	rt II Build an Extension	
3	Extension-Level Actions	
	Establish Extension-Level Settings	3-1
	Migrate Runtime Dependencies	3-3
	Resolve Upgrade Issues	3-4
	Export Your Workspace as an Archive	3-5
	Import Resources to Your Workspace	3-6
	Migrate Imported Resources	3-6
	Switch a Workspace	3-7
	Switch a Workspace's Environment	3-7
4	Manage Your Extension with Git	
	Get Oriented with Git	4-1
	Pull Changes from the Remote Branch	4-3
	Make Your Changes Public	4-3
	Commit Your Changes to the Local Branch	4-4
	Push Your Changes to the Remote Branch	4-5
	File-Management Operations	4-5
	Refresh Your Workspace	4-8
	Resolve Conflicts Using the Git Panel	4-10
	Use the Context Menu to Resolve Conflicts	4-13
	Use the Conflict Editor to Resolve Conflicts	4-14
	Stash Your Changes	4-16
	Create a Stash	4-16
	Apply a Stash	4-17
	Delete a Stash	4-18
	Revert Commits	4-20
	Cherry-Pick Commits Between Branches	4-22
	Get the Revision ID of a Commit	4-22
	Cherry-Pick a Commit From Another Branch	4-23
	Create or Switch a Branch	4-25
	Rename a Branch	4-27
	Delete a Local Branch	4-28
	Merge Remote Branches	4-28
	- 5	7 20



The Data Palette

2-30

	Push a Scratch Repository to a Remote Repository	4-30
	Using Branches with a Sandbox	4-30
	View Git History	4-31
	How Do Sandboxes Relate to Git Branches?	4-32
	Switch a Sandbox	4-33
	Refresh a Sandbox	4-36
	Disassociate a Sandbox	4-36
	Publish a Sandbox	4-37
5	Work With Services	
	Manage Service Connections	5-2
	What Are Service Connections?	5-2
	Service Connections: Static Versus Dynamic	5-3
	Create a Service Connection	5-5
	Create Service Connections from the Oracle Cloud Applications or Integration Applications Catalog	5-6
	Create a Service Connection from a Service Specification	5-8
	Create a Service Connection from an Endpoint	5-12
	Edit a Service Connection	5-18
	Manage Service Endpoints	5-19
	Retrieve Service Metadata for a Dynamic Service Connection	5-24
	Add Server Variables for Service Connections	5-26
	Add Transforms	5-27
	Convert a Service Connection (Static to Dynamic or Dynamic to Static)	5-38
	Test Service Connection Responses	5-40
	Update Schema of the Request or Response	5-44
	Manage Backends	5-46
	What Are Backends?	5-46
	Create a Backend	5-50
	Create a Custom Backend	5-50
	Create a Child Backend	5-53
	Edit a Backend	5-53
	Edit a Backend Server's Authentication Details	5-55
	Add Server Variables for Backends	5-56
	Add a Local Server to Use a Different Backend Definition During Development	5-59
	Set the Backend's Authentication Method and Connection Type	5-62
	How Does the Identity Propagation Authentication Method Work?	5-62
	Edit a Server's Authentication Details	5-64
	Sample Use Case: How to Use Authentication to Connect to Oracle Integration Cloud REST APIs	5-64



Work With Layout	s in Your Extension	
Exploring a Layout		6
Extending a Layout in a D	Dependency	6
Create a Layout		6
Create a Rule Set in a La	yout	6
Override Field Properties		6-1
Preview, Share, a	nd Publish Your Extension	
Preview Your App UI		7-
Share Your App UI		7-
Customize Optimization		7-
Optimization Configu	rations for build.json	7-
How to Migrate Grun	tfile.js Configuration Into build.json	7-1
Debug and Audit Your Co	ode	7-1
Troubleshoot Build Is	sues	7-1
Suppress Optimization	on	7-1
Change the Log Leve	el	7-1
Publish Your Extension		7-1
Enable or Disable the	e CI/CD Pipeline for Publishing	7-2
Publish Your Change	s Without CI/CD Pipelines	7-2
Publish Your Change	s Through CI/CD Pipelines	7-2
Deploy to Test and P	roduction Instances	7-2
Create or Renew OA	uth Tokens to Deploy Your Extension	7-3
Impact of P2T on Ext	rensions	7-3
Deploy an Extension	From Your Local System	7-3
Specify a Different Ve	ersion for the Extension	7-3
What If Dev and Prod	Instances Aren't On the Same Release?	7-3
View Your Deployments		7-3
Manage Your Published I	Extensions	7-3
Roll Back a Published Ex	tension	7-4
Build and Package Your	Extension Manually	7-4
Build Your Extension	Using Grunt	7-4
Deploy Your Extension	on Using Grunt	7-4
Work With Transla	ations	
What are Translation Res	sources?	8-
Create a Translation Bun	dle	8-



	Create Translation Keys	8-4
	Associate a Translation Key with a UI Component	8-6
	Expressions in Translatable Strings	8-9
	Use an Expression in a Translatable String	8-9
	Override a Translation Key Value	8-13
	Download and Upload Translation Bundles	8-14
Part	III Configure an App UI	
9	Customize an App UI	
	How Tos	9-4
10	Customize Dynamic Tables and Forms	
	Control Your Display with Rule Sets	10-3
	Determine What's Displayed at Runtime With a Rule Set	10-3
	Open a Rule Set From Oracle Cloud Applications	10-4
	Edit a Component's Layout in the Properties Pane	10-6
	Open a Rule Set From Visual Builder Studio	10-8
	Create a Rule in a Rule Set	10-10
	Create a Rule Based on User and Device	10-14
	Create a Layout in a Rule Set	10-15
	Edit a Field's Properties in a Layout	10-17
	Use Conditions to Hide and Show Fields in a Layout	10-19
	Configure How Columns Are Rendered in a Dynamic Table Layout	10-22
	Set a Field to Display as a Text Area in a Form	10-23
	Set How User Assistance is Rendered in a Layout	10-25
	Control How a Field is Rendered with Templates	10-27
	Create a Field Template	10-27
	Apply a Template to a Field	10-31
	Set the Default Template for a Field in a Layout	10-33
	Add and Group Fields in Dynamic Form Layouts	10-36
	Add a Link to a Page in an App UI	10-37
	Create Templates for Form Layouts	10-39
	Apply a Template to a Form	10-44
	Create Fields For a Layout	10-45
	Create a Virtual Field	10-50
	Add Converters and Validators to Fields	10-52
	Use Context Parameters in Extensions	10-55
	Work with Polymorphic Objects in a Layout	10-59
	Control Your Display with Business Rules	10-62



	what are Business Rules?	10-63
	Work with Business Rules	10-65
	Create a Rule For Forms	10-65
	Create a Rule For Tables	10-68
	Use Nested Rules	10-73
	Add an If/Else Rule	10-76
	Add a Switch Rule	10-78
	Filter Your Rules	10-80
	Set Conditions for a Rule	10-83
	Override Field Properties in a Form	10-94
	Understand What Will Be Shown at Runtime	10-99
	Display Messages in a Form	10-103
	Create a Rule to Validate a Field	10-105
	Use Templates with Validation Rules	10-110
11	Customize Dynamic Containers	
	How Do Cases Work?	11-2
	Configure a Dynamic Container	11-2
	Guidelines for Working with Container Sections	11-10
	Add Content From a Custom Object to Your Page	11-10
	Extend a Container in a Fragment	11-14
12	Customize Variables and Constants	
	Change the Value of an Extendable Constant	12-3
	Hide and Show an Element Using a Constant	12-7
	Create and Edit Variables and Constants	12-9
	Set Property Values Using Rules	12-11
13	Trigger Actions in Dynamic Components	
	Trigger Actions in Layouts	13-2
	Define Behavior for Components in Layouts	13-2
	Start an Action Chain From an Action Chain	13-6
	Start an Action Chain from a Field or Variable	13-7
	Use Lifecycle Events to Start Action Chains in a Layout	13-11
	Call Custom JavaScript Functions	13-13
	Use Events Defined in a Dependency	13-14
	Trigger Actions in Dynamic Containers	13-16



Work With Fragments From Dependencies	
What are Referenceable and Extendable Fragments?	14-2
Add a Fragment From a Dependency	14-3
Extend a Fragment in a Dynamic Form or Table	14-6
Extend a Fragment in a Dynamic Container	14-7
t IV Ruild an Ann III ar Fragment	
t iv Build an App of of Fragment	
Develop an App UI or Fragment	
Create an App UI or Fragment	15-2
How Are App UIs Structured?	15-5
What Are Scopes?	15-6
Add a Dependency	15-7
Establish App UI Settings	15-13
Brand Your App UI with a Custom Root Page	15-13
Control Access to Your App UI	15-15
Work With Pages and Flows	
Create and Manage Pages	16-2
Manage Page Settings	16-3
Set a Page's Layout	16-4
Create Pages From Templates	16-7
Create Pages From Patterns	16-8
Create Pages From Fragments	16-9
Change Page Templates	16-12
Create and Manage Flows	16-13
Manage Flow Settings	16-14
Embed a Flow Within a Page	16-16
Display SaaS Data In Your App UI	16-18
Navigate Between Pages and Flows	16-22
Navigate Between Pages in the Same Flow	16-22
Navigate Between Pages in Different Flows	16-23
Navigate Between Pages in Different App UIs	16-26
Navigate From Fragments and Layouts to App UIs	16-28
Work With Components	
Add Components to Pages	17-1
Add a Component Using Code Completion	17-5
	What are Referenceable and Extendable Fragments? Add a Fragment From a Dependency Extend a Fragment in a Dynamic Form or Table Extend a Fragment in a Dynamic Container IV Build an App UI or Fragment Develop an App UI or Fragment Create an App UI or Fragment How Are App UIs Structured? What Are Scopes? Add a Dependency Establish App UI Settings Brand Your App UI with a Custom Root Page Control Access to Your App UI Work With Pages and Flows Create and Manage Pages Manage Page Settings Set a Page's Layout Create Pages From Templates Create Pages From Patterns Create Pages From Patterns Create Pages From Fragments Change Page Templates Create and Manage Flows Manage Flow Settings Embed a Flow Within a Page Display SaaS Data In Your App UI Navigate Between Pages in the Same Flow Navigate Between Pages in Different Flows Navigate Between Pages in Different App UIs Navigate From Fragments and Layouts to App UIs Work With Components Add Components to Pages



Work With JET Core Pack Components	17-6
Bind a Component to Data	17-8
How Do Quick Starts Work?	17-9
Add an Image to a Page	17-11
Add an Icon to a Page	17-13
Add a Camera Component to a Page	17-14
Filter Data Displayed in a Component	17-18
Filter Data by Filter Criteria	17-19
Filter Component Data by Text	17-21
Filter Component Data by URL	17-23
Use Conditions to Show or Hide Components	17-26
Use Converters to Change Date and Time Formats and Time Zones	17-30
Add Dynamic Components to Pages	17-35
Add a Dynamic Table to a Page	17-37
Add a Dynamic Form to a Page	17-41
Determine What's Displayed at Runtime With Rule Sets	17-45
Create a Layout for a Dynamic Table or Form	17-51
How To Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null	17-54
Use Field and Form Templates	17-55
Control How a Field is Rendered with Field Templates	17-55
Apply a Template to a Field	17-57
Start an Action Chain from a Field	17-60
Control How a Form Layout is Rendered	17-63
Apply a Template to a Form	17-67
Add and Group Fields in Dynamic Form Layouts	17-69
Use Business Rules With Your Rule Sets	17-71
Create a Rule for Forms	17-73
Create a Rule for Tables	17-79
Include and Exclude Rule Sets in Business Rules	17-81
Create Templates for Business Rules	17-82
Edit a Field's Properties	17-84
Set a Field to be Read Only	17-84
Set How User Assistance is Rendered in a Layout	17-86
Set a Field to Display as a Text Area in a Form	17-87
Set a Field as Required	17-90
Specify How a User Input Overrides a Field Value	17-93
Map a Field to a Global Field Template	17-94
Use Conditions to Show or Hide Fields in a Layout	17-95
Control How Fields With a Null Value are Displayed in Forms	17-97
Configure How Columns Render in a Dynamic Table's Layout	17-98
Add Converters and Validators to a Field	17-100



	Create Fields For a Layout	17-103
	Create a Calculated Field	17-104
	Create a Virtual Field	17-107
	Add a Dynamic Container to a Page	17-109
	Add Fragments as Sections in a Dynamic Container	17-116
	Re-Ordering a Container's Content	17-120
	Guidelines for Working with Container Sections	17-121
	Change a Dynamic Container's Layout	17-121
	Make a Container Available to Extensions	17-124
	Make a Rule Set Extendable	17-126
	Add Custom Web Components to Pages	17-128
	Install a Web Component from the Component Exchange	17-130
	Update a Web Component from Component Exchange	17-132
	Import a Web Component Using a ZIP Archive	17-133
	Create a Web Component	17-135
	Uninstall a Web Component	17-136
18	Work With Variables, Types, and Constants	
	What are Variables?	18-1
	Create Variables	18-6
	Enable Variables as Input Parameters	18-12
	Track Variables to Detect Unsaved Changes	18-16
	Create Variables to Temporarily Store Data Changes in a Buffer	18-18
	Create Types	18-20
	Create a Custom Object or Array Type	18-21
	Create a Type From an Endpoint	18-24
	Create a Type From Code	18-26
	Make Variables and Types Available to Extensions	18-31
	Configure How Variables are Customized in the Properties Pane	18-33
	Organize How Constants Are Listed in the Properties Pane	18-48
	Service Data Provider	18-51
	Creating a Custom Fetch Action Chain - An Example	18-51
	Delay Display of SDP Data	18-60
19	Work With JavaScript Action Chains	
	About Action Chains	19-3
	About the Action Chain Editor	19-5
	Create Action Chains in Design View	19-7
	Create Action Chains in Code View	19-10
	About the Action Chain Code	19-12



Visually Create an Action Chain	19-15
Built-In Actions	19-29
Add an Array Operation Action	19-31
Add an Assign Variable Action	19-32
Use Filter Builder to Create Filter Criteria for an SDP	19-34
Filter Builder's Code Editor	19-37
Add a Call Action Chain Action	19-37
Add a Call Component Action	19-39
Add a Call Function Action	19-39
Add a Call REST Action	19-41
Service Definitions	19-45
Transform Functions	19-45
Add a Call Variable Action	19-45
Add a Code Action	19-46
Add a Fire Data Provider Event Action	19-46
Add a Fire Event Action	19-49
Add a Fire Notification Action	19-51
Add a For Each Action	19-53
Add a Get Dirty Data Status Action	19-54
Add a Get Location Action	19-62
Add an If Action	19-63
Add a Navigate Back Action	19-66
Add a Navigate To Application Action	19-66
Add a Navigate To Flow Action	19-68
Add a Navigate To Page Action	19-70
Add an Open URL Action	19-72
Add a Reset Dirty Data Status Action	19-73
Add a Reset Variables Action	19-74
Add a Return Action	19-74
Add a Run In Parallel Action	19-75
Add a Scan Barcode Action	19-79
Add a Share Action	19-81
Add a String Operation Action	19-82
Add a Switch Action	19-83
Add a Try-Catch Action	19-86
Custom Actions	19-89
Create a Custom Action	19-89
Create the Action Files	19-89
Add the Metadata	19-91
Add the Code	19-95
Specify Path to Code	19-97
Start an Action Chain	19-97



Start an Action Chain When a Variable Changes	19-101
Start an Action Chain From a Lifecycle Event	19-102
Start an Action Chain By Firing a Custom Event	19-105
Test Action Chains	19-110
Create a Test for a Test Case	19-113
Run the Tests	19-119
Use the Tests Footer in Your App UI	19-120
Work With JSON Action Chains	
What is an Action Chain?	20-2
Create an Action Chain	20-3
Built-in Actions	20-10
Add an Assign Variables Action	20-11
Filter Builder's Code Editor	20-15
Add a Call Action Chain Action	20-16
Add a Call Component Action	20-18
Add a Call Function Action	20-20
Add a Call REST Action	20-21
Add a Call Variable Action	20-25
Add a Fire Data Provider Event Action	20-26
Add a Fire Event Action	20-28
Add a Fire Notification Action	20-28
Add a Get Location Action	20-30
Add a Reset Variables Action	20-31
Add a Scan Barcode Action	20-32
Add a Share Action	20-35
Add a Take Photo Action	20-36
Add a For Each Action	20-37
Add an If Action	20-40
Add a Return Action	20-42
Add a Run In Parallel Action	20-43
Add a Switch Action	20-44
Add a Navigate Action	20-45
Add a Navigate Back Action	20-47
Add an Open URL Action	20-48
Custom Actions	20-49
Create a Custom Action	20-49
Create the Action Files	20-49
Add the Metadata	20-51
Add the Code	20-55

Start an Action Chain From a Component



19-97

	Specify Path to Code	20-57
	Test Action Chains	20-57
	Manage All Tests in Your App UI	20-59
	Start an Action Chain	20-60
	Start an Action Chain From a Component	20-60
	Start an Action Chain When a Variable Changes	20-64
	Start an Action Chain By Firing a Custom Event	20-65
	Start an Action Chain From a Lifecycle Event	20-66
21	Work With Events and Event Listeners	
	Define Events in Your App UI	21-1
	Create Event Listeners for Events	21-4
	Choose How Custom Events Call Event Listeners	21-7
	Enable Asynchronous Handling for Component Events	21-8
	Raise Fragment or Layout Events that Emit to the Parent Container	21-10
	Use Events Defined in the Unified App	21-13
	Make an Event Available to Extensions	21-14
22	Work With Resource Files	
	Import Resources	22-2
	Work with the Image Gallery	22-4
	Create Declarative References to Imported Resources	22-7
23	Work With Code	
	Work With JavaScript	23-1
	Add a Custom JavaScript Function	23-2
	Use Variables with a JavaScript Module	23-6
	Work With Third-Party JavaScript Libraries	23-6
	Use RequireJS to Reference Third-Party JavaScript Libraries	23-7
	Add JavaScript Modules As Global Functions	23-8
	Manage Global Functions	23-12
	Example JS Module Defined As a Global Function	23-16
	Example functions.json to Declare Global Functions	23-16
	Work With JSON	23-19
	Trigger Code Insight	23-20
	Manage Code Editor Settings	23-21
	Files in Source View	23-26



24 Work With the Diagram View

View a Flow's Navigation in Diagram View	24-2
Add Pages and Action Chains to a Flow in Diagram View	24-2
Add a Page in the Flow Diagram	24-3
Create an Action Chain in the Flow Diagram	24-6
Bind an Action Chain in the Flow Diagram to an Existing Event Listener	24-10
Show or Hide an Action Chain in the Flow Diagram	24-11
Work With Fragments	
Create and Add a Fragment to a Page	25-1
Define Actions and Events in Fragments	25-6
Manage Fragment Settings	25-7
Reuse a Fragment	25-9
Pass Data Between a Fragment and Its Parent Container	25-10
Enable Fragment Variables as Input Parameters	25-11
Enable Page Variables to Provide Initial Values for a Fragment's Input Parameters	25-14
Bind Fragment Input Parameters to Page Constants	25-17
Automatically Write Back a Fragment Variable's Value to Its Container Variable	25-20
Automatically Create and Wire a Fragment Variable on Its Container	25-22
Sample Scenario: Create a Fragment and Pass Values	25-25
Create Custom Events that Emit to a Fragment's Parent Container	25-38
Automatically Wire a Fragment's Custom Event to the Parent Container	25-45
Set the Binding Type for Variables in Dynamic Components	25-48
Pass a Fragment's Context to VDOM or Custom Web Components	25-49
Defer Rendering of a Fragment's Content	25-50
Make a Fragment Available to Other Extensions	25-53
Add Slots to a Fragment	25-55
Add Default Content to a Fragment Slot	25-61
Set Data Context for a Fragment Slot	25-61
Customize How Fragment Properties Display in the Properties Pane	25-64
Customize How a Fragment Variable is Displayed in the Properties Pane	25-64
Section Fragment Properties for Display in the Properties Pane	25-82
Common Use Cases	
Populate a List of Values Based on Another LOV	26-1
Selectively Enable a Field	
•	26-9
Selectively Display a Field Add Offline Support for Your App UI	26-9 26-16 26-28



Part V Troubleshooting

27	Trouble	shooting	and	FAQs
----	---------	----------	-----	------

Resolving the Error "Page cannot be previewed"	27-1
How Do I Clear My Extension's Resource Cache?	27-4
How Do I Recover My Extension After Deleting the Project?	27-4
Resolve the "Loading Completed with Errors" Message	27-6

Part VI Appendix



Preface

Extending Oracle Cloud Applications with Visual Builder Studio describes how to use a webbased visual development tool to create and configure application extensions to customize Oracle Cloud Applications.

Topics:

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

Audience

Extending Oracle Cloud Applications with Visual Builder Studio is intended for Oracle Cloud Applications users who want to create, edit and publish their application extensions.

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 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, VB Studio does <i>not</i> support shortcuts like the ones described here.



Keyboard Shortcuts for Code Editors

To help you work efficiently, VB Studio supports the following keyboard shortcuts when working with code editors in the Designer (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		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

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

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

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 deployed App UI, 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 App UI 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 deployed App UI:

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.



Part I

The Essentials

Acquaint yourself with the most essential concepts related to customizing Oracle Cloud Applications (or building your own App UIs), then learn how to create a workspace so you can start working on your extension.

Topics:

- The Basics
- · Get Started



1

The Basics

Welcome! To use Oracle Visual Builder Studio (or VB Studio, for short) effectively, you should know the answers to the questions addressed in this chapter. Even if you've extended Oracle Cloud Applications in the past, we recommend at least skimming through this material to make sure you're equipped to use the most recent version of VB Studio to its fullest potential.

- What Can You Do with Oracle Visual Builder Studio?
- What Is an Extension?
- What Is a Workspace?
- What Is the Designer?
- What Are Dependencies?
- What Are Dynamic Components?
- What Are Fragments?
- How Do I Use My Sandbox in Visual Builder Studio?
- What's the Extension Lifecycle?

This chapter presents the basic concepts that underlie VB Studio. For how-to information, click the links sprinkled throughout each conceptual topic. Or, if you want to dive right in, go straight to Get Started, or just get a high-level overview by reading What's the Extension Lifecycle?

What Can You Do with Oracle Visual Builder Studio?

VB Studio gives you the power to customize Oracle Cloud Applications to suit your company's specific business needs. In fact, you'll have the exact same tools at your disposal—Visual Builder Studio and Oracle Javascript Extension Toolkit (JET)—that internal Oracle developers use to create Oracle Cloud Applications in the first place, giving you an unprecedented level of power and control over your Oracle Cloud Apps ecosystem.

Take a look at some of the common use cases you can easily achieve with VB Studio:

- In the Digital Sales application, you want the **employees**' view of a given page to be exactly as it came from Oracle, but you want **managers** to see a field that doesn't even exist in Oracle's original app—one that computes the sales rep's commission based on the sales amount for each completed deal. You simply create a layout that includes a *custom field* to calculate the commission amount for the managers' view, then create a *rule* that displays that layout only when the user is a manager.
- Suppose you need to add new content to a page in an Oracle Cloud Application. If there's a *dynamic container* on that page that's marked as extensible, you can use that container to display things like text fields, images, or even a button that starts a chain of events (known as an action chain). You can even include data from custom objects you created in Application Composer in your App UI, and make certain fields customizable by your users.
- Need your own set of pages? Create an App UI, which will look and feel exactly like all the
 other Oracle Cloud Applications in your instance. You can link to this App UI from an
 existing Oracle Cloud Application, to make it appear as though your custom pages are part

of the app itself, or make the App UI accessible as a separate module from Oracle's new Ask Oracle menu.

- On another page in the Digital Sales app, you want an error to appear if a user enters a
 number less than 100 or greater than 10,000 in the Discount field, even though the original
 field contains no such value checking. Just set a simple field *validator* on the Discount
 field, and any number outside that range will be instantly flagged.
- You know that the users who will be customizing the Oracle Connections app for your firm
 will need access to confidential employee information provided by a REST API that is
 usually tightly controlled. To prevent each customizer from having to request access to this
 REST API, you create a service connection to this data in an extension, thereby allowing
 each person who needs the data to gain access to it simply by adding the extension as a
 dependency. (Note that while a service connection does provides access to data, it does
 not impose any kind of data security.)

All of these scenarios can be implemented in VB Studio by creating an extension for one or more App UIs, which is the primary focus of this guide. However, some scenarios require users to create a bespoke *responsive application* instead of an App UI, particularly when they require certain capabilities that App UIs don't currently provide. You might consider this option if you need to:

- Access REST data sources with advanced authentication protocols;
- Create custom business objects in your own database;
- Run an application outside of Oracle Cloud Applications;
- Provide application access to users who don't have Oracle Cloud Application credentials:
- Create a customized look and feel.

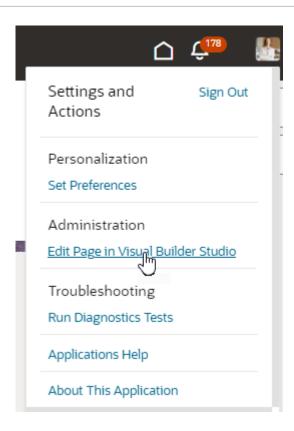
If you have one or more of these requirements, you can create a responsive application (using either Visual Builder or Visual Builder Studio) and deploy it to a standalone Visual Builder runtime environment. Just like App UIs, a responsive application can access Oracle Cloud Application data through the service catalog, as well as leverage Redwood templates and components. If desired, you can even embed such applications into an Oracle Cloud Application or other web site, using the technique described in Embed a Web App in an Oracle Cloud Application.

Which Tool Do I Use to Customize?

The tool you use to customize an Oracle Cloud Application is determined by the application you want to extend.

Configuring and Extending Applications describes how you can use Oracle Cloud Application tools to customize many Oracle Cloud Applications to meet your business needs. However, if you see Edit Page in Visual Builder Studio while viewing a page in Oracle Cloud Applications, that's your signal that you need to use VB Studio to make your changes to the App's user interface, rather than a tool native to Oracle Cloud Applications. You'll see this option in the Settings and Actions menu:





All you have to do is click the link to jump over to the Designer in VB Studio, where you can start customizing your App. Depending on how Oracle built your Oracle Cloud Application, you may land in *Express mode*, a curated design experience for less technical users. If you want to use the full power of the Designer, click the **Advanced** toggle in the Designer's header, then continue using this guide. If you're comfortable in Express mode, see What Can You Do with Visual Builder Studio in Express Mode? in *Extending Oracle Cloud Applications in Visual Builder Studio Express Mode* instead.

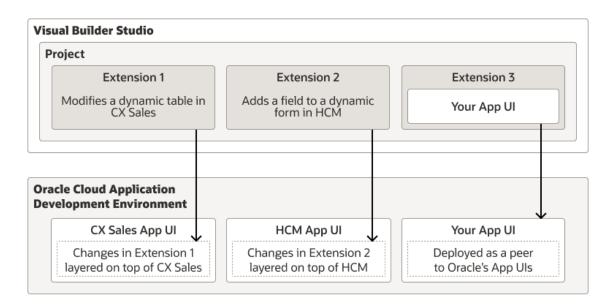
Note:

Use the Application Composer to make any underlying data model changes you may need *before* you use Visual Builder Studio to expose those changes in the user interface. Configuring and Extending Applications can help guide you through that process.

What Is an Extension?

Extensions are what you use to deliver new capabilities into Oracle Cloud Applications. Those capabilities may take the form of customizations you make to the App's user interface, or your extension may include things like App UIs, to deliver new pages or resources to your Oracle Cloud Applications instance.

An *App UI* is simply an application that includes a user interface component in the form of Visual Builder pages and flows. Some App UIs are created by Oracle—like certain Oracle Cloud Applications—but you can build your own App UIs and deploy them as peers alongside Oracle's App UIs in your Oracle Cloud Applications ecosystem, as shown here:



A single extension can contain:

- Configurations you make to one or more Oracle Cloud Applications, and/or;
- One or more App UIs that you create, and/or;
- One or more resources that you want to contribute to the Oracle Cloud Application ecosystem, like a service connection, image, CSS (Cascading Style Sheets), or JavaScript function.

Let's examine each of these options in a bit more detail.

A configuration to an App UI can be as simple as hiding certain fields in a dynamic form or table for certain audiences, to displaying brand new content through the use of a dynamic container. You control what's displayed at runtime on a page through the use of *display logic*, which you configure through *rule sets* or *business rules*, depending on how your Oracle Cloud Application was created. Suppose you want to configure a dynamic table in the CX Sales App UI so that certain columns are hidden and others are added when the user viewing the page is a manager. You'd create a rule that checks for the user's role and, if the user is indeed a manager, apply the layout you've created that contains the desired columns. All non-managers would see the page with the default layout applied. The figure above shows these customizations for the CX Sales App UI, which are applied at runtime. You can learn more about display and layouts in Work With Layouts in Your Extension.

When would you create your own App UI? Suppose you've just created a custom object in App Composer, and now want to allow your users to interact with that object through a brand new set of pages. In other words, you want to create a fully functioning app, just as Oracle did when it created Digital Sales, Help Desk, and others. You've got the same tools that Oracle has—namely, VB Studio and JET—so you can accomplish this task in the exact same way: After creating an extension, you use the Designer to create a new App UI, then add all the pages, page flows, and all the other functionality you need. You could keep these pages as a standalone app for your users—even make them available from the central landing page—or you could enable users to seamlessly navigate between the pages of an Oracle-created app and your custom app simply by adding a *navigation action* between pages.

When might you use an extension to provide access to a commonly needed resource? Suppose you know that the team who plans to customize the Connections App UI at your site will need access to confidential employee information provided by a REST API that is usually tightly controlled. To prevent each developer from having to request access to this REST API,



you create a service connection to this data in an extension, thereby allowing each person who needs the data to gain access to it simply by adding the extension as a dependency.

You may also notice that the image above contains something called a *project*. In VB Studio, you will likely belong to one or more projects, each of which is devoted to a discrete software effort. When it comes to extensions, it's easiest to use **one** project for all the extension work you do within an Oracle Cloud Application environment family. (An *environment family* is defined as all the instances you use for test, development, and production.) Within that project, you can create a separate Git repository to store the source files for each *pillar*—one repo for HCM, another for CRM, and so on. If your team is too large to make that feasible, however, or if you simply want to isolate the extension work for each pillar, you can keep things completely separate by creating a unique project for each Oracle Cloud Application.

Note:

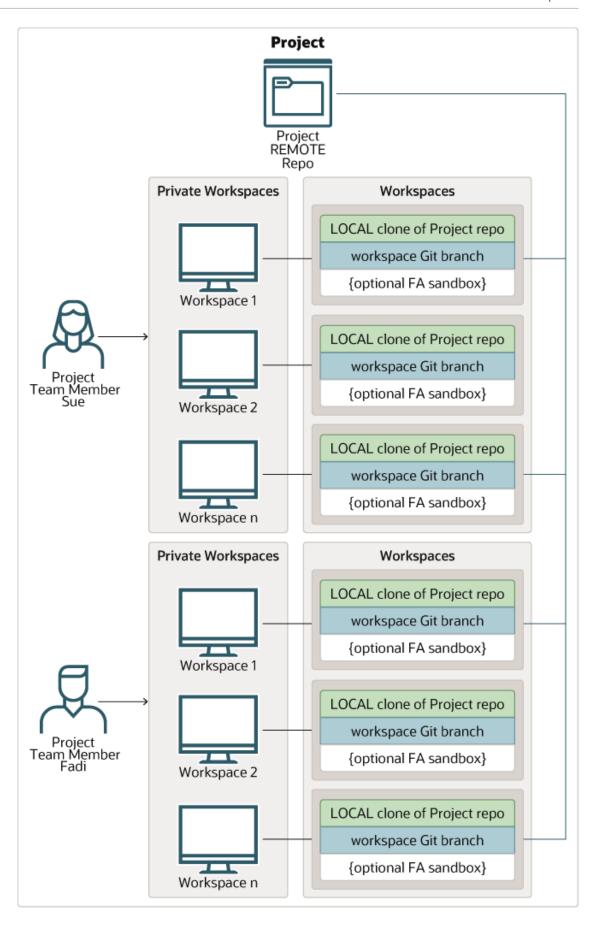
After you deploy your extension to your Oracle Cloud Apps environment, you manage the extension using the environment's Deployment tab; you don't manage the constituent App UIs individually. If it's important to manage a particular App UI individually, you could create a branch in your Git repo for just that App UI and publish it separately.

What Is a Workspace?

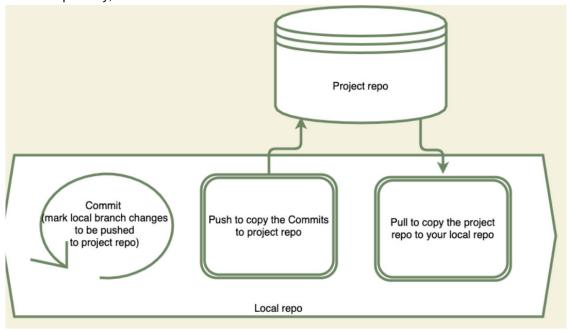
All of your work in VB Studio is done in the context of a *workspace*, a completely private area where you can work on your extension. Before you can use the Designer to create an App UI or configure an Oracle Cloud Application, you must first select or create a workspace, which defines:

- The Git repository where your files will be stored. Physically, you work within a copy of the Git repository, called the **local repo**, and push your changes periodically to the **remote repo**, which belongs to the *project* you're working within.
- The branch containing the source files you want to use.
- The Oracle Cloud Applications environment where you plan to publish your extension.
- A sandbox, if you're using one, which contains data model changes that haven't been
 published yet. (The data model changes that have been published are already available to
 your extension.)



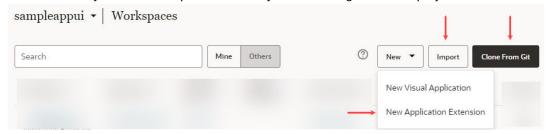


This figure shows how Sue and Fadi, who both belong to the same *project*, can have multiple workspaces focused on different extensions. VB Studio <u>projects</u> are usually defined by an administrator, one per discrete software effort. For example, you might have one project for building a bespoke visual application, and another project for all the configurations your team is making to Oracle Cloud Applications. In this figure, notice how each workspace uses separate branch within a clone (the local repo) of the project's Git repository (the remote repo), periodically using standard Git commands like Commit, Push, and so on to copy updates to the remote repository, like this:



If you're not familiar with working in Git, this video will help you learn the basics: Video When working within a workspace, keep the following in mind:

• The development environment must already be set up and defined in the VB Studio project before you can create a workspace. You have several options for creating a workspace; which method you choose depends on how you're working within the project:

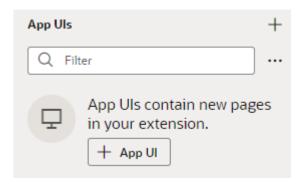


See Get Started for guidance on when to create a new workspace (that is, a new extension), when to import, and when to clone.

- Depending on your needs, you may have one single project for all your Oracle Cloud Application configurations, with a different Git repo dedicated to each pillar—like HCM, CRM, and so on. Or you might have things separated by pillar, with one whole project dedicated to each pillar's extension work. Regardless, each time you click Edit Page in Visual Builder Studio while looking at an Oracle Cloud Applications page, VB Studio checks to see if there's already a project started with a Git repository for that Oracle Cloud App. If there is, VB Studio clones the Git repo as the basis for your workspace.
- Your workspace is based on the Oracle Cloud Application you were viewing when you clicked Edit Page in Visual Builder Studio to jump over to Visual Builder Studio. If you

choose **New Application Extension** to create a new App UI (essentially, creating an empty extension), your base application is the **Unified Application**, which underlies *all* App UIs in the Oracle Cloud Application ecosystem. This doesn't mean you're limited to working on a brand new App UI in your extension. If you later decide that you want to configure an App UI within the same workspace, you can always add that App UI as a *dependency*, then create a new branch in your workspace to make those changes. (You add something as a dependency when you want to a) configure it (that is, make changes to it), or b) access its resources, like service connections or Layouts, while building your own App UI.)

Likewise, if your workspace is originally based upon an Oracle Cloud Application that is also an App UI, like Digital Sales, you can still create a brand new App UI in your extension by clicking **+ App UI**:

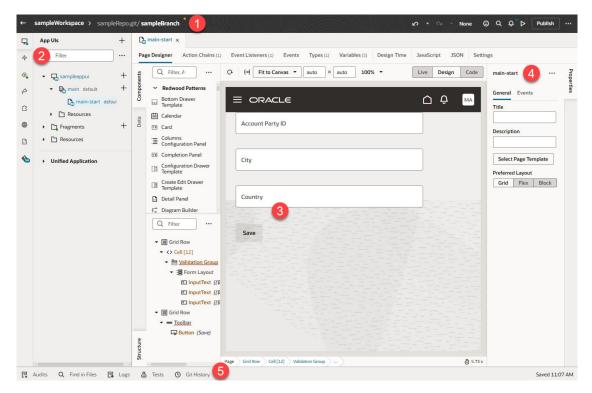


You're the only one who can access your workspace. Changes to files you make in your workspace aren't visible to other team members until you a) merge them to the project's Git repo, b) choose to share them with others for testing, or c) Publish them. See Preview, Share, and Publish Your Extension for more.

What Is the Designer?

Much of what you do in VB Studio is within the context of an extension, and almost everything you do within an extension takes place within the Designer. The Designer is divided into five main areas:





- 1. Header
- 2. Navigator
- 3. Canvas/Editors
- Properties pane
- Footer

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

Note:

If you've used the Designer to create *visual applications*, you will notice slight but important differences between that version of the Designer and the one you use to work with App UIs.

The Header

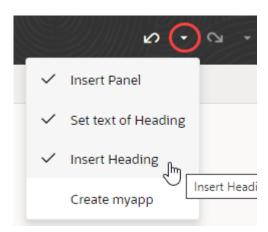
The header contains information about your current VB Studio session, as well as access to the tools you need to move your extension through the development process.



Here's what each element does:



Label	Element	Description
A	Workspace	The name of your current workspace, which defines all the resources you need to create an extension. You may have several active workspaces at any one time, one for each discrete body of work you're responsible for. To switch to a new workspace, just click the workspace name and pick a new one.
В	Git repository/ branch	VB Studio stores all the files for your extension in a Git repository. If you're working with multiple people on the same extension, you'll all work within the same repository, possibly—but not necessarily—in separate branches. If you see a yellow dot in the header next to the repository name, that means there are uncommitted changes in your workspace. Click the repo name to see a list of commands you can use while working with your repository.
		If you don't know much about Git repositories, this video can help you learn the basics, or read through Manage Your Extension with Git.
С	Undo	Undo one or more of your changes. To undo your most recent change, click the Undo icon (hover your cursor over the icon to view the action that will be undone). To undo multiple changes, click the Undo dropdown 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	Element	Description
D	Redo	Redo one or more changes after undoing them. To redo your most recent change, click the Redo icon (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:
		✓ Insert Heading
		✓ Set text of Heading
		Insert Panel
E	Sandbox	If you require any underlying data model changes, you should use Application Composer to make them first in a sandbox, then use the Edit Page in Visual Builder Studio option to expose those changes in the UI with VB Studio. Click the sandbox name to associate a new sandbox with your workspace. You might want to do this if another sandbox contains the data you need for the extension you're working on. See Switch a Sandbox.
F	Feedback	Submit your feedback about VB Studio to Oracle.
G	Go to File	Search your project'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.
H	Notifications	View notifications received over the past two days. To get your attention, notifications pop up instantly in the bottom right corner of your window, where info, confirmation, and warning messages clear automatically after five seconds but error messages stay until they are manually cleared. If you've missed a notification or want to review one again, click Notifications in the header to view all notifications from the last two days. Notifications older than two days are automatically cleared, or you can click Clear all to clear them yourself. When you've got unread notifications, the Notifications icon is badged according to the type of notification received. Missed info, confirmation,
		and warning messages are badged with a blue dot ; missed error
		messages are badged with a red dot a unread, you'll see the red dot over Notifications—even if the last notification was of a different type—indicating that an error condition requires your action. You can act on notifications from within the Notifications panel just as you would when the notification pops up in the bottom corner of your window. You can also track the progress of tasks that are running in the background right from the notification; for example, when a job has been triggered as part of the publishing process, you'll see a progress bar indicating that a task is in progress.
I	Preview	Before committing your changes to the branch, you can use the Preview action to see how your pages look and behave in a browser.



Label	Element	Description
J	Publish	Commit changes in the current branch to your local repo, push them to the project's Git repo, and kick off package and deploy jobs to publish your branch to your development environment. (If you want to publish the entire extension, make sure you commit and push changes in all other branches before clicking Publish.)
K	Menu	Open a menu containing the Share, Import, and Export actions, as well as commands for opening the Settings editor and navigating to the Visual Builder Studio Help Center.

Note:

At times, the header may also contain a toggle switch with the user modes "Express| Advanced". *Express mode* is a slimmed-down VB Studio experience, targeted to business users whose needs are well-defined, whereas *Advanced mode* provides access to the full Designer. This toggle appears only when the page you are working with comes from certain Oracle Cloud Applications, most notably those that use *business rules* to determine the logic governing runtime behavior. When working with such pages, one automatically lands in Express mode after clicking **Edit Page in Visual Builder Studio**, but Advanced mode is always available for more complex use cases. This book (the one you're reading) covers Advanced mode, while What Can You Do with Oracle Visual Builder Studio? explains how to use Express mode.

The Navigator

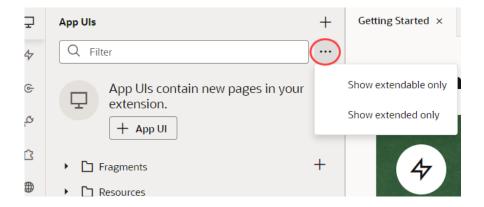
The Navigator helps you move between the artifacts in your extension, and provides access to the VB Studio editors:

Icon	Element	Description
₽	App UIs	 There's a lot going on in this section of the Navigator. You can: Create an App UI, if you want to add new pages to your extension; Create a fragment, if you need a reusable piece of UI you can use in App UIs or configurations of App UIs (or within a layout template); Create a root page to brand the pages in your App UI with a common header, footer, background, or other elements, to provide a consistent look and feel. Import resources into the Global Resource folders to use across your extension, or; Add another extension as a dependency, so you can refer to its App UI's service connections, Layouts, global resources, and so on from your own App UI, or so that you configure an App UI to meet your own business needs.
4	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. You can also view the Layouts provided by any App UI you may have as a dependency.



lcon	Element	Description
©	Services	To access external REST APIs in your extension, you can create connections to the services that provide access to these API endpoints. VB Studio also includes a catalog of predefined services in the form of an Oracle Cloud Applications <i>backend</i> . This backend exposes REST APIs—from Human Capital Management, Sales, and more—that your App UIs can consume right out of the box. You can also create custom service connections to access services that aren't listed in this catalog. See Add Service Connections to Your Extension for details.
po	Dependencies	When you add an extension as a dependency, you gain access to the resources that comes with the extension, like its service connections, Layouts, and more. Add a dependency when you want to configure one of the App UIs contained within the extension, or when you want to use the extension's resources to build your own App UI. The Dependencies panel shows you all the extensions containing an App UI that has at least one artifact flagged as extendable—a dynamic component, a variable, and so on. If an extension doesn't have an extendable App UI, it won't appear in this list.
	Components	The Components tab helps you to install and manage the components that you download from the Component Exchange, a repository of components that can be installed in your VB Studio instance.
(4)	Translations	To ease with translation, all text strings in an App UI—such as headings, labels, and messages—can be stored in a separate external file, rather than hard-coded in the App UI. This means that you can translate the App UI by simply downloading this file, translating it, and uploading a newly translated file.
	Source	Although the Designer is primarily a visual editor, you can always work in source code if you prefer.
•	Git	Shows you the list of files you've changed, but haven't yet committed to your branch. If you have merge conflicts in your branch, you'll see them listed here. Click a conflicted file to open it in the conflict resolver tool so you can resolve any issues.

By default, the App UI section displays only artifacts that you can extend, but you can use the options menu next to the Filter field to select items that have already been extended, or items available for extension:



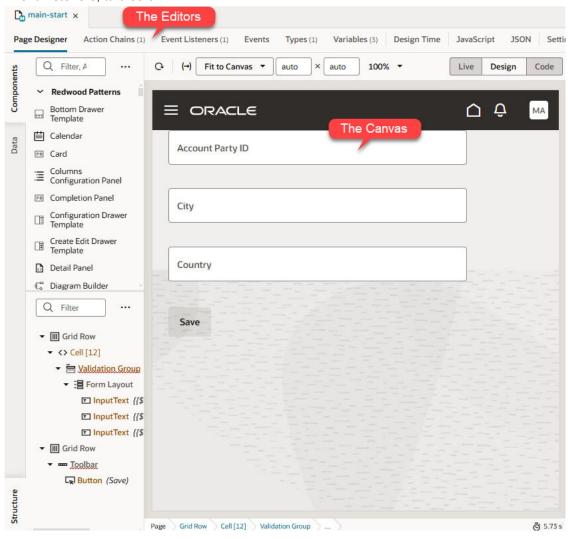


Tip:

If you need more real estate in the Designer, just click any of these icons to temporarily hide the Navigator entirely.

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 VB Studio. 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, VB Studio 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 enter 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 your App UI.



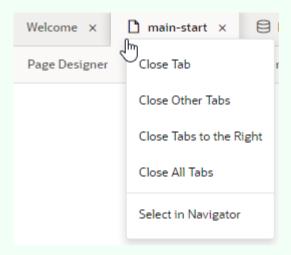
The JSON and JavaScript panes represent what will ultimately be checked into Git when you publish your extension. Therefore, if you look at these panes for an App UI you have added as a dependency, you won't see anything there—unless you have customized that App UI in some way.

0

Tip:

When working with multiple artifacts (pages, flows, layouts, 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.

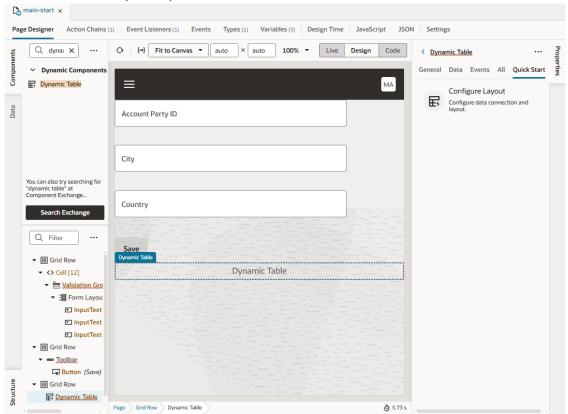


The Properties Pane

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 and streamline your extension:

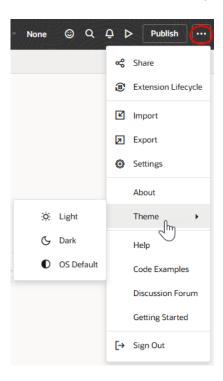
Element	Description	
Audits	Scan the code in your extension for places containing errors, warnings, info and to- dos. Your code is scanned when you open the Audits pane.	
Find in Files	Search your extension for a text string.	
Git History	View a list of Git actions you have performed in your workspace. The window displays details about each action, including the type, date and files involved.	
Logs	View build logs when your extension is shared or deployed.	
Tests	View a list of all action chain tests.	



Color Theme

The Designer, by default, uses a light theme based on Redwood to set the color palette for your work environment. To personalize your environment to use a dark theme or sync with your OS settings:

- Click Menu in the upper right corner.
- 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.



Tip:

You can also control this setting from your preferences. Click **Go to project page**, click your avatar, select **Preferences** and **General**, then look for the **Preferred color theme** setting. See Set Your Preferred Color Theme.

No matter where you configure this setting, your preferred color theme takes effect for all pages in the user interface, including the Designer.

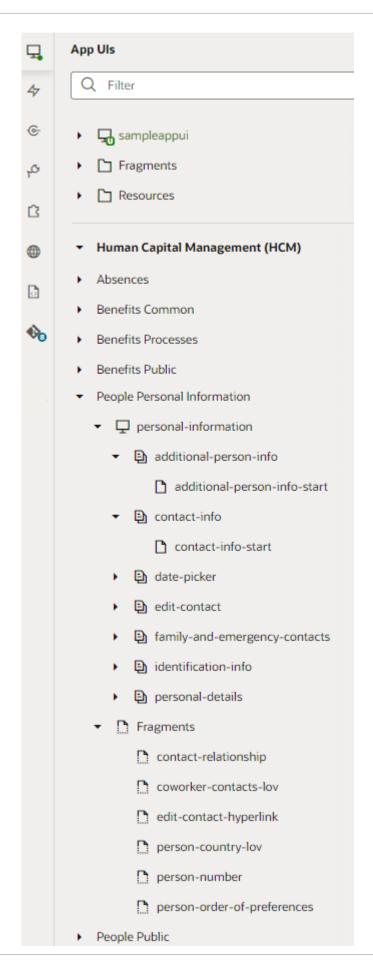


What Are Dependencies?

When an extension is listed as a *dependency*, it means you can reference any of that extension's resources, like a service connection or Layout, while building your own App UI. It also means that you can configure (customize) the extension's App UIs to meet your own business needs.

Dependencies are listed by pillar in the Designer's Navigator, in a separate section on the App UIs pane. At a minimum, you'll see the Unified Application (which serves as the underpinning to the Oracle Cloud Applications ecosystem) listed here. Depending on how you came to Visual Builder Studio, you may see other extensions listed here as well. For example, here's what you may see when your dependent extension belongs to the Human Capital Management pillar:







The People Personal Information extension shown here—which contains the personal-information App UI—is listed as a dependency under the Human Capital Management pillar. This means that you can now open personal-information and tweak any of its pages with extendable artifacts—dynamic tables or forms, dynamic containers, variables, and so on. In addition, you can use the fragments in the extension's Fragments folder in your own App UIs. If a fragment in this folder has at least one artifact marked as extendable, you can customize the fragment for your own use. Your extension has access to everything contained within its dependent extensions.

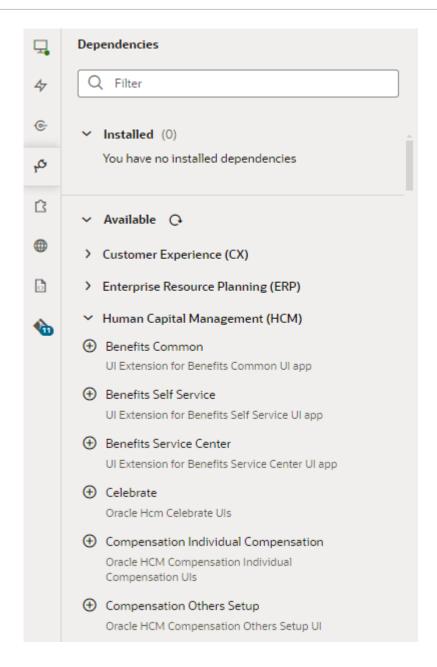
You may also see several other extensions from the same pillar listed here, for example, Absences, Benefits Common, and so on. That's because *all* extensions in the same pillar as your dependent extension become available for you to explore. So even though your extension might only depend on resources in People Personal Information, you can explore all extensions in the HCM pillar to locate resources you might want to add to your extension without actually adding them to your extension.

So when might you see an extension/App UI listed as a dependency? If you jumped over from Oracle Cloud Applications by clicking Edit Page in Visual Builder Studio, VB Studio automatically adds the extension/App UI containing the page you were just viewing to the list of dependencies, while also listing all other extensions in the same pillar. (The exception here is when you're working with extensions belonging to the CX or Other pillar. In this case, only dependencies that you've explicitly added are shown. In other words, except for the Unified Application, you won't see any dependent extensions listed when you jump over to Visual Builder Studio while editing a page belonging to CX or Other—until you actually add a dependency.)

If you created a workspace by selecting **New Application Extension** on the Workspaces page, you're essentially creating an empty extension, which is what you use to create a new App UI or to extend the Unified Application. In this case, you'll only see **Unified Application**. Once you add a dependency, all extensions belonging to that pillar will be listed under the pillar's name.

To add a dependency for your extension, use the **Dependencies** pane, where all available dependencies are listed by pillar for easier identification:





Keep in mind that you add the *extension* containing the App UI as a dependency, not the App UI itself.

Simply put, you add an extension as a *dependency* when you want to a) configure an App UI (that is, make changes to it) in the extension, or b) access the resources in the extension, like service connections or Layouts, while building your own App UI. (You'll find these resources listed by pillar on the Services and Layouts pane.)

The Unified Application provides global services and a common user interface shell to all of the App UIs that plug into it, whether created by Oracle or developed at your enterprise. For this reason, it's considered a dependency for everything in the Oracle Cloud Applications ecosystem. You don't have to worry about this underpinning, but you should know that it, too, is configurable; all you have to do is open the Unified Application and modify it, just as you would any other App UI. Note that the ask-oracle-bo-metadata and ask-oracle-chat fragments, while globally available to all App UIs in your extension, are internal to the Unified Application and cannot be customized or referenced in your extension.

If you're building your own App UI and want to make it available to others to configure, you must be sure that the App UI includes at least one extendable artifact —a variable, a dynamic component, or something else marked **Available to extensions**. If it doesn't, the extension won't appear in the Dependencies pane.

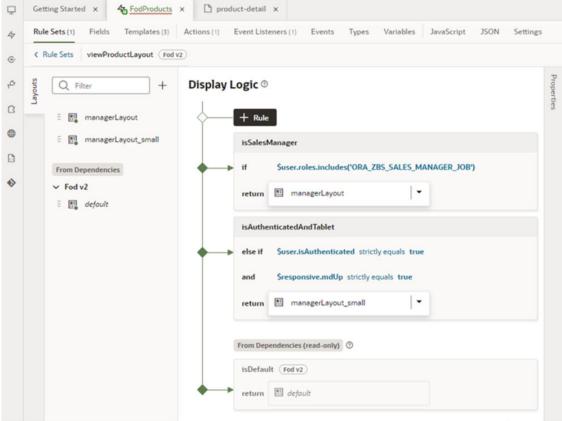
What Are Dynamic Components?

A *dynamic component* is an extendable UI component, such as a form, table, or container, that uses display logic to determine what the component shows at runtime; for example, what fields are displayed in a table and how they are rendered. When you configure an Oracle Cloud Application or another App UI, in most cases you'll be working with one or more dynamic components to achieve the effect you want.

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 location) to determine what is displayed in the component. For example, for a dynamic table showing employee time card data, you might want to show the Salary Rate column only to users who are managers, and to hide that column from all other viewers.

There are two ways to specify display logic: *rule sets* and *business rules*. The main difference between them is that whereas a rule set operates on a single dynamic table or form, business rules allow you to impact a field no matter which dynamic component (or which page) it appears on.

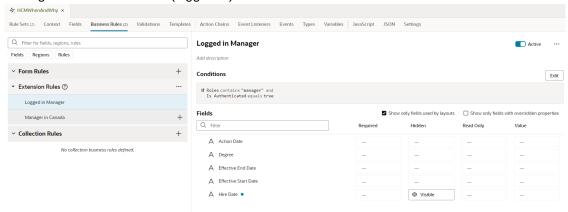
Here's an example of what a rule set for a dynamic form looks like:



In this rule set, the display logic has three rules: the default rule at the bottom, and two rules that have been added above it. The top rule, <code>isSalesManager</code>, determines if the user viewing the page has the user role (ORAZBSSALESMANAGERJOB). If they do, then the user will see the fields defined in the <code>managerLayout</code> layout displayed in the form. If they don't, the next rule (<code>isAuthenticatedAndTablet</code>) is evaluated. This rule set also has two layouts, <code>manageLayout</code> and <code>manageLayout_small</code>, in addition to its default layout. Each layout contains a different configuration of fields, columns, and so on, which dictate how the component should look when rendered at runtime. See Control Your Display with Rule Sets for more information on how to use rule sets and layouts.

Whereas rule sets dictate the conditions under which a given layout is applied to a dynamic component, business rules dictate the conditions under which fields *within* the dynamic component behave. You can make a field read-only/editable, hidden/shown, or required/optional. These conditions impact fields no matter what dynamic component they are rendered in, because they are applied at the business object level (that is, the underlying data structure). With business rules, instead of configuring dynamic tables and forms individually, you could create a single rule to conditionally hide every occurrence of a field, in every form where it appears and on every page where it appears. If you were to do this using rule sets, you would need to configure the rule set for *each* form to hide the field. Business rules can handle simple and complex display needs that might otherwise require tens, if not hundreds, of individual layouts if the rule set approach was used instead.

Here's an example of using business rules to hide the Hire Date field when the user is a manager and authenticated (logged in):



Business rules can also set a default value for a field, or display a message when a given condition is met. See Control Your Display with Business Rules for more.

Business rules and rule sets can be applied to the same Oracle Cloud Applications page, depending on how the developers at Oracle chose to build the page, and whether they chose to make the component extendable. At runtime, if you've made changes through both methods that happen to impact the same field, the changes made through rule sets take precedence.

Note:

If you've created your own fields for a layout, as described in the section Create Fields For a Layout, bear in mind that these types of modifications take lower precedence at runtime. In other words, such fields can be overwritten with changes stemming from business rules and/or rule sets (in that order).



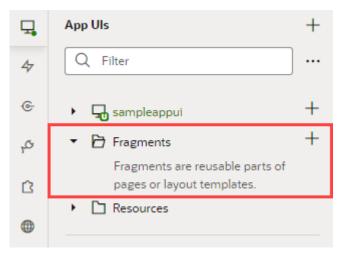
There are three types of dynamic components that can be used in app extensions: tables, forms, and containers. What is displayed in a component and how you customize it depends on what type of component it is:

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. Watch this video to better understand how dynamic UIs work: Video: Work With Dynamic UIs	
Dynamic container	Dynamic containers are pre-defined 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. Suppose you want to create a page that lets users toggle between 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.	

See Add Dynamic Components to Pages for guidance on adding dynamic components to an App UI, or Customize Dynamic Tables and Forms if you're configuring dynamic components in an App UI that someone else has created.

What Are Fragments?

When you look at the App UIs tab in the Navigator, you may notice a section called Fragments, as shown here:



A *fragment* is a reuseable piece of UI that you can include in an App UI, or even a configuration of another App UI. For example, suppose you had a large app with many complex pages—pages with a foldout layout, for example, or with multiple panels or tabs—that had become unwieldy to work on and costly to render. By isolating the content of a given tab or panel within a fragment, you effectively can modularize your App UI's logic, which allows you to maintain each panel or tab separately. In addition, fragments created for one page can be reused in other parts of pages, and can even serve as entire page templates.

Besides the benefits of reuse, fragments provide performance gains. Typically, all components used in a page load when the page renders. But sometimes you don't need all components, 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.

As you work with fragments, keep in mind that the same fragment can be used in more than one App UI *within the same extension*. If you want to use a fragment in another extension, just add the extension containing the fragment as a dependency to the new extension.

Work With Fragments tells you more about fragments and how to use them. Also see Work With Fragments From Dependencies if you are customizing an App UI to suit your business needs.

Note:

The Unified Application, which underlies all App UIs in your extension, also contains fragments, specifically <code>ask-oracle-bo-metadata</code> and <code>ask-oracle-chat</code>. These fragments, while globally available to all App UIs in your extension, are internal to the Unified Application and cannot be customized or referenced in your extension.

How Do I Use My Sandbox in Visual Builder Studio?

VB Studio can access sandboxes in Oracle Cloud Applications that have changes to the application's data model that haven't been published yet.

If the changes in a sandbox are relevant to your extension, you can associate the sandbox with your workspace so you can access it while you're working. Here are some tips:

- You'll want to do all your work in a branch that is associated with the sandbox with those changes, and continue to use the same branch with that sandbox until you are finished.
- If you need to edit the extension using a different sandbox as well—for example, a sandbox defining an alternative data model that is being considered—it's better to associate it with a new branch than to use an existing branch. This will help isolate the changes you make for each sandbox.
- For each branch-sandbox pair, you can create a separate workspace instead of switching a workspace's branch and sandbox. VB Studio helps keep your changes in sync by automatically recommending a sandbox for your workspace if you choose a branch that is already using it.

If you're using a sandbox with your workspace, VB Studio will notify you if there are changes to the sandbox (for example, it was published or deleted) or to the Oracle Cloud Application environment that could affect your extension. The notification will contain a description of the change and a link you can use to help address it. For example, here's a notification you might see when the sandbox associated with a workspace needs to be refreshed because of changes in the application's environment:



Sandbox refresh needed

Another sandbox has been published in the Fusion Applications environment associated with this workspace. To avoid potential conflicts, you might want to refresh to pick up those changes.

3 minutes ago

Refresh

Clicking the Refresh link will start the process to bring your sandbox up-to-date. For more about working with sandboxes in Oracle Cloud Applications, see Overview of Sandboxes in Configuring and Extending Applications.

When you're ready to publish your extension to your Oracle Cloud Applications environment, you should publish the associated sandbox at the same time or shortly before to minimize the risk that the data model and extension will be out of sync. In addition, once you've published your extension, your users must sign out, then sign back in to the Oracle Cloud Applications environment to be certain they're seeing the latest changes.

Visual Builder and Visual Builder Studio: What's the Difference?

Although there is some overlap between Oracle Visual Builder and Oracle Visual Builder Studio, they are in fact two distinct Cloud offerings with different capabilities.

Here's a brief overview of the similarities and differences:

	Visual Builder	Visual Builder Studio (VB Studio)
Do I need a license?	Yes, Visual Builder requires a license, based on the number of OCPUs that you provision.	If you have already purchased an Oracle Cloud subscription, VB Studio is a free entitlement for you. Oracle developers use VB Studio to build Oracle Cloud Applications, which is the same tool you use to customize and extend those apps. For that reason, VB Studio is provisioned as part of every Oracle Fusion Cloud Apps tenancy.
What comes with it?	Along with an integrated editor called the Designer, Visual Builder comes with a web/app server, a database, a proxy for authenticating and calling REST services, and integration with Oracle's identity service for authentication.	VB Studio is a platform for teams to manage their development process and source code, as well as to automate CI/CD. VB Studio includes Git, CI/CD pipelines, issue tracking, development dashboards, wiki, and more. It also comes with the integrated editor, the Designer, to build both responsive apps and App UIs.
What was the former name?	Visual Builder used to be known as "Visual Builder Cloud Service (VBCS)", and is sometimes referred to as "VB standalone".	VB Studio's DevOps capabilities used to be packaged separately and were known by the name "Developer Cloud Service (DevCS)".



	Visual Builder	Visual Builder Studio (VB Studio)
What can I build?	Using the Designer, you can build visual applications (also called <i>responsive</i> applications), then deploy or "host" them on the Visual Builder instance itself.	 With VB Studio, you can build: Visual applications (also called responsive applications), which are then deployed to a Visual Builder instance, purchased separately. Unlike Visual Builder, VB Studio is not a hosting platform. Extensions (containing App UIs or other resources) to customize or add new capabilities to Oracle Cloud Applications, which are then deployed to an Oracle Cloud Application instance. Bespoke applications using the programming language of your choice, which are then deployed to Oracle Cloud.

You may also be wondering what the difference is between visual applications and extensions:

- Visual applications function as standalone apps that are accessible from anywhere, with their own security implementation. They can use different look and feel schemas, have their own custom domain names, and use their own database for storing data in custom objects.
- Extensions are created as additional pages in your Oracle Cloud Application instance.
 They are an integral part of the UI experience (not iFramed), appear in the Oracle Cloud Application menus, and leverage the same users and roles for authentication and authorization. Extensions are only accessible by Oracle Cloud Applications users and have the same look and feel as the rest of Oracle Cloud Applications.

While both types of apps can integrate data from external REST services, they differ in the authentication types and connection topologies that are supported out of the box. Notably, visual apps can support proxy-based flows and avoid CORS issues for external REST services, and also support more authentication. For more information, see the authentication topic in visual app documentation and in App UI documentation.

What's the Extension Lifecycle?

Whether you're configuring an Oracle Cloud Application or creating an App UI from scratch, the evolution of your extension follows a very similar path. Here's the typical process for creating and publishing an extension:

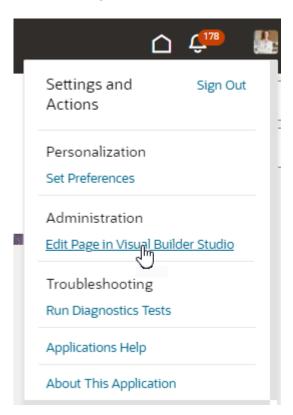


If you're creating a new App UI, steps 1-3 don't apply. Instead, you'd create a new workspace, create a new App UI, then start dropping components on the page. The remaining steps in this topic are common to both App UI creation and configuration.

In Oracle Cloud Applications, open the application you want to configure.
 Make any necessary data model changes in your sandbox before proceeding.

The changes you make are upgrade-safe, meaning that they will persist from one version of your Oracle Cloud Application to the next.

Click your user name at the top-right corner to open the Settings and Actions menu, then select Edit Page in Visual Builder Studio:



3. Select a *project* and a *workspace*.

To edit a page in VB Studio, you need a project that has been set up to configure Oracle Cloud Applications in this environment. If you're a member of several projects, select one. If a project already has a workspace for the page you're editing, you'll want to pick one of

those. These projects are badged as Recommended but you're not a member—you'll see these under **Other Projects**—select the project and click **Request Membership** to contact the project owner and get yourself added. You'll be notified by email when your request is approved, at which point you can try editing the page again.

If you're not a member of any projects, and you don't have any recommended projects, you'll need to create one. Click **+ Create**, then enter a **Project Name** in the New App Extension Project dialog. If you know others may work with you in this project, optionally select their names in the **Add Members** list. (Make sure someone has enabled these users to access VB Studio by following the instructions in Set Up VB Studio Users.) Click **Create**.

Projects created this way use default settings to get you started right away. Once the project is created, you can access its properties to explore all the options available to you (click **Go to project page** at the top left of the header, then look for **Properties** under **Project Administration** in the VB Studio left navigator).

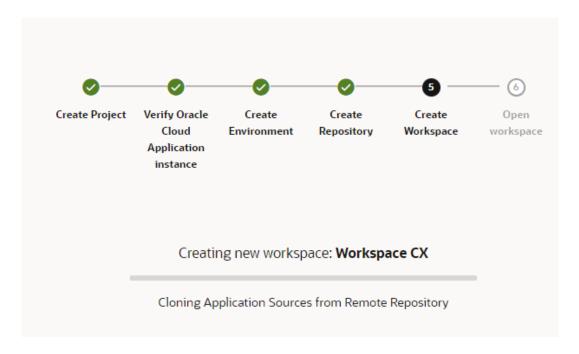
for **Properties** under **Project Administration** in the VB Studio left navigator). As someone who creates the project, you automatically become the project owner, with rights to manage it.

At this point, you can customize the left navigator to show or hide menu items, both for yourself as well as other project users. By default, you'll see Organization, Project Home, Workspaces, Git, Merge Requests, Builds, Environments, and Project Administration in the menu, but you can customize this view by clicking in the left navigator. See Configure the Left Navigator. More importantly, you may want to set merge restrictions on the main branch in

the project's Git repository to control who pushes commits to it and how. See Set

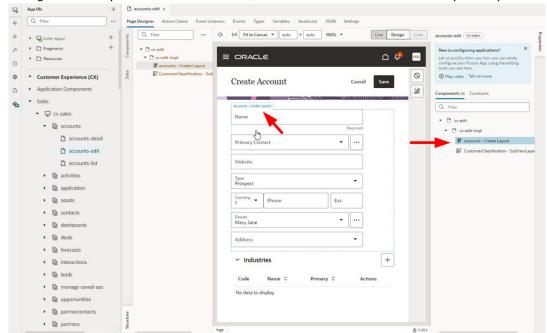
Whether you create a new project or select an existing one, you'll need a workspace, which we'll create for you if you don't have one. For example, this is what you see when a workspace is created as part of the new project workflow:

Merge Restrictions on the main Branch.



If you already have a workspace in the project for a page in a given pillar, VB Studio will either automatically open that workspace for you, or present you with options as described in Configure an Oracle Cloud Application.

4. Once you access a project and a workspace, you'll see your page open in the Designer with several editor tabs. In the canvas, you'll see the page you were just viewing, with the

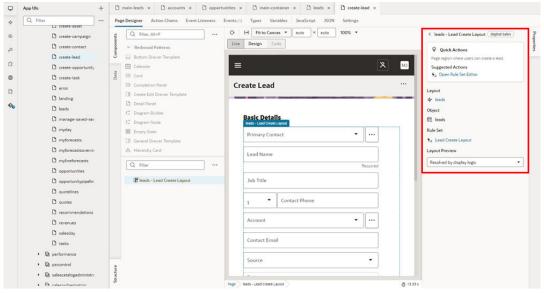


configurable components outlined in blue, both in the canvas and in the Properties pane:

In the Navigator pane on the left, you can see that your Oracle Cloud Application—in this case, Sales—has automatically been added as a dependency for this extension (as has the Unified Application, which is considered a dependency for all extensions).

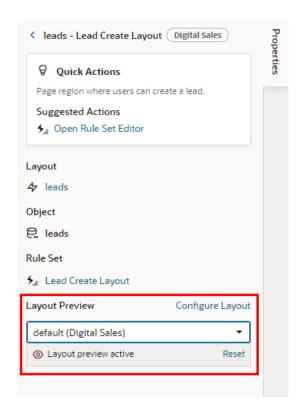
Your view of the Navigator may be different depending on the base application you're configuring, or if you're working with pages in a brand new App UI—but the lifecycle phases are the same.

5. When you click a component on the canvas, you'll see some of the component's properties in the Properties pane:



In this example, you can see details about the dynamic component defined in the dependency. To configure the component, you can use the **Open Rule Set Editor** link under Suggested Actions to open the editor.

6. View your changes with the preview tools. For some components, you can also use the **Layout Preview** in the Properties pane when editing a page to check your layouts in the Page Designer:



Or use **Preview** in the Designer's header to see your changes as they'll appear in the browser page:



- 7. Share your changes to get feedback.
 - It's good practice to ask your team members to review the changes you make to an extension before you publish it. Click **Menu** in the upper right corner and select **Share** to generate a URL for the preview, which you can then give to team members so they can test your work. If you're working with a sandbox, be sure to give the testers the name of the sandbox along with this URL. If you're not working with a sandbox, simply share the URL.
- 8. Commit your changes to a branch. Click the Git menu in the header and select the command to commit and push the changes in your workspace to the remote branch. Alternatively, click **Publish** in the header to automatically commit, push, and merge your changes.
- 9. Create a merge request asking team members to review your changes. Your team members can review the changes to the source files in the branch and approve the request if they look good. For some projects the merge requests might be optional, but a project owner can make merge requests and reviews mandatory.
- 10. Merge your changes into the project repository.

 When your changes have been reviewed and approved, you can merge the branch containing your changes into the project repo's main branch. If you're using the Publish option, your changes are also deployed to your environment's Oracle Cloud Applications instance. How this is done depends on your extension's CI/CD pipeline setting:

- When the CI/CD pipeline is disabled, your changes are deployed directly to your target instance during publish. Starting with version 25.01.1, this is the default setting when you create a workspace to configure your Oracle Cloud Application through the Edit Page in Visual Builder Studio link.
- When the CI/CD pipeline is enabled, your changes are deployed by triggering package and deploy jobs included as part of a CI/CD pipeline. The package job will create the build artifact that is deployed to your Oracle Cloud Applications Development environment, or a Test environment for further tests. After testing, a separate build job can publish the build artifact to the live (production) environment, assuming your administrator has set things up that way.

To ensure your changes will load successfully, the target instance must be running the same Oracle Cloud Application release version as your development instance. If you develop an extension on, say, 24D in your Test environment, then want to deploy the extension to your 24C Prod environment, you should wait until your Prod instance has been upgraded to 23D before you deploy the extension. In most cases, there shouldn't be more than a two-week gap between pod upgrades. See Oracle Applications Cloud – Fusion Applications Update Policy (Oracle account sign-in required).

11. View your published changes.

After your changes are deployed (either directly or via build jobs), you'll be able see your work by going to your Oracle Cloud Applications instance and pointing your browser to the App UI you just created or configured. You may need to re-authenticate to see your latest changes.

What happens if someone else is working on an extension that modifies the same page that you're working on? See How Are Extensions Applied at Runtime? for answers.

How Are Extensions Applied at Runtime?

What happens if two users deploy an extension on top of the same App UI, either from the same project (in other words, the same Git repository) or different projects? How are their changes applied at runtime?

The answer depends on where the changes are made and when. Let's consider three scenarios:

- Suppose User A and User B are extending the same App UI, each working on separate branches within the same Git repository. User A publishes his changes first. Later, when User B publishes, VB Studio fetches User A's changes, merges them with User B's, and deploys them as a single unit. If for some reason VB Studio cannot perform the merge successfully, User B is presented with a merge conflict, which he must resolve in the process of manually merging the branches before publishing. In this scenario, the extension contains a combination of the changes from User A and User B.
- Now let's suppose that User A and User B are working on my_App_UI in different projects and repositories, but at different times. That is, first User A publishes an extension to my_App_UI, but later User B creates a new project and Git repository to extend the same App UI. Because VB Studio knows that User A has already created an extension for my_App_UI, VB Studio seeds User B's new Git repo with the latest extension published by User A. User B can then make additional changes on top of User A's extension and publish, without disturbing the changes made by User A.



Finally, imagine that User A and User B are working simultaneously on the same App UI, but in different projects and repositories. User A publishes his extension and verifies them. User B then makes a change that effectively overwrites the changes made by User A. Because the date stamp on User B's changes are more recent than User A's, User B's changes take precedence.

To avoid possible conflict between extensions, VB Studio recommends that you always begin your extension work by going to the Oracle Cloud Application you want to extend, then clicking **Edit Page in Visual Builder Studio**. This gives VB Studio a chance to guide you towards joining a project/Git repository that is already dedicated to extending the App UI, rather than creating duplicate code.

Extension Best Practices

For optimal behavior, there are a few best practices you should follow when setting up and working on your extensions:

- If your team plans to extend Oracle Cloud Applications from multiple product families, like HCM, SCM, and so on (known as *pillars* in VB Studio), the way you set up your projects depends on your organization's specific needs:
 - In general, it's easiest to use a single VB Studio project for all your extension work, with a different Git repository for each pillar. Using a single project keeps everything together in one place, limits the number of permissions each user requires, and generally makes sense if your team is on the smaller side.
 - If, on the other hand, you have users that need to be isolated from each other's work—say, different consultant organizations doing implementation work for different pillars, or a larger team that uses a common Oracle Cloud Application environment family to develop, test, and host extensions from multiple pillars—you may opt to separate your work by creating one project for each pillar.

Whether you choose to use a single project with multiple repositories or multiple projects, the goal is to keep the work for each pillar separated in its own extension (repository).

Note:

Keep in mind that each project starts fresh with a clone of Oracle's code for a given Oracle Cloud Application, so any customizations you've made using other projects won't be automatically included. If you already have more than one project created for a Oracle Cloud Application, the best option is to simply consolidate all changes into a single project manually and continue from there.

- To access Visual Builder Studio, start on the Oracle Cloud Applications page you want to
 modify, then click Edit Page in Visual Builder Studio from the Settings and Action menu.
 Using this method directs you to the right project and workspace and avoids creating extra
 Git repos and extensions that aren't needed.
- After an Oracle Cloud Application release update, continue to use the same project and workspace for your extension work. VB Studio automatically updates your workspace to take advantage of any new functionality as soon as you open it. These changes are propagated to the associated Git repository (extension) when you click Publish.



Not all Oracle Cloud Application release updates will require updates to your extension source code.

- To ensure your changes load successfully when publishing your extension, the target instance must be running the same Oracle Cloud Application release version as your development instance. If you develop an extension on, say, 24D in your Test environment, then want to deploy the extension to your 24C Prod environment, you should wait until your Prod instance has been upgraded to 24D to deploy the extension. If your instances are out of synch, you'll see this message: "Dependent Extension with Extension id: xxxxand version >=xxxxxx doesn't exist". In most cases there shouldn't be more than a two-week gap between pod upgrades. See What If Dev and Prod Instances Aren't On the Same Release?
- Avoid deleting projects that contain active extensions—that is, extensions that are
 deployed to a live production instance. If you do delete such a project, the extensions will
 remain active, but you won't be able to update them nor perform any lifecycle activities on
 them moving forward.



Get Started

If you've read the concepts in The Basics (which is highly recommended), you know that regardless of whether you want to configure an Oracle Cloud Application or build your own App UI, you create and publish your work within an *extension*.

Each time you create a new extension in VB Studio, you must associate it with a Git repository, which stores all your work. This chapter helps you get started, depending on what you want to do, and explains how and when to create a new Git repository (or how to use a scratch repository, if you prefer). Once you're in the Designer, this chapter also helps you find your way around the Page Designer, one of the most actively used and important editors.

Hint: If you don't know much about Git repositories and how they operate within the context of VB Studio, you may want to watch this video: Video

What Do You Want To Do in VB Studio?

The path you take to get started in Visual Builder Studio varies, depending on what you want to do. Click the link that best describes your goal:

- I want to customize (configure) a page in an Oracle Cloud Application.
- I want to add a new page to an Oracle Cloud Application.
- I want to add a new application to the Oracle Cloud Application ecosystem.
- I want to add a resource to the Oracle Cloud Application ecosystem (like a Layout or service connection) so that others can use it.
- I want to create a bespoke application and host it on a Visual Builder instance.

Options two, three, and four involve creating a new App UI in Visual Builder Studio.

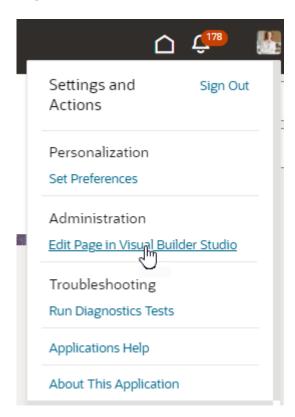
Configure an Oracle Cloud Application

You can configure an Oracle Cloud Application to hide or rearrange fields in a dynamic form or table, introduce new behavior based on a variable's value, add your own content to a page, and much, much more. The easiest way to start is by viewing your page in Oracle Cloud Applications:

 Enter the URL for your Visual Builder Studio instance. (If you don't have it, you can navigate to VB Studio from your Oracle Cloud Application's Navigator. Select Configuration, then Visual Builder.)

Before continuing, make sure you or someone else in your organization has completed the steps in Essential Set-Up Tasks . In addition, if you have multiple VB Studio instances, make sure you're using the one that's associated with the correct Oracle Cloud Applications instance. That is, if you're on the DEV1 Oracle Cloud Applications instance, you must use the VB Studio associated with DEV1, and not, say, the VB Studio instance associated with TEST, or a VB Studio instance in a completely different Oracle Cloud Applications environment family. Of course, you can deploy extensions to any instance in any environment family, as long as you have a VB Studio environment defined to support that.

2. In your Oracle Cloud Application, navigate to the page you want to modify, click your user name at the top-right corner to open the Settings and Actions menu, then select Edit Page in Visual Builder Studio:





If you don't see the **Edit Page in Visual Builder Studio** option in your Oracle Cloud Application, it could be because:

- You don't have the right privileges to access VB Studio. Check with your Oracle Cloud Application administrator if you're not sure.
- You may not be working in an environment that has an instance of VB Studio associated with it, such as a TEST environment. Again, check with your Oracle Cloud Application administrator to see if this is the case.
- Your Oracle Cloud Application has not yet adopted Oracle's new Redwood design pattern, so this page is not extensible using VB Studio. In that case, refer to Configuring and Extending Applications for instructions on how to customize your Oracle Cloud Applications with App Composer to meet your business needs.

3. Select a *project* and a *workspace*.

To edit a page in VB Studio, you need a project that has been set up to configure Oracle Cloud Applications in this environment. If you're a member of several projects, select one. If a project already has a workspace for the page you're editing, you'll want to pick one of

those. These projects are badged as **Recommended**. If a project is recommended but you're not a member—you'll see these under **Other Projects**—select the project and click **Request Membership** to contact the project owner and get yourself added. You'll be notified by email when your request is approved, at which point you can try editing the page again.

If you're not a member of any projects, and you don't have any recommended projects, you'll need to create one. Click **+ Create**, then enter a **Project Name** in the New App Extension Project dialog. If you know others may work with you in this project, optionally select their names in the **Add Members** list. (Make sure someone has enabled these users to access VB Studio by following the instructions in Set Up VB Studio Users.) Click **Create**.

Note:

Projects created this way use default settings to get you started right away. Once the project is created, you can access its properties to explore all the options available to you (click **Go to project page** at the top left of the header, then look

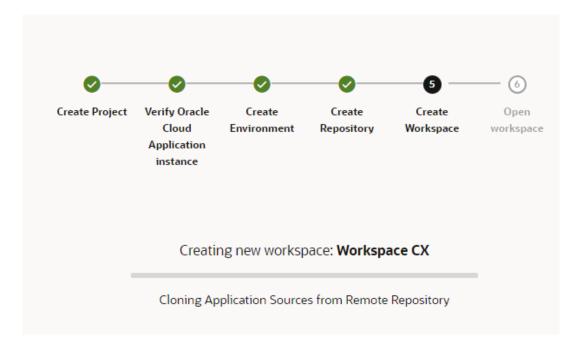
for **Properties** under **Project Administration** in the VB Studio left navigator). As someone who creates the project, you automatically become the project owner, with rights to manage it.

At this point, you can customize the left navigator to show or hide menu items, both for yourself as well as other project users. By default, you'll see Organization, Project Home, Workspaces, Git, Merge Requests, Builds, Environments, and Project Administration in the menu, but you can customize

this view by clicking in the left navigator. See Configure the Left Navigator. You may also want to set merge restrictions on the main branch in the project's Git repository to control who pushes commits to it and how. See Set Merge Restrictions on the main Branch.



Whether you create a new project or select an existing one, you'll need a workspace, which we'll create for you if you don't have one. For example, this is what you see when a workspace is created as part of the new project workflow:



If you're the first person to edit an Oracle Cloud Applications page in a given pillar, things are fairly straightforward: A new workspace is created for you as part of a new repository, which will be used to host changes for *all extensions within this pillar*. This means you'll have distinct repos and workspaces for extensions in different pillars, for example, one repo for all HCM changes and another repo for SCM changes. The exception here is if the page you're editing belongs to the CX pillar. In this case, when the Oracle CX extension's settings have the **Pillar** and **Pillar Subcategory** fields configured, a new Git repository and workspace are created to host changes for *each module within this pillar*, for example, one repo for CX Sales, another for CX Service, and yet another for CX Marketing.

If you're not the first person to edit a page in a given pillar, here's how workspaces behave based on your setup:

- If you already have a workspace in the project for the page you're editing, VB Studio will automatically open that workspace for you.
- If you have more than one workspace for the same page in that project, you'll be prompted to select one.
- If you have a workspace for a different page, but it belongs to the same pillar, you must re-use that workspace, even though it wasn't originally created for the page you're currently editing. This avoids creating a surplus of workspaces—and Git repositories—every time you edit a page for the first time in a project.
- If someone else is already working on a page in the same pillar—whether it's the page you're trying to edit or not—a new workspace is created for you by cloning your teammate's existing repository.

If you're looking to edit a page that doesn't belong to any pillar, you can create a new repo with a workspace, or re-use an existing workspace that was created for some other page.

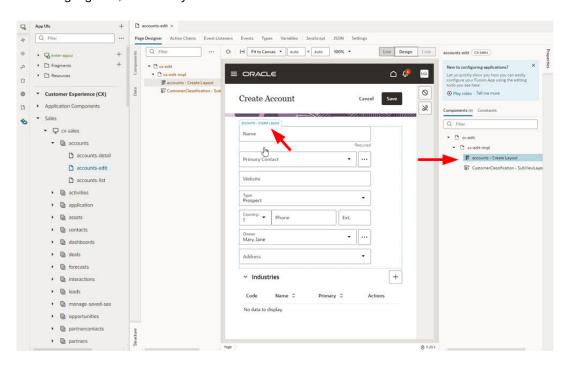


It's worth noting that if your environment's Oracle Cloud Applications instance is connected to VB Studio via Basic Auth or three-legged OAuth when both instances are in the same identity domain, VB Studio won't create a workspace or open an existing workspace associated with that environment. Instead, here's what happens:

- If a workspace doesn't exist but the environment does, a new environment with the Oracle Cloud Applications instance added as an IDCS resource is created and a new workspace is associated with it.
- If a workspace tied to the environment exists, a new environment with the Oracle Cloud Applications instance added as an IDCS resource is created and the existing workspace is switched to use the new environment.

The default workspace name generated when you jump over to VB Studio typically uses the pillar of the page you're trying to extend, in the format <code>Workspace PILLAR</code>, for example, <code>Workspace CX</code>. If that name already exists, a number is added to the name and incremented as needed, for example, <code>Workspace CX 1</code>, <code>Workspace CX 2</code>, and so on.

Once you access a project, you'll see your page open in the Designer, with the extensible areas highlighted, like this dynamic form:



- 4. Select an extensible area by clicking it, then selecting a component or rule set in the Properties pane on the right. You'll then be placed in one of VB Studio's editors, where the real work begins.
- 5. Use Configure an App UI to help you configure your Oracle Cloud Application to meet your organization's requirements.
- 6. When you're done making your changes, you can preview and share your extension to test and prepare your configurations for publication, then publish them.



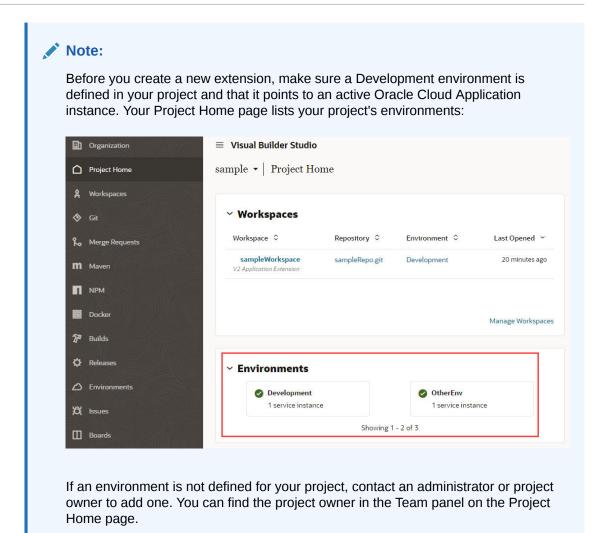
If you're ready to work on another page, make sure you push your changes to the project's Git repository before you make the switch, so the code underlying your changes becomes available to your team members. If your changes haven't been pushed to the project repo, your teammates are likely to start with a version that does not include your updates when they click **Edit Page in Visual Builder Studio**.

To publish your changes, you can individually commit and push your changes, or click **Publish** to combine multiple Git operations. Note that configuration changes you make by editing an Oracle Cloud Application page (through the **Edit Page in Visual Builder Studio** link) are deployed directly to your target instance during publish, unless the extension's CI/CD pipeline is enabled.

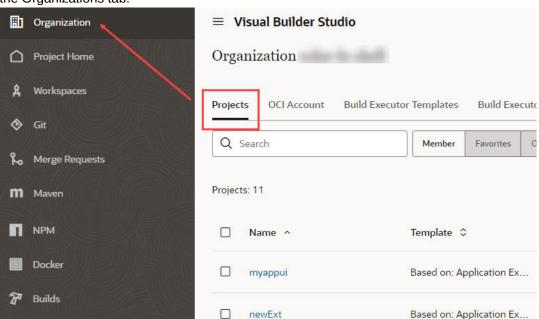
Create an Extension

If you accessed VB Studio by clicking a link in an Oracle Cloud Application, your extension (also considered your workspace) is usually created along the way. However, if you're starting from VB Studio, you'll need to explicitly create an extension. Typically, you'd do this to create your own application—or App UI—that lives alongside other Oracle applications.



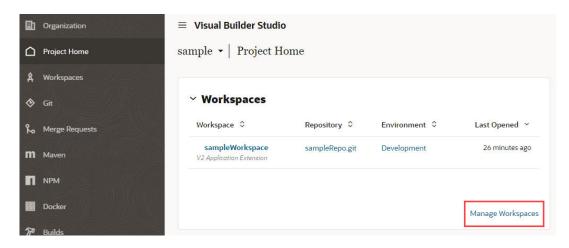


To create a new extension:



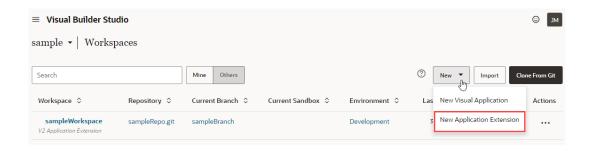
 If you haven't already, log into VB Studio and select the project you want to work in from the Organizations tab:

2. On the Project Home page, select the workspace you want to work in, or click **Manage Workspaces** to create a new workspace:



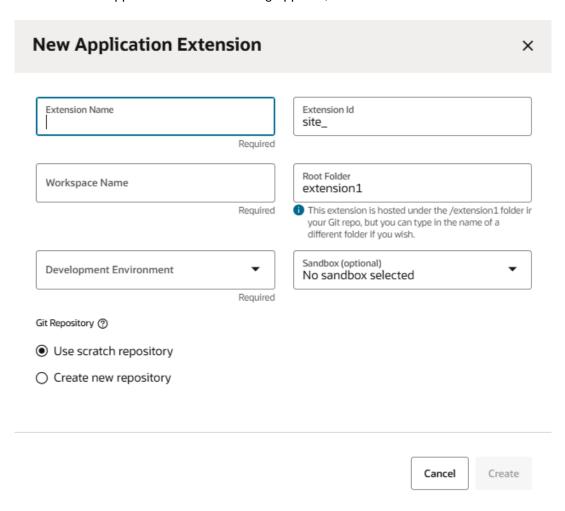
Remember that a workspace is tied to a Git repository, and that it's considered best practice to use a single Git repo for *all* changes made to a single App UI. So, if you know that someone else has already made changes to an App UI you're planning to work on, it's best to either clone or import the existing repository at this point, rather than create a new one. See Collaborate on an Extension.

3. On the Workspaces tab, select **New**, then **New Application Extension**:



When you choose this option, you're essentially creating an empty extension, which is what you use to create a new App UI. In this scenario, your extension uses the Unified Application, which underlies all App UIs in the Oracle Cloud Application ecosystem, as its base application.

4. When the New Application Extension dialog appears, enter a name for the extension:



Because your extension may become available to others, keep these naming standards in mind:

- Keep names meaningful and brief.
- Do not use "extension" (or its variants).
- Do not use (or combine) generic phrases like "UI", "app", or "application".



Tip:

If you need to rename the extension later for some reason, you can do so on the Settings panel.

- 5. If you want, change the extension ID and workspace name. Both fields are automatically filled in based on the extension name, but you can change it if you like.
 - The extension ID (used internally by Oracle) by default adds the extension name to the prefix site_.
 - The workspace takes the extension name by default.
- 6. To change the default location of your extension's files in the Git repo, change the Root Folder from extension1 to some other folder name. Your extension will be hosted under this root folder.
- 7. If multiple environments are available, select the Development Environment for your extension. Only environments that are in the same identity domain as the logged-in user are listed. (If only one environment is available, it is automatically selected for you.)



Note:

Environments with Oracle Cloud Application instances connected via Basic Auth or OAuth are not supported for extension development and will not show in the Development Environments list. To use that instance for development, you'll need to create a new environment, then add the instance as an IDCS resource using the Identity Domain option.

- 8. Select a sandbox if your new extension will involve changes that have been made to the base app's data model in an unpublished Application Composer sandbox. (If a change is published, it's already part of the data model and therefore a sandbox isn't needed. But if the change is still in a sandbox, you need to make it accessible to your extension by naming the sandbox explicitly. You may also need to re-authenticate to the Oracle Cloud Application periodically to make sure you have access to the absolute latest data model.)
- 9. Click Create new repository, then enter the name of the Git repository and branch that will be created for storing your files. If you don't want to create a Git repository now—for example, you only want to experiment with creating an extension—keep the Use Scratch repository option. Selecting this option will create a repository in your workspace that only you can see, but you won't be able to perform Git operations within the workspace. Also, build jobs and pipelines that allow you to build and publish your extension won't be created. See Use a Scratch Repository for more details.
- 10. Click Create.



Note:

After you create your extension, it's a good idea to go into Settings and provide a description for your extension right away. This will help people who are trying to add your extension as a dependency later understand what's in it.

From here you can create a new App UI, or configure an existing App UI.

Collaborate on an Extension

To work on an App UI, you must have a workspace; it's your ticket into the Designer. When you create a workspace, you must associate it with a Git repo—either one you create from scratch or, if you want to collaborate with someone who has already started configuring this particular App UI, a copy of the Git repo they've been working in.

In fact, if you know that someone else has already made changes to an App UI you're planning to configure, it's considered best practice to use the same Git repository in your workspace, rather than creating a new one.

If you need to work on an App UI that's already been worked on, you have two choices:

- Clone the repository in your own workspace
- Import a repo that has been exported as an archive into your own workspace

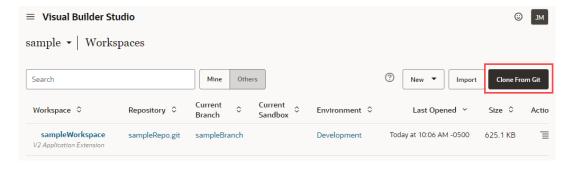
Clone an Existing Repository

When cloning an existing repository, you clone the branch containing the changes you want in your workspace. Usually, you'll want to clone the main branch to ensure that your new branch contains the most up-to-date changes.

Two workspaces shouldn't use the same branch. If you want to use a branch used in another workspace, the owner of the Git repo will first need to push that branch. It will then be available for you to check out when creating a workspace that clones that repository.

To create an extension by cloning someone else's repository:

- 1. Select the project you want to work on from the Organization tab.
- 2. Click **Workspaces** in the left navigator (or select **Manage Workspaces** on the Project Home page), then click **Clone from Git**:





Tip:

You can also create a workspace from the Git repositories page: Click \mathbf{Git} $\mathbf{\Phi}$, then \mathbf{Create} $\mathbf{Workspace}$.

- Enter a name for your workspace.
- 4. Select the Git repository to clone. If you're cloning from the Project Home page, the repo is already selected for you.

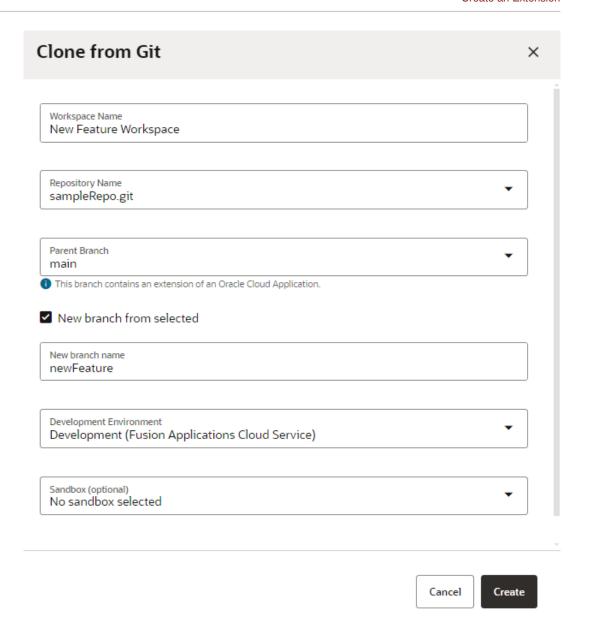


- 5. Select the repository branch that has the changes you want in your workspace. This is typically main for the latest changes, but it can be any branch.
- (Optional) Select New branch from selected and enter a name for the branch you want to create. You can create additional branches and switch between branches in the workspace.
- 7. If multiple environments are available, select the Development Environment for your extension. Only environments that are in the same identity domain as the logged-in user are listed. (If only one environment is available, it is automatically selected for you.)

Environments with Oracle Cloud Application instances connected via Basic Auth or OAuth are not supported for extension development and will not show in the Development Environments list. To use that instance for development, you'll need to create a new environment, then add the instance as an IDCS resource using the Identity Domain option.

8. (Optional) Select the sandbox you want to use with this workspace, if any.





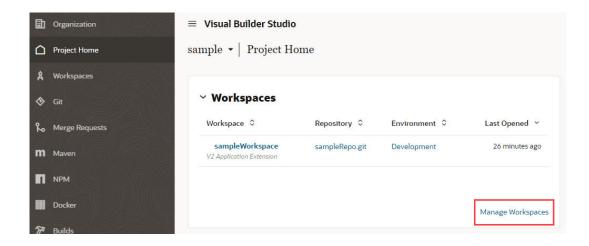
9. Click Create.

Import an Extension Archive

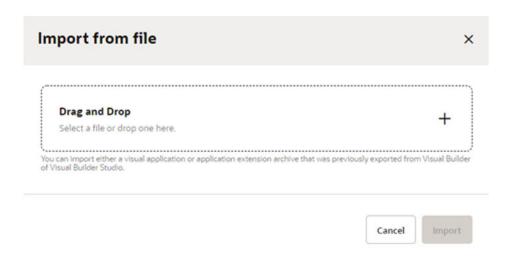
If a team member gives you an archive of an extension, you can import it to create a workspace containing all the files in their branch of the extension's Git repository. When you create a workspace by importing a file, you create a new Git repository and branch.

To create a workspace by importing an archive:

- 1. Select the project you want to work on from the Organizations tab.
- Click Manage Workspaces:



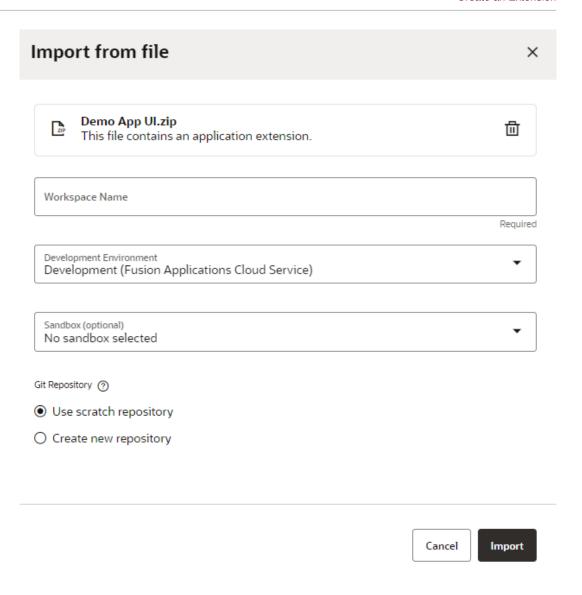
Click Import to open the Import from file dialog:



4. Drag the archive into the **Drag and Drop** area, or click in the drop area to locate the archive on your computer.

VB Studio automatically checks the content of your archive to verify that it is a valid extension archive. If it's not, you'll see a message that you can't create a workspace by importing it.

The **Import from file** screen looks like this:



- **5.** Enter a name for the new workspace.
- If multiple environments are available, select the Development Environment for your extension. Only environments that are in the same identity domain as the logged-in user are listed. (If only one environment is available, it is automatically selected for you.)

Environments with Oracle Cloud Application instances connected via Basic Auth or OAuth are not supported for extension development and will not show in the Development Environments list. To use that instance for development, you'll need to create a new environment, then add the instance as an IDCS resource using the Identity Domain option.

- 7. Choose a sandbox if you need one.
- B. Choose the Git repository you want to use with the workspace:
 - Choose Use a scratch repository if you're just experimenting;

 Choose Create a new repository (based on the one you're importing) and provide a name for both it and the branch you want to use.

9. Click Import.

When your new workspace opens, you'll see the contents of the archive you just imported in the Designer, and you can start working from there.

Related information: Export Your Workspace as an Archive

Use a Scratch Repository

When you create a workspace, you have the option of creating a scratch repository, instead of a new repository based on the main branch of the project's repository. A *scratch repository* is useful when you're experimenting and you're pretty sure you'll never want to merge your changes into an existing repository.

A scratch repository is a private repository that only exists in your workspace. Only you can use the scratch repository, and it's deleted when you delete the workspace. While this option is good for exploring ideas and trying out things, keep in mind that you won't be able to perform advanced Git operations (such as switching branches, and pulling or merging changes) from within the Designer. Also, package and deploy jobs in a CI/CD pipeline—which allow you to build and publish your artifacts simply by clicking Publish in the Designer—won't be created for you.

If you want to leverage full Git functionality within the Designer and enable collaboration, you'll need to push the scratch repository to a **new** remote repository. This way, your teammates can clone the repository and build on your changes. Further, when you push your scratch repo to a remote repo, package and deploy jobs are automatically created for you as part of a build pipeline (you can choose to *not* create this pipeline, if that's what you want). See Push a Scratch Repository to a Remote Repository.

Add a New Page to an Oracle Cloud Application

You can add a page to an App UI that you created, or to an Oracle Cloud Application (that is, an App UI created by Oracle). But suppose you want to add a few of your own pages to your Human Capital Management Cloud (HCM) application, perhaps to keep track of your employees' COVID vaccination status.

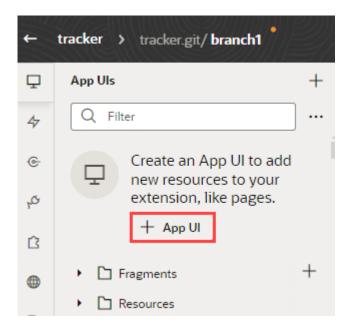
Let's take a quick look at how you might accomplish this:



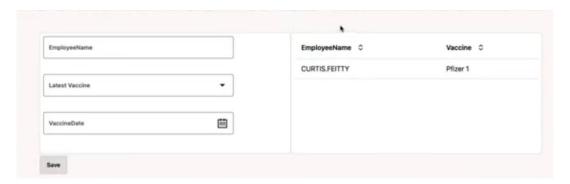
This scenario assumes that your HCM app contains a page with an extensible area, like a dynamic container, where you can add your navigation element.

- Create a new extension (including a new Git repository) by following the instructions in Create an Extension.
- Once you land in the Designer, click +App UI to create your new COVID tracker app:



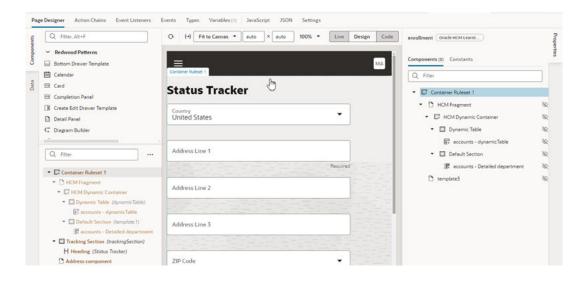


3. Use the instructions in Work With Pages and Flows to help you build your tracking app, perhaps something that looks like this:



This sample app, which accepts a parameter called <code>Employee</code>, allows the employee to enter his or her name, type of vaccine received, and date of the vaccine, then click Save to store it in the database. This information is then reflected on the right side of the screen. The data comes from two objects available from a service connection based on the Human Capital Management service catalog.

- **4.** After you build the App UI, use **Preview** to test it, then click **Publish** to deploy the extension to your Oracle Cloud Application Development environment.
- **5.** Jot down the URL of your App UI, which should be in the form: https://hostname/fscmUI/redwood/AppUIname
- **6.** Create another workspace to configure your HCM application, or clone the workspace that already exists for modifications made to this Oracle Cloud App. Remember, it's considered best practice to make all changes for a given App or App UI in the *same* Git repository.
- 7. In your HCM application, find a page with an extensible area, like a dynamic container, that you can configure. You can check this by viewing the page in the Designer and looking for extensible areas:



- 8. In the extensible area, add a navigation element to link to your new App UI. For example, you might add a simple button labeled COVID Tracker that uses the Open URL navigational element to specify the App UI's URL you noted in step 5.
- Preview and publish the extension.

Add an Application to the Oracle Cloud Applications Ecosystem

An App UI is simply a specialized application that, from a development perspective, is part of the Oracle Cloud Applications universe.

You should create an App UI only if you have a clear understanding of the audience you're creating it for and their needs. If you need an application that functions entirely independently from Oracle Cloud Applications, you probably want to create a *web application*, which is hosted on a Visual Builder instance. See *Building Responsive Applications with Visual Builder Studio* for more information.

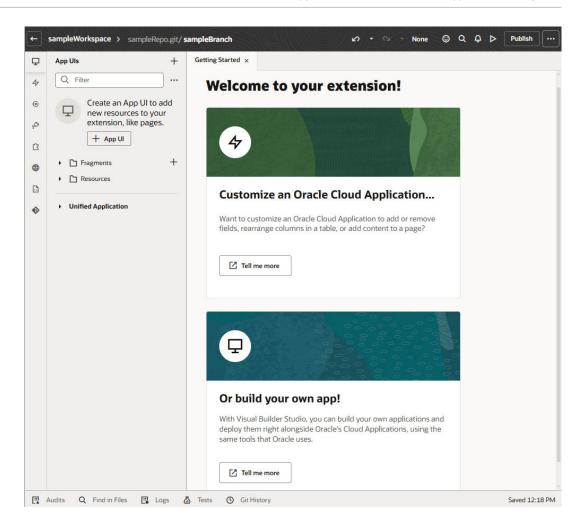


As a best practice, it's a good idea to create only one App UI in each extension for typical scenarios.

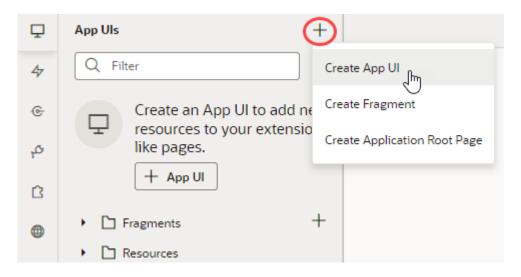
To create an App UI:

- Log into VB Studio.
- 2. Once you're in your project and workspace, your screen should look something like this:





3. Click + next to App UIs, then click Create App UI:



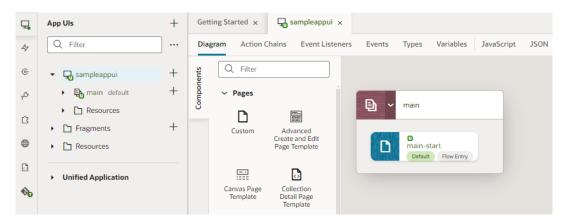
4. Enter a name for the App UI.

By default, this name is used to form the URL for the App UI, as in https://

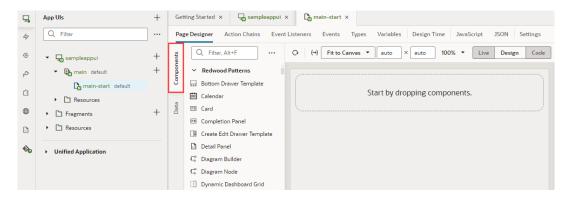
UnifiedAppName/redwood/AppUIName. However, you can use the App UI ID field to override the App UI name and supply your own string for the final portion of the URL, if

needed. Make sure you retain the x- prefix, which is used to distinguish customer App UIs from those created by Oracle in case they have the same name.

The first thing you'll see is a visual representation of your App UI on the Diagram tab:



The Diagram view can be quite handy as your App UI increases in complexity, as it displays your App UI's default pages, navigation flows, and even audit status at a glance. For now, though, you probably want to click main-start and start dropping components onto your page:



Develop an App UI or Fragment contains all the information you need to build an App UI. When you're done making your changes, there are several ways you can test and prepare your App UI for publication. See Preview, Share, and Publish Your Extension for more information.

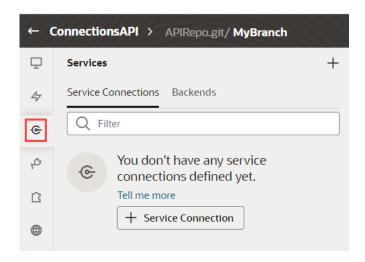
Add a Resource to the Oracle Cloud Applications Ecosystem

An extension is the vehicle you use to deliver new capabilities into Oracle Cloud Applications. In some cases, those capabilities may take the form of *resources*, like Layouts or service connections, that you want others to leverage in their own extensions.

Suppose you know that the team who plans to customize the Connections Oracle Cloud Application at your site will need access to confidential employee information provided by a REST API that is usually tightly controlled. To prevent each developer from having to request access to this REST API, you create a service connection to this data in an extension, thereby allowing each person who needs the data to gain access to it simply by adding the extension as a dependency.

Let's take a look at the steps you might follow to achieve this:

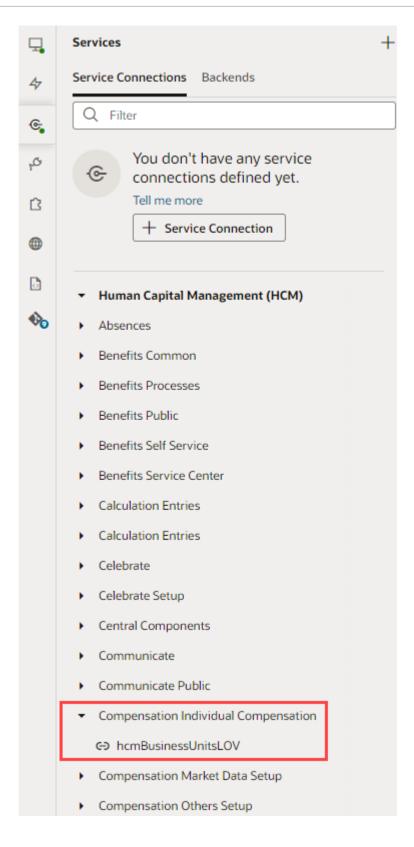
- 1. Create a new extension by following the instructions in Create an Extension.
- 2. In the Designer, click **Services** in the left Navigator:



- Click + Service Connection, then click Define by Specification in the Select Source panel.
- Provide the values required, using Create a Service Connection from a Service Specification to guide you.
- On the service connection's Overview tab, click Accessible to application extensions, to ensure that other extensions will have access to yours.
- Click Publish to deploy the extension to your Oracle Cloud Application Development environment.

Your extension is now available for others to add as a dependency. After doing so, they will see the service connection you created on their own Services tab, listed by pillar in a separate section:





The service connection is now available for use just as if the extension owner had created the service connection in his or her own extension.

Create a Bespoke Application

In Oracle Visual Builder Studio (VB Studio) terms, a *bespoke* application typically addresses a business need that lies outside the capabilities of an App UI.

You might consider creating a bespoke application if you need to:

- Access REST data sources with advanced authentication protocols;
- Create custom business objects in your own database
- Provide application access to users who don't have Oracle Cloud Application credentials
- Create a customized look and feel

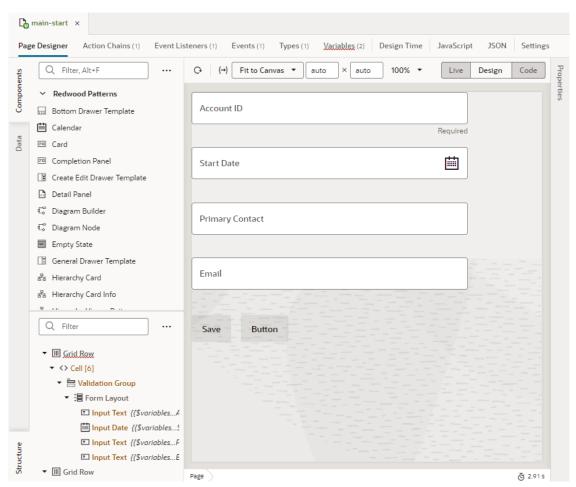
If you have one or more of these requirements, you can create a responsive application and deploy it to a separate Visual Builder instance—you cannot deploy to VB Studio itself. See Get Access to Visual Builder Instances for more information.

To get started, work through the steps in Set Up VB Studio for Developing Visual Applications. When you're ready to create your app, use Building Responsive Applications with Visual Builder Studio, or get your feet wet by completing the Create a Web Application in Oracle Visual Builder Studio workshop.

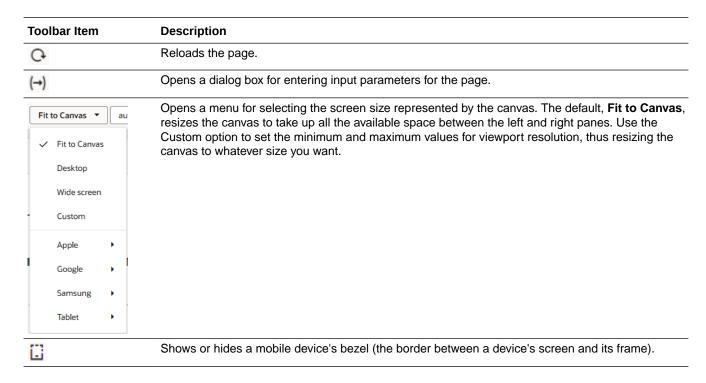
Use the Page Designer

There are several editors available to you in the Designer—such as Actions, Event Listeners, and so on—but the one you'll likely use most frequently is the *Page Designer*. The Page Designer opens when you click a page (or fragment) in the Navigator's App UI pane:





Along the top of the Page Designer you can see a toolbar, which lets you configure the Page Designer itself:



Toolbar Item 100% ▼ Live Design

Code

Description

Opens a dialog box for changing the magnification of the canvas.

Toggles between Live, Design, and Code views.

- **Live**: Displays the page as it will look and behave when it is published. Use Live view to switch to an interactive mode where you can click components and see how they'd behave; for example, you can enter data in a form and submit it as a user would and instantly see the changes take effect. Live view also gives you the option to tweak application code and render pages even when they're missing the required context.
 - You can interact with pages in Live view, however, some behavior, such as how pages are rendered or how navigation links work in the app, might be different from the published app. To fully test your application, you should open it in Preview mode in a new browser tab, instead of using Live view. See Preview Your App UI.
- Design: Contains a canvas area that you use to place and position components in the page.
 This is the mode you use most frequently.
- Code: Use to edit the page's code. In Code view, 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.

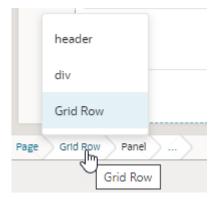


Tip:

Hold the **Ctrl** key (**Cmd** on Mac) to momentarily switch between **Live** and **Design** views. 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.

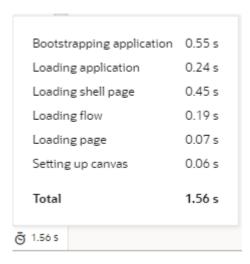


At the bottom of the canvas (not in the toolbar), a breadcrumb path displays a hierarchical list of panlinks for a selected component to indicate its placement in the page's structure. Hover over ... (the last link in the breadcrumb path) to navigate to the component's children. You can also view and switch to a component's siblings: Simply hover over a link in the breadcrumb path, then select the component you want to switch to from the menu:



Clicking a component 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 the breadcrumbs (shown by default), select **Hide Breadcrumbs** in the context menu.

Toolbar Item	Description
ē	Displays the 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 Page Designer offers several different ways for you to interact with your page. Specifically, you can:

- Drag and drop components on to the page, then associate them with data, using the Components palette;
- Work with page components from a hierarchical perspective, in Structure view;
- Start with the data, then decide how to represent the data in the user interface, using the Data palette.
- Work with the code directly, by clicking Code in the Page Designer toolbar.

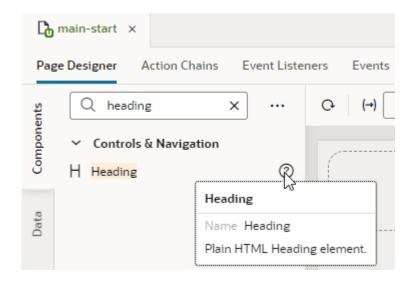
Hint: You can hide or show any of these panes by clicking the Components, Data, or Structure tab.

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 for use in your extensions. See Work With JET Core Pack Components on how you can take advantage of them in your pages.

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 App UI's audit and are retained only to allow existing App UIs 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.

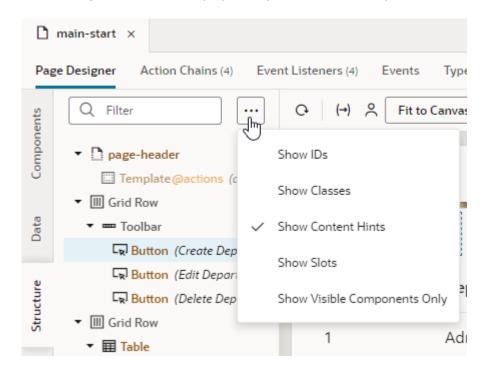
The Structure View

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.

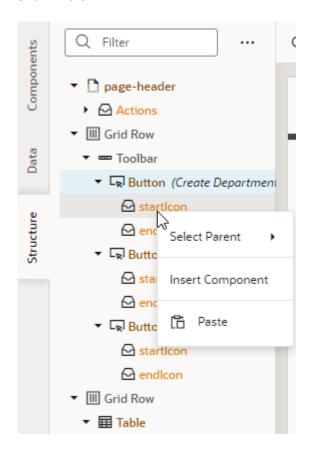


You can choose to display a component's details, for example, its id, classes, or hints about its content (enabled by default). You can use **Show Visible Components Only** to view only the nodes of the components visible on the page. When combined with the default view that fades

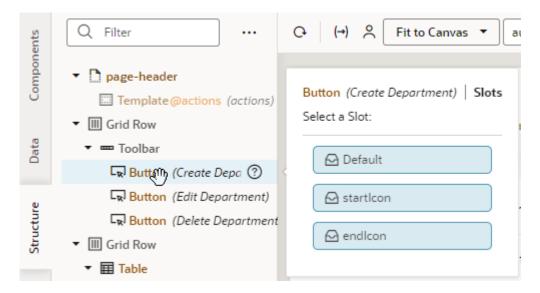


into the background components that are not currently displayed on the page, this option can trim background information and allow you to focus on parts of the page, a section at a time.

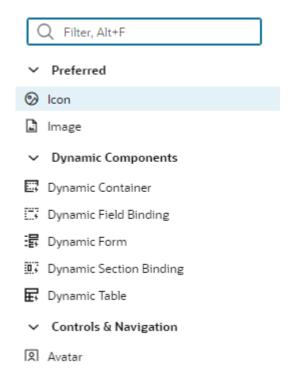
You can also enable **Show Slots** to display the location of empty as well as occupied slots. When slots are visible, you can easily locate the slot where you want to drop a component, as shown here:



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:



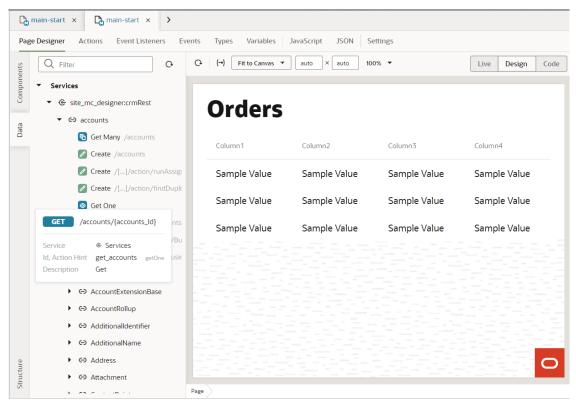
The Data Palette

The Data palette provides a data-centric approach to page design. Instead of choosing UI components and binding each component to a data source, as you would when using the Components palette, this approach starts with your data source and lets you choose from UI options that optimally display the data.

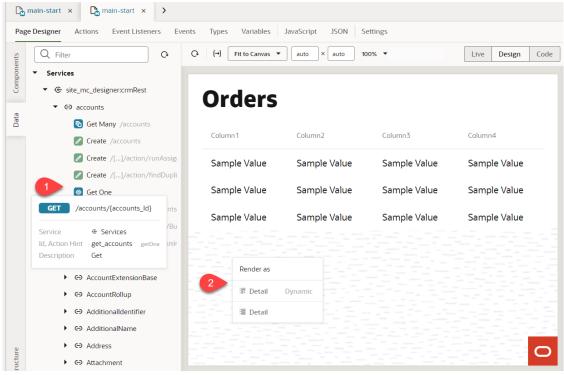
Data is the basis of any application, and when working with pages in VB Studio your data sources are always based on REST. In particular, *service connections* link your App UI with external REST APIs to retrieve data, which you can then surface by choosing a component suggested by VB Studio.

In this example, the service connection $site_mc_designer$ has been created for this extension, based on the crmRestApi in the Oracle Cloud Application catalog. Within that connection, the endpoints for the accounts REST API are displayed:



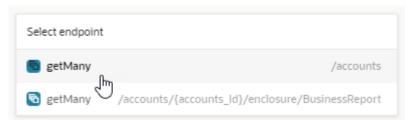


Suppose you want to show account data on a page. You would simply drag an endpoint from the Data palette and drop it on the canvas. (You can drop a data element on the page canvas, in the Structure view, or in Code view.) When the **Render as** pop-up appears, you can choose from the list of components that VB Studio suggests based on the REST endpoint you've chosen:



Notice that 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.

If a service contains multiple endpoints of the same type—for example, two <code>Get Many</code> endpoints—you'll be additionally prompted to select the correct endpoint:



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
Get Many	Table, Table Dynamic, List, List Dynamic
Create	Create Form, Create Form Dynamic
Get One	Detail Form, Detail Form Dynamic
Update	Edit Form, Edit Form Dynamic

The Data palette is available to you when working with pages, fragments, and dynamic container templates.

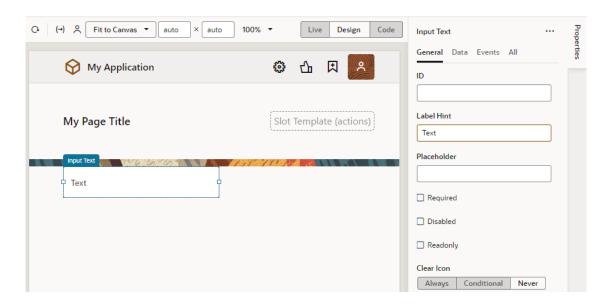
When working with quick starts in the Data palette, keep in mind that both standard and dynamic component quick starts add a form or a table to the existing page *and* configure it. Normally, quick starts for standard components *add* pages to your App UI, and quick starts for dynamic components *configure* an existing form or table—for the Data palette, quick starts do both. 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:

Properti es Tab	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.
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.

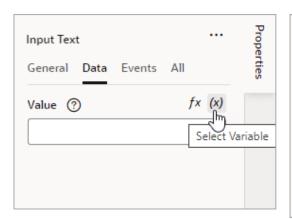
Variables Picker

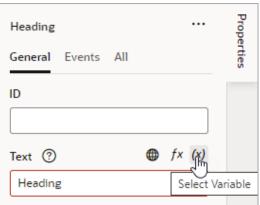
When working with a component's properties, you can bind a variable to the component, populating it with the data stored in the variable. For example, you might bind an Input Text component to a particular field from a REST endpoint, or bind a Heading component to a

variable. To do this, you use the Variables picker, which you access by clicking the $^{(x)}$ 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 (x) 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.

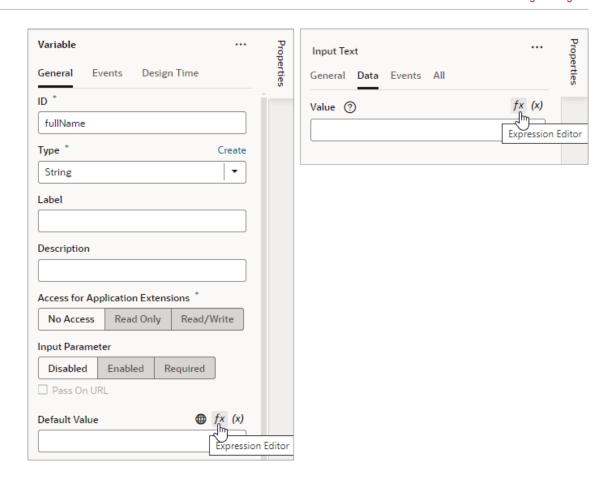
Expression Editor

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 f^x 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 f^{χ} .
- 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 VB Studio. 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 VB Studio, 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 VB Studio
To concatenate the firstName and lastName variables:	firstName + ' ' + lastName	<pre>\$page.variables.firstName + ' '</pre>
		+ \$page.variables.lastName
To check whether employee and job are not null or undefined, before trying to access jobTitle:	employee?.job?.jobTitle	<pre>\$variables.employee?.job?. jobTitle</pre>
To calculate the total price of a row based on its price and quantity:	row.price * row.quantity	<pre>\$current.row.price * \$current.row.quantity</pre>

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 To Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null



Part II

Build an Extension

The chapters in this part help you manage things at the *extension level*. You can think of an extension as a "container" of sorts for App UIs, configurations to App UIs, resources, and whatever else you choose to add to it.

To create a new extension, see Create an Extension.

Topics:

- Extension-Level Actions
- Manage Your Extension with Git
- Work With Services
- Work With Layouts in Your Extension
- Preview, Share, and Publish Your Extension
- Work With Translations



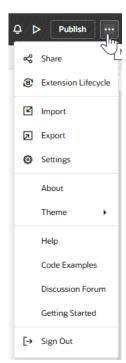
Extension-Level Actions

Extensions are what you use to deliver new capabilities into Oracle Cloud Applications.

An extension can contain configurations made to an App UI, App UIs that you create, fragments, resources that you want to share, or a combination of these. Although each of these assets have their own settings (as do individual pages within an AppUI or fragment), settings made at the extension level apply to the extension as a whole.

Establish Extension-Level Settings

To view your extension's settings, click **Menu** in the upper right corner and select **Settings**:



Here are the extension-level settings and how to use them:

Setting	Description
General	General extension settings:
Version	Your extension's version number. Although VB Studio assigns a version number when you create an extension, you can change the value at any time.
Workspace Name	Name of the current workspace (which you can also see in the header).
Project Name	Name of the current project. This is handy, as otherwise you'd have to exit the Designer completely if you wanted to know which project you're in.
Repository Name	Name of the current Git repository (which you can also see in the header, along with the current branch).
Extension Name	Name of the extension as it will appear in the Dependencies list. (Recall that Dependencies lists extension names, not App UI or any other resource names.)



Setting	Description
Extension ID	Internal extension identifier, which starts with site



If you used the Edit Page in Visual Builder Studio option to jump over to VB Studio from an Oracle Cloud Application, the names and IDs generated by default use the pillar of the page you're trying to extend (instead of its extension ID). For example, the extension name takes the format PILLAR Extension, like HCM Extension. If that name already exists, a number is added to the name and incremented as needed, for example, HCM Extension 1, HCM Extension 2, and so on.

Extension Pillar	The extension's product family or pillar, for example, Customer Experience (CX) or Human Capital Management (HCM) in the Oracle Cloud Applications ecosystem. All App UIs in this extension inherit this value.
Extension Pillar Subcategory	When Customer Experience (CX) is selected as the Pillar, subcategories provide a more granular grouping. These subcategory values— CX Sales , CX Service , and CX Marketing —show under the main Customer Experience (CX) category in the Navigator on the App UIs and Dependencies panes (as well as the Layouts, Services, and Translations panes). If you leave it as Default , this extension isn't assigned to a subcategory.
Pillar Theme	The pillar's color theme used by all App UIs in this extension for branding purposes. This setting uses the release's Default color theme for elements such as the Redwood texture stripe and background images. Select a different option if you want the App UI's pages to use a nondefault color.
Extension Description	Optional description of the extension. Before you publish your extension, it's a good idea to make sure this description is as helpful as possible, so that people who are trying to add your extension as a dependency later can understand what's in it.
Unified Application	Version number of the Unified Application, which determines the runtime dependency used by the extension. Runtime dependencies are client-side libraries that, along with the accompanying version of Oracle JET, determine features and other improvements available to the extension. See Migrate Runtime Dependencies.
Target Branch	Default target branch in your workspace's remote repository where your changes will be merged when you click Publish . This remote branch is preselected in the Publish dialog, but you can change it, if needed. See Publish Your Extension. In addition, when others merge to this remote branch, you'll be notified so you can refresh your workspace to pick up their changes.
Enable CI/CD pipeline	When a CI/CD pipeline exists for the selected target branch, this option enables or disables the CI/CD pipeline used for publishing your extension to your environment's Oracle Cloud Applications instance. See Enable or Disable the CI/CD Pipeline for Publishing. If a CI/CD pipeline doesn't exist (for example, because you created a branch for developing a feature and have not yet created a pipeline, or if you pushed a scratch repo to a remote repo and chose not to create a pipeline), this option won't show.

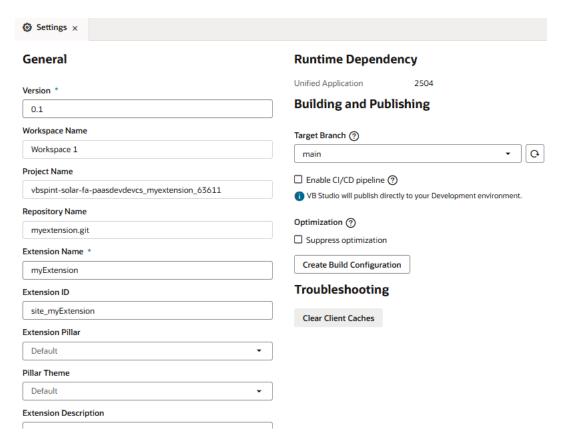


Setting	Description
Suppress Optimization	By default, all extension resources such as stylesheets and HTML files are included in an optimized resources bundle to improve performance when it is shared or deployed. To change this configuration to include and exclude resources, see Customize Optimization. To entirely suppress optimization, see Suppress Optimization.
Clear Client Caches	Option to clear the extension's resource cache. See How Do I Clear My Extension's Resource Cache?

Migrate Runtime Dependencies

In VB Studio, **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 extension, like what JET and Redwood components you can use.

To see what version you're on, click **Menu** in the upper right corner and select **Settings**. The current version is displayed under **Runtime Dependency**, as shown here:



Your extension's runtime dependency is determined by the *Unified Application*, which is the central hub of the entire Oracle Cloud Applications ecosystem. Oracle upgrades the Unified Application's internal libraries quarterly, which means that you'll be asked to upgrade your extensions quarterly as well. When your extension is due for an upgrade, you'll see a banner like this in the Designer:





If you have any outstanding changes, you'll be asked to commit, stash, or revert your changes before you can upgrade. Click **Show Changes** to go to the Git panel and take action.

When you're ready to upgrade, click **Migrate** in the Settings editor to begin the process.

Occasionally, Oracle may add additional migrators outside of the normal upgrade process, in which case you'll be notified that new migrators are available. You can then migrate your app from the Settings editor. Again, if you have uncommitted changes, you must click **Show Changes** and take action for the **Migrate** button to become active:

Runtime Dependency

Unified Application 2504

Migrate

We've added new migrators since the last time this app was upgraded.

Make sure you commit, stash, or revert any outstanding changes first. Show changes.

Resolve Upgrade Issues

When you access an application following an upgrade, VB Studio tries to update the application's metadata for compatibility with the newer version. If this update fails, VB Studio 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 (extension.json) cannot be parsed. This can happen when there's an unresolved merge conflict or the JSON syntax is invalid. It can also occur when there's a version mismatch between your VB Studio instance and the environment's Unified Application. For example, if you're extending an App UI on VB Studio 23.01 with the underlying Unified Application on 22.10 (as indicated on the app-level Settings editor), opening the app will bring it up in recovery mode.

If your app launches in recovery mode, 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 VB Studio 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.

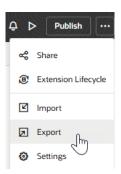
Once you resolve all issues, click **Migrate** to restart and complete the upgrade. Click the button as often as required until all issues are resolved.

Export Your Workspace as an Archive

When you export an extension, the exported archive contains the files in your workspace's branch of the extension's Git repository.

You may want to export an extension so that someone else can import it when creating a workspace or when replacing source files in their existing workspace. In this way, you can collaborate with other team members by giving them a copy of your local Git repository.

To export an extension when working in a workspace, click **Menu** in the upper right corner and select **Export**.

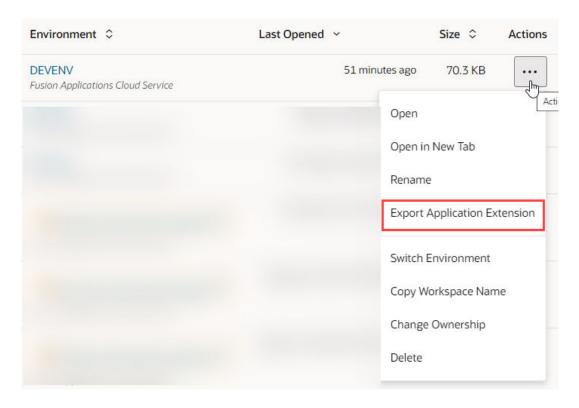


A confirmation displays, and after a moment or two, you'll see your current workspace's contents downloaded as an archive to your local file system.

To export an extension from your project environment where you see all your workspaces:

- Click at the top left of the header to get to the VB Studio navigator.
- 2. Click Workspaces in the VB Studio left navigator.
- 3. Click a workspace's Actions menu and select Export Application Extension:





An archive that contains the workspace's contents is downloaded to your local file system.

Import Resources to Your Workspace

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

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

The resources are imported to your extension's root directory.

Migrate Imported Resources

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

To migrate these resources:

Click Migrate in the banner.





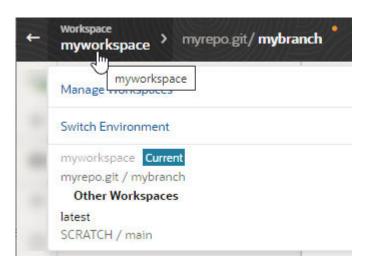
If your workspace has changes that are yet to be committed to your Git repo, you'll need to commit, stash, or revert them before you can migrate resources. Click **Show changes** to go to the Git panel and take action, then click **Migrate** in the banner.

- In the Upgrade Imported Files dialog, click Upgrade to make your app's resources compatible with its current runtime version.
- 3. Click **Commit Changes** to commit the changes to your Git repo.

Switch a Workspace

When you're working on things in parallel, you might need to switch between workspaces to continue your work. It's easy to do that in the Designer.

1. With your workspace open in the Designer, click the workspace name in the header.



2. Select the workspace you want to switch to.

Note:

If the workspace you want to switch to is tied to an environment in a different identity domain, you'll be prompted to switch the workspace's environment before the workspace can open in the Designer.

Switch a Workspace's Environment

You may need to switch the environment your workspace uses when the old one is down, or has been decommissioned, repurposed, or replaced. By switching environments, you can resume development in the same workspace without having to push changes to the remote repository, then creating a workspace that uses the new environment.

To switch your workspace's environment, you need at least one other environment for the type of project you're working on: another Visual Builder instance or Oracle Cloud Application

instance (if you use VB Studio to access the built-in catalog of Oracle SaaS/PaaS REST services or to extend Oracle Cloud Applications). If no other environments are available, ask your project owner or an administrator to create one before you try to switch environments. Then follow these steps:

 With your workspace open in the Designer, click the Workspace menu in the header, then select Switch Environment:



The Switch Environment dialog notifies you of the environment you're using and presents a list of other environments that are in the same identity domain as the logged-in user. (The environment currently associated with your workspace is not included.)



Tip:

You can also switch a workspace's environment on the Workspaces page, using the **Switch Environment** option in the workspace's Actions menu (•••).

If multiple environments are available, select the environment you want to use. If only one environment is available, it is automatically selected for you.

- Optional: If your selected environment contains an Oracle Cloud Application instance, select a sandbox.
- 3. Click Switch.

4

Manage Your Extension with Git

Git lies at the heart of your work with extensions, whether you're configuring an App UI that Oracle built or creating one of your own. A *Git repository* contains your source files for the changes made to a given App UI.

If you're relatively inexperienced with Git, you may find it helpful to watch this video, which explains concepts like remote and local repositories, as well as how they tie in to Visual Builder Studio workspaces:

() Video

We've designed Visual Builder Studio so that you don't really have to learn Git if you don't want to. The actions described in Preview, Share, and Publish Your Extension are likely sufficient for you to complete work on your extension. However, the more you understand about how Git works within VB Studio, the more flexibility and power you'll have within the tool. If you'd like to learn a bit more, we recommended that you familiarize yourself with these topics before you start working in the Designer:

- · Get Oriented with Git
- Make Your Changes Public
- Resolve Conflicts Using the Git Panel

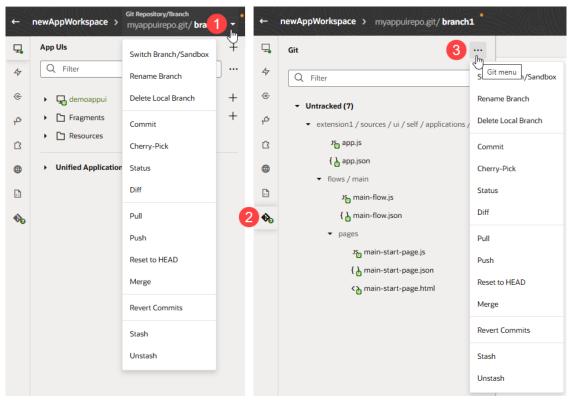
The rest of the topics in this chapter are geared towards users who are more advanced in their use of Git.

Get Oriented with Git

Within a workspace, the Designer connects you to the project's repository, enabling you to switch branches, push and pull sources, and merge changes—all through Git commands in the Designer.

You can access Git commands from two locations in the Designer:

- In the header, by clicking the arrow next to your repository name and working branch (label 1 in the image).
- In the Git panel, which you can display by clicking Git (label 2) in the navigator. The Git panel shows you the status of your workspace files (for example, whether they are changed, untracked, or in conflict). It also includes a menu (Label 3) that lets you access the same Git commands as in the header. You can also perform operations at the file level in the Git tab.



As you make changes in the Designer, you'll notice the Git tab badged to indicate status. You can use this badge view—without actually accessing the Git tab—to get a quick Git summary of your workspace. You'll see a count of the files that have changed in the workspace as well

as a color indicator of the type of changes; green () indicates new files, blue ()

indicates modified files, and red (**) indicates files that conflict with changes made by other team members. When conflicts exist, the Git Panel provides tools to assist as you review and resolve issues.

A series of overlay icons also appear when you create, modify, or delete files. These icons indicate the status of the artifact or file (uncommitted, conflict, or recently added) with regard to the branch that your workspace uses.

Here is a description of the icons that you'll see:

Status Icon	Status Description
L _d	The file has never been committed, possibly because it was just created.
C _d	The file has been added, but it won't become tracked until the next commit.
	The file has not been modified since it was last committed.
<u> </u>	The file has modifications that have not been committed.
{ <mark>}</mark>	The file has conflicts that are not yet resolved. You'll usually see this status icon appear during Pull or Merge operations (or during a Cherry-Pick).



Status Description Icon



The file has been deleted, but the change is yet to be committed. You'll see this status icon only in the Git Panel.

Pull Changes from the Remote Branch

Use the **Pull** option to update your workspace with commits from the remote branch.

Pulling the most recent changes from the repository is especially important when you and other team members work on an extension, to keep the source files in your workspace up-to-date with everybody's changes. It can help you avoid making changes to files that someone else is editing or has edited, which can result in a code conflict that you would then need to resolve.

To pull changes from a remote branch into your workspace:

- Click the Git menu in the header or in the Git panel.
- Select Pull.
- 3. From the Remote Branch list in the Pull (Rebase) dialog box, choose the remote branch that has the content you want to pull to your workspace.
- **4.** If your working branch contains uncommitted changes, you have the option of committing those changes.
 - To commit your changes before pulling in content from the remote branch, enter a commit message (with the Commit changes before pulling checkbox selected).
 - To pull content without committing your changes, clear the Commit changes before pulling checkbox.
- If you chose to commit changes, click Commit All and Pull; otherwise, click Pull Changes.

Make Your Changes Public

When you edit files in your workspace, you're working within your *local* repository, which means that only you can see them. To share your private changes with your team, you'll need to push them to a shared project repository (the *remote* repo).

When you make changes in your workspace, an yellow dot appears next to the branch name in the header to let you know that you have uncommitted changes:

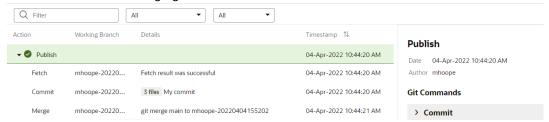


Saving your changes to a remote repo is a two-step process: **commit** and **push**.

1. The first step you must perform is a commit. When you make a commit, you're basically grouping together the files in your local branch that you eventually want to move to the remote branch. You also provide a description of the changes you're making in the commit, which can be very helpful later if you need to track down which files were changed.

For example, suppose you added a new dynamic layout to include a field. When you committed the changes, you added a commit message describing the update, "Added new

layout X for field Y". You then added a rule for the new layout, using the commit message "Added rule Z for layout X". By reviewing the history for the repository (using the **Git History** tab at the bottom of the Designer), you can easily see what files were changed and the reasons for changing them:



2. The second step is **push**, which synchronizes all the files you have committed thus far (since the last push) with the remote branch. If you have edited some files but haven't committed them, you'll be prompted to commit before you can push.



You can use the **Publish** action in the header if you want to commit, push, and merge the remote branch into the main branch in one operation. See **Publish Your Extension**. Don't use **Publish** if you're not ready to merge the remote branch into the main branch.

Commit Your Changes to the Local Branch

As you work on your application in the Designer, you can use the **Commit** option to save your changes to the local branch.

When you make a commit, you're basically grouping together the files in your local branch that you eventually want to copy to the remote branch. You also provide a description of the changes you're making in the commit, which can be very helpful later if you need to track down which files were changed.

Commit your changes as often as you can, so you have a string of commits with messages that clearly describe your updates.

To commit changes in your workspace:

- 1. Open your workspace.
- Click the Git menu in the header or in the Git panel, and select Commit.
 - VB Studio checks the status of your working branch and identifies files that have changed since your last commit.
- 3. If you're not ready to commit some of your changes, scroll through the **Items to Commit** and deselect the files you don't want to commit.
- 4. Enter a commit message that describes the changes you've made in the remaining files.
- Click Commit.

When the commit is successful, close the confirmation message that appears.



Push Your Changes to the Remote Branch

Use the **Push** option to move changes you've committed from the local branch in your workspace to the remote branch, thus making your changes visible to team members. You'll need to explicitly commit your changes before you can push them to a remote branch.

Note:

Before you commit and push your changes, you should update the source files in your workspace by pulling the most recent versions from the repository. If the file versions in the remote branch are newer than the versions in your workspace, you'll see a status message when attempting to push your changes that the push was rejected and that you should pull the most recent versions from the repository.

To push your commits from the local branch in your workspace to the remote branch:

- 1. Click the Git menu in the header or in the Git panel.
- Select Push.

If there are no changes to push, click **Close**. Otherwise:

- To push committed changes to the remote branch, click Push Changes.
- To commit all uncommitted changes, enter a commit message and click Commit All and Push. If you don't want to commit uncommitted changes, but want to push previously committed changes to the remote branch, clear the Commit changes before pushing checkbox.

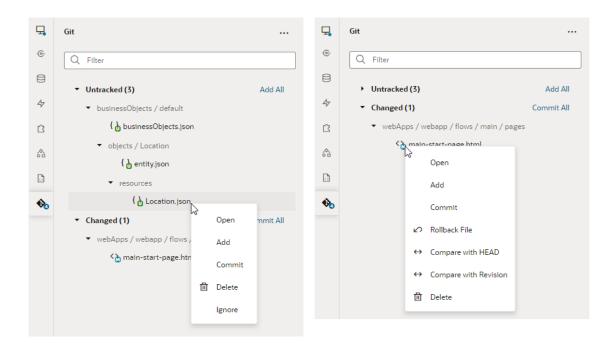
When the push is successful, close the confirmation message that appears.

File-Management Operations

When you work with files in the Git panel, you can perform some operations directly from there, even at the individual file level.

The following image shows the options available to you based on a file's status:





Action	Steps
Open	Right-click an untracked or a changed file and select Open to view the file in the corresponding visual editor. For example, opening the main-start-page.json will show the main-start page in the JSON editor.
Add	Right-click an untracked or a changed file and select Add to mark the file for inclusion in your next commit to the current branch. You can also click the Add All link to add all unstaged files (both untracked and modified files that haven't yet been committed), or to select the files you want to add. After an untracked file has been added, it'll move from the Untracked category to the Changed category.
Commit	Right-click an untracked or a changed file and select Commit , enter a message, and click Commit to commit the file to your current branch. You can also click the Commit All link next to Changed to group and commit multiple changed files.
Rollback File	Right-click a changed file and select Rollback File , then confirm to revert modifications made to the file since its last commit to the current branch. This option is available only for a file with changes that have been committed to the branch. You won't see this option when your workspace is in an interrupted state (for example, when conflicts occur during Merge or Pull operations).
Delete	Right-click an untracked or a changed file and select Delete to remove the file. Deleting a untracked file removes it from your next commit, but deleting a changed file removes a file that was previously committed to your branch.



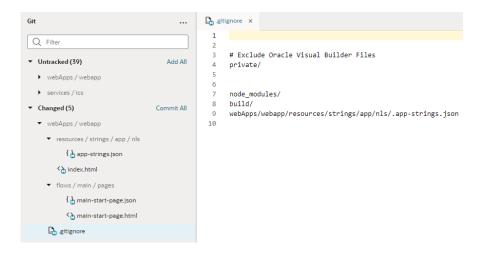
Action

Steps

Ignore

Right-click an untracked file and select **Ignore** to exclude a file from being tracked. Typically, you do this for generated files that you don't want to be tracked in Git. Ignoring a file removes it from the Untracked category and adds a <code>.gitignore</code> file in the Changed category.

If you want to start tracking an ignored file, double-click the .gitignore file to open it, then delete the file's entry from the list:



Compare with HEAD

Right-click a changed file and select **Compare with HEAD** to compare the file's changes in your workspace with the latest on the current branch in the repo (known as HEAD in Git terms). When you compare a file with HEAD, you are comparing changes you've made to the file but not yet committed with the commit at the very tip of the current branch.

You won't see this option for non-text files or files with conflicts (conflicts can occur during Merge or Pull operations).

Compare with Revision

Right-click a changed file and select **Compare with Revision** to compare the file's changes with a revision. You won't see this option for non-text files or files with conflicts (conflicts can occur during Merge or Pull operations).

You can compare a file against a revision, a branch, or between two revisions:

- Use Revision to compare the file with a revision on the current branch (default) or on another branch. Use search to find a revision by its ID, especially if it might be in another branch (see Get the Revision ID of a Commit).
- Use Different Branch to compare the file's changes with the tip of another branch, local or remote.
- Use Between Two Revisions to compare any two revisions of the file on the current branch.



Action Steps

For large files that take longer than the default 10 second diff processing timeout, you'll be prompted to show the incomplete results, or retry the diff calculation using the maximum 30 second processing timeout. In both cases a progress indicator displays the diff calculation progress.

When you compare a file, you'll see a read-only view of the file's changes in a diff viewer. The diff view will automatically refresh if the file's content changes.



Use the options in the toolbar to view revision information, navigate between changes, and change how the output is displayed:

Icon	Description
\uparrow	Navigate to previous change in diff view
\downarrow	Navigate to next change in diff view
→	Show all unchanged lines
<u></u>	Hide all unchanged lines
Œ	Show diff view inline
	Show diff view side by side
\$ ×	Show unchanged region
×	Fold unchanged region

Refresh Your Workspace

When collaborating with others, it's important to keep your workspace up-to-date with everyone's changes to make sure you're working with the latest files and avoid changes that



overlap or conflict with each other. You can do this manually via a Git merge or pull—or simply refresh your workspace, when notified, to bring the latest updates into your workspace.



For optimal performance, the option to refresh your workspace takes effect only in workspaces with fewer than 1000 files.

Refresh notifications depend on the target branch to which changes are merged in your project's remote repository. By default, this is the main branch, meaning you'll be notified only when changes are merged to the main branch. If you want to be notified when changes are merged to a different branch in the remote repo, select that branch as the target in the Settings editor **before** you start making changes in your workspace. This way, you'll be notified whenever changes are made to that branch and can refresh your workspace to pick up those changes.

In addition to merged changes, you can refresh your workspace to pull changes from your current branch in the remote repository. A *merge* is used when changes exist in the target branch of the remote repository; a *pull* is used when changes exist in the remote version of the workspace's branch. When both types of changes exist, the pull takes precedence.

Here's an example of how changes in the remote repository trigger a refresh notification and prompt you to merge the updates: Assume your remote repository contains the main and feature branches. You now create a workspace (with the target branch set to main by default). To start your work, you switch branches and create a new local branch called mywork, based on the feature branch. Because you intend to publish your changes from mywork to feature, you set the target branch to feature in the Settings editor, then start making changes in your workspace. While you're working, a teammate publishes their changes to the remote feature branch. You receive a notification prompting you to refresh your workspace. You do this, merging the latest changes from feature into mywork in your workspace. Once your work is complete, you click Publish to seamlessly merge your changes from mywork into feature in the remote repository.

Here's what to do when you're notified to refresh a workspace:

1. Click **Refresh Workspace** in the notification. Here are examples of what you may see depending on the Git operation required to refresh your workspace:





Tip:

The notification to refresh your workspace is an informational message that clears after five seconds. If you missed the notification or want to take action later, look for the message under **Notifications** in the header. It's helpful to periodically review unread notifications, so you don't miss important information. VB Studio checks for changes in your repository every 10 minutes.

2. Click Merge or Pull Changes, then Close.

If you run into conflicts, click **Show Conflicts** to go to the Git panel and take steps to resolve the conflict. See Resolve Conflicts Using the Git Panel.

Resolve Conflicts Using the Git Panel

Use the Git panel to review and resolve conflicts that occur when your changes clash with those made by other project members.

Conflicts usually occur when you and your teammate change the same line in different ways, or if one of you deleted a file while the other modified it. In such cases, VB Studio cannot tell which version is correct—it's up to you or your teammate to make that decision and resolve the conflict.

Note:

To help minimize the number of conflicts, the best practice is to make sure your local branch and the remote branch are in sync and up-to-date. Typically, you make changes in a branch private to your workspace, then publish them to a shared branch in your remote repository, so your changes become available to others. To include others' changes in your workspace, you must keep your local branch in sync with the remote branch by merging changes from that remote branch into your workspace.

VB Studio simplifies the task of keeping your workspace up-to-date by periodically checking your repository for updates and notifying you when your workspace needs to be refreshed. Simply click the **Refresh Workspace** link in the notification to update your workspace with the latest changes and avoid conflicts. See Refresh Your Workspace.

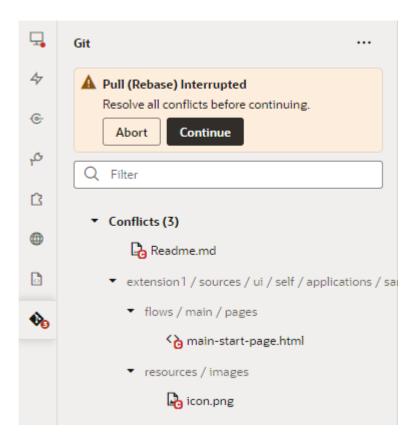
You'll be informed of conflicts when you attempt to merge, pull, or revert commits in your workspace. For example, let's say you've changed lines 2 and 3 in readme.md and committed the changes to a local branch in your Git repo. Now if someone else modified the same lines in readme.md and committed the changes to a remote branch, you'll be warned of conflicts between the file's remote version and your local version when you do a pull request to refresh your workspace, as shown here:





When this happens, here's what to do:

 Click Show Conflicts to open the Git panel and view the files identified as containing conflicts:



If you don't want to continue the operation, click **Abort** in the Git panel to return your local repo to its previous state (the last commit made to the branch, known as HEAD)—but you

won't be able to refresh your workspace until you abandon your local changes or resolve the conflicts.

To resolve conflicts and continue your operation, right-click the file listed under Conflicts in the Git Panel and take action.

The options that show depend on the type of conflict in the file. Here's an example of what you see when people make different changes to the same line of the same file (readme.md):



Action	Steps	
Open in conflict editor	Select this option to open the file in the conflict editor; you can also just click the file to open it in the conflict editor. The conflict editor has controls to help you navigate between conflicts and provides options to resolve them. See Use the Conflict Editor to Resolve Conflicts.	
Open in default editor	Select this option to open the file in the default editor, which are the designated artifact editors in VB Studio (for example, the Page Designer for an .html file or the JavaScript editor for a .js file). This option is useful for non-text files (such as Excel worksheets and schema files) or other artifacts that aren't supported in the conflict editor. You can then open these files in the default editor and manually resolve the conflicts.	
Resolve	Select this option to quickly resolve the file's conflicts, instead of going through each conflict in the conflict editor. This option is useful when the file has only a few conflicts that you can easily resolve by selecting either your version or the other version. See Use the Context Menu to Resolve Conflicts.	
	Note: Resolve is not an option for non-text files. If you run into conflicts for binary files (say, an image file),	

Delete Select this option to delete the file from the Git repo.

It's also possible to review and resolve merge conflicts from VB Studio's Git page. See Resolve a Merge Conflict.

you'll need to delete the file from the Git repo, then add it again after you've resolved all other conflicts

and committed them to the remote repo.

- When you have many files with conflicts, you'll need to right-click each file in the Git panel and take action.
- 4. Once all the conflicts are resolved, click **Continue** in the Git panel (or header menu) to continue your interrupted pull, merge, or revert operation.
- 5. Remember to push your changes to the remote repo.

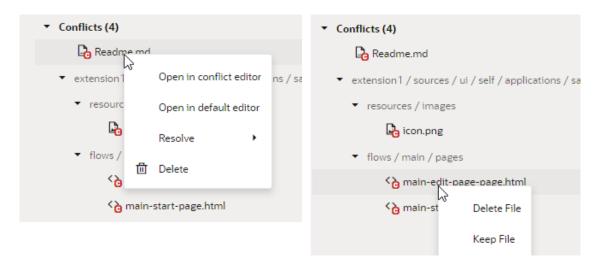
Use the Context Menu to Resolve Conflicts

When a file contains conflicts, you can use its context menu in the Git panel to quickly resolve conflicts. You do this when the file has just a few simple conflicts that you can easily resolve by keeping your branch's changes or the other branch's changes.



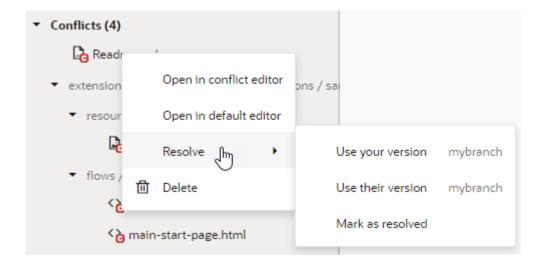
If you want to keep changes from both branches (or a combination of them), or if the file has a large number of conflicts, it's better to open the file in the conflict editor and resolve each conflict separately.

The options available to you in the context menu depend on the type of conflict in the file. For example, what you see when a file is deleted in one branch and modified in another (as shown here on the right) is not the same as what you see for a file modified by different people on different branches (shown on the left):



For demo purposes, we'll use the example of a file that's been modified by you and someone else.

1. Right-click the file with conflicts in the Git panel and select **Resolve**.



- 2. Select the option you want to use:
 - Select Use your version to keep your changes in the local branch.
 - Select Use their version to keep changes in the remote branch.
 - Select Mark as resolved to mark the file as resolved by staging it in Git.



Tip:

When you select a file version (**Use your version** or **Use your version**), the file is automatically marked as resolved. Select **Mark as resolved** only if you manually made changes to a file either in the default editor or the conflict editor and did not mark the file as resolved (using **Resolve and Close**) in the editor.

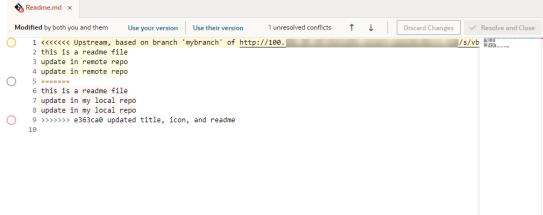
Use the Conflict Editor to Resolve Conflicts

When you open a file with conflicts in the conflict editor, you can use the available controls to navigate between conflicts and resolve them.

Right-click the file with conflicts in the Git panel, then select **Open in conflict editor**. You can also double-click the file, especially if the type of conflict doesn't give the option to open the file in the conflict editor.

When the file opens in the conflict editor, conflicts between your local version of the file and the remote version are highlighted. For example, here's an example readme.md file (with





Decide how you want to resolve the conflict. You can use the options in the toolbar or the markers that appear next to a conflict in the editor.

The options available to you depend on the type of conflict you are resolving. For example, if you deleted a file in your local branch and someone modified the same file in the remote branch, you might see **Delete File (Your Version)** and **Keep File (Their Version)**. (It's worthwhile to note that the content of deleted files shows in read-only mode and cannot be edited until the conflict is resolved. Note also that non-text file content won't show in the conflict editor.)

For demo purposes, we'll use the example of a file that's been modified by you and someone else. Here are the options available to you for this use case:

Action	Step
To keep changes in the remote branch	Select Use their version in the toolbar, or
	 Click in the canvas and select Their Version.
To keep your changes in the local branch	Select Use your version in the toolbar, or
	 Click in the canvas and select Your Version.
To keep changes from both branches	Click in the canvas and select Use Both Changes.

Note:

You can also ignore these options and update the file as you would any text file, to keep the lines you want and delete the rest. You do this when you want to incorporate changes from both branches, for example, to keep some of your changes and some of theirs.

The lines between <<<<< and >>>>>> represent a conflicting change, with changes from each ref separated by ======. Remember to delete these markers as well as Git-related comments when resolving a conflict.

3. If the file has more than one conflict, click \checkmark in the toolbar to go to the next conflict and take steps to resolve it.

- 4. Optional: Click **Discard Changes** and confirm when prompted to revert *all* the updates you've made to resolve conflicts in the file.
 - Once you discard your changes, resolve the conflicts again to proceed.
- When you see the No unresolved conflicts message in the toolbar, click Resolve and Close to mark the file as resolved.

Stash Your Changes

Sometimes as you make changes in your workspace's repository, you might need to work on another task even when you're midway through some other change. If what you're working on isn't quite ready to be committed, you can temporarily save—or **stash**—your changes, switch context to do something else, then come back and pick up where you left off.

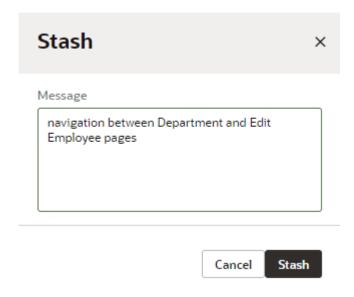
Create a Stash

You can create a stash when you want to save uncommitted changes in your extension. This way, you keep the progress you've made so far without having to commit unfinished, potentially broken, changes.

Stashing saves all the changes you've made in your working branch but not yet committed and reverts the branch's contents to HEAD (the last commit made to the branch). With your changes safely stashed away, you can switch context and freely make other changes. Then when you're ready to resume your work, you can apply the stash and continue where you left off.

To stash uncommitted changes in your working branch:

- 1. Click the Git menu in the header or in the Git panel.
- 2. Select Stash.
- 3. In the Stash dialog box, enter a message. If you don't, the stash will be marked by default as a work in progress on top of the branch and commit that you're creating the stash from, something like WIP on branch: revisionID last-commit-message. This default might not be relevant to your particular changes, so providing a good description can help you clearly identify the changes you're saving.





Click Stash.

When successful, close the confirmation message that appears. The Git Panel also refreshes to show that there are no more uncommitted changes for the current branch.

You can now switch context to perform any Git operation relating to your new task.

To view your stash, click **Unstash** in the header's Git menu. Stashes are visible across branches, so you can save a stash in one branch, then switch to another branch and apply your stash there.

Apply a Stash

When you are ready to work on something you previously stashed, you can apply that stash to bring back your saved changes and pick up where you left off. You can either **apply** a stash or **pop** it to your working branch.

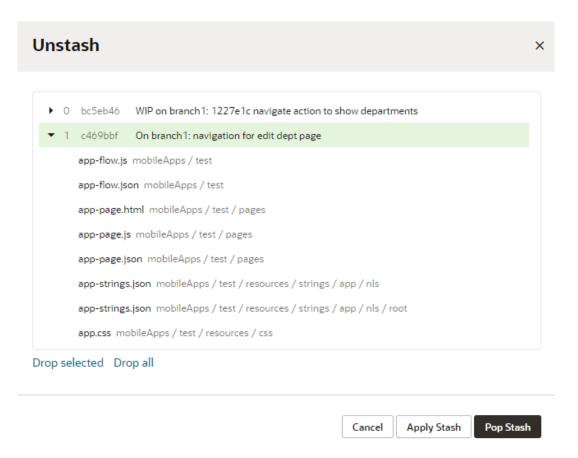
- Applying a stash restores its changes *and* keeps the stash in the stash list. Do this if you want to apply the same stash to multiple branches.
- Popping a stash restores its changes and removes the stash from the stash list.

To restore changes saved in a stash to your working branch:

- 1. Click the Git menu in the header or in the Git panel.
- Select Unstash.

You'll see a list of stashes, both from your current branch and from other branches.

3. Locate your stash in the Unstash dialog box. You can expand the item to view its list of changed files and check if they include the changes you want to restore.





4. Apply your stash:

- To restore a stash to your current branch and keep it in the stash list, click Apply Stash.
- To restore a stash to your current branch and delete it from the stash list, click Pop Stash.

When successful, close the confirmation message that appears. You can then continue from where you left off.

If you've made local changes that will be overwritten by other changes in the stashed files, the stash won't be applied and the operation will be aborted. You'll then need to commit those changes or stash them before you can re-apply the stash you wanted to apply first. If applying your stash causes conflicts, you need to resolve the conflict before you can resume your work.

Delete a Stash

When you no longer need a stash, delete it as a best practice. You can also delete all your stashes to keep your stash list clean. Remember though you can't recover a stash after you've deleted it.

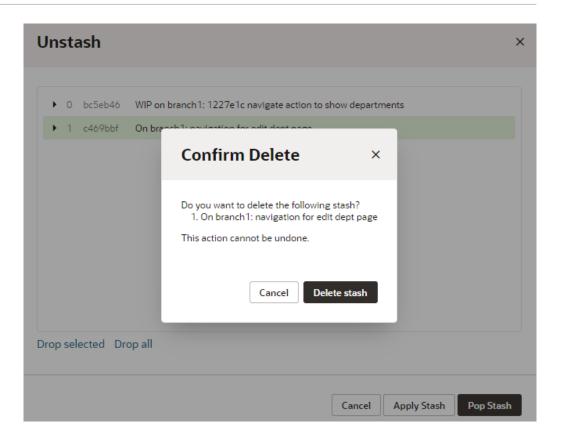
To delete a particular stash or to delete all stashes:

- 1. Click the Git menu in the header or in the Git panel.
- 2. Select Unstash.

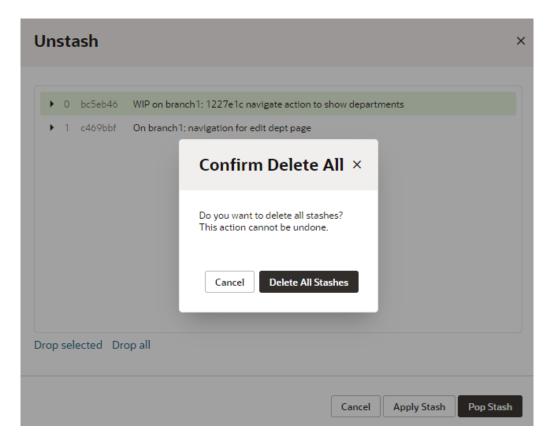
You'll see a list of stashes, both from your current branch and from other branches.

- Delete your stash:
 - To delete a particular stash, select it in the Unstash dialog box. You might want to
 expand the item and check its list of changed files to make sure you're okay to discard
 the saved changes. Click **Drop selected**, then **Delete Stash** to confirm.





 To delete all stashes, click Drop all in the Unstash dialog box, then Delete All Stashes to confirm.



Revert Commits

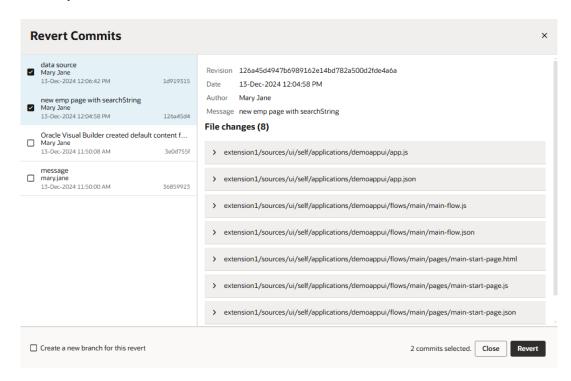
You can undo committed changes by reverting one or more commits from your repository's history. Reverting can help you undo commits that perhaps introduced a bug, or back out changes that were accidentally merged to your repository's branch.

When a commit is reverted, the changes from the particular commit are removed from your local workspace, and a new commit is created to reflect your repository as it stands with the removed commits. This way, it's possible to even undo a revert.

Before you revert commits, make sure you commit or stash any pending changes in your workspace if you need them, or reset or rollback changes if you don't need them.

To revert one or more commits:

- Click the Git menu in the header or in the Git panel.
- 2. Select Revert Commits.
- 3. In the Revert Commits dialog, select one or more commits to revert in the current branch. Use the details pane on the right to view the changes in each commit. Details of the commit you last selected will show:



Note that reverting merge commits (those that have more than one parent) is not supported, so you won't see those commits in the Revert Commits dialog.

- 4. Optional: As a best practice, select Create a new branch for this revert, and enter the name of the branch that will be created based on the current branch (for example, revert/ExperimentalChange). Creating a separate branch is recommended when you want to isolate your changes to a particular branch in your local workspace and later create a code review on them (merge request), or deploy the changes for testing before merging them to their target.
- Click Revert.



- If your commits were successfully reverted, you'll see a notification in the lower right corner.
- If you run into conflicts, you'll need to first resolve the conflicts. Click Show Conflicts
 to open the Git panel and resolve the conflicts. Once you do that, click Continue to
 proceed with your revert.

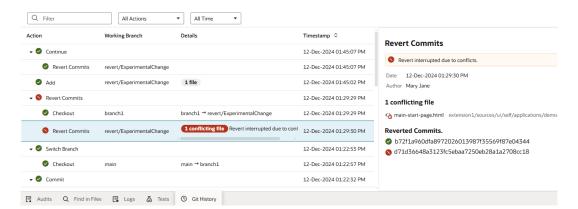


Alternatively, click **Abort** to discard the current revert operation for all selected commits, which resets the branch to its state before the revert operation was started. This means:

- Any commits reverted during the current operation are removed,
- Any conflicts in files are reset, and
- Any pending commits that were to be reverted after this commit are canceled.

Once aborted, you'll need to start another revert operation with the commits you want.

6. Click Git History to check your Git actions. Here's an example of a revert operation for two commits, where one commit was successfully reverted and the other was interrupted because of conflicts; once the conflict is resolved and the operation continued, the commit is successfully marked as reverted:





Cherry-Pick Commits Between Branches

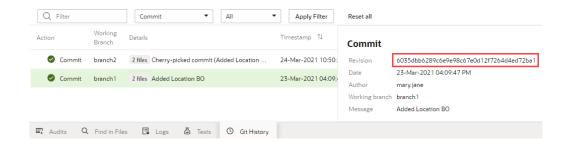
You can cherry-pick commits when you want to apply a commit from one branch to another in your workspace. Typically, you'd cherry-pick commits from a remote branch to the working branch in your workspace.

Cherry-picking is useful when you work with multiple Git branches in a repository, either on your own or in a team environment. When you only need a specific commit from another branch, it's easier to cherry-pick that commit, instead of merging that entire branch to a remote repository. You can cherry-pick commits only between branches in the same repo.

Get the Revision ID of a Commit

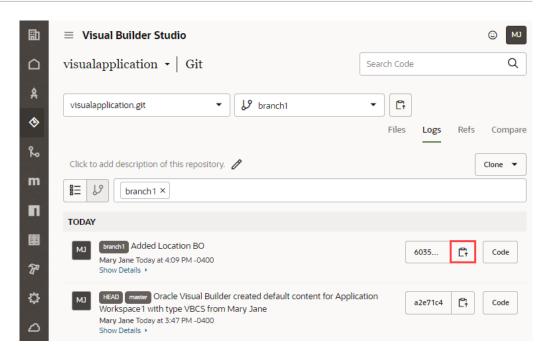
Before you can cherry-pick a commit, you'll need the ID of the revision you want to include in your branch.

- If the commit is local to your workspace, follow these steps:
 - Click the Git History panel in your workspace.
 - 2. Select Commit from the action filters and click Apply Filter.
 - 3. Select the commit you want, then copy the revision ID in the details pane:



- If you or one of your teammates already pushed the commit to a remote repository, follow these steps:
 - Click at the top left of the header to get to the VB Studio navigator.
 - 2. In the VB Studio left navigator, click **Git** .
 - 3. If required, select the branch with the commit you're looking for.
 - 4. Click the Logs tab and find the commit.
 - 5. Click **Copy revision to clipboard** (\Box) to copy the revision ID.





Cherry-Pick a Commit From Another Branch

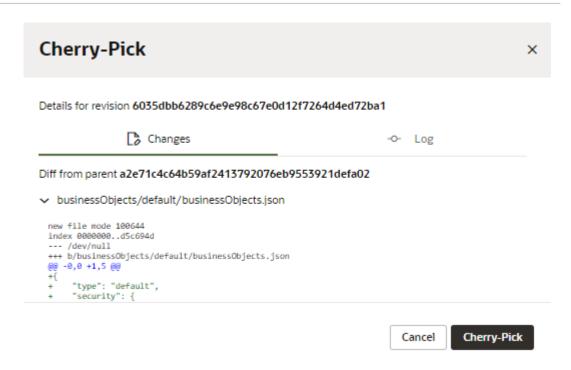
Once you have a commit's revision ID, you can cherry-pick it to your working branch. Make sure you select the correct branch to cherry-pick the commit.

A cherry-pick applies the changes from a commit to your working branch, but it does not commit the changes automatically. You'll need to review the cherry-picked changes and explicitly commit them to your branch. This new commit will have a different revision ID, even if its changes are just those from the original cherry-picked commit.

To cherry-pick a commit to your working branch:

- 1. Click the Git menu in the header or in the Git panel, and select Cherry-Pick.
- 2. In the Cherry-Pick dialog, enter the ID of the revision you want to cherry-pick and click **Fetch Revision** ().

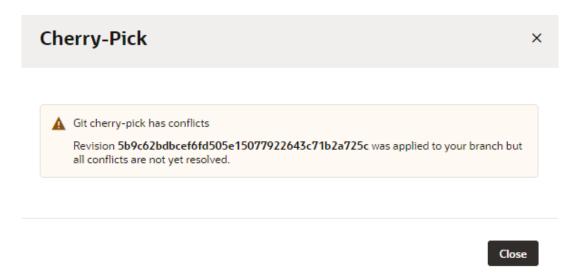
Use the **Changes** and **Log** tabs to view the changes associated with the commit you're cherry-picking:



If your branch has uncommitted changes, you'll be prompted to commit your changes.

 Click Cherry-Pick. Alternatively, click Commit All and Cherry-Pick to first commit your unsaved changes, then do the cherry-pick. (You'll still need to explicitly commit the cherrypicked changes afterward.)

If you run into conflicts (say, because a file has been modified on two different branches), you'll see a message similar to this image:



To resolve the conflict, click **Close**, then go to the **Git Panel** to resolve the issue in the conflict editor.

4. Once the commit is successfully applied (and any conflicts resolved), commit the cherry-picked changes to your branch, either from the Git Panel or the Commit option in the header's Git menu.

Create or Switch a Branch

When you create an extension with a Git repository (instead of a scratch repo), VB Studio creates two branches in the repo: the default branch (usually, main) and a working branch.

The main branch contains the initial set of source files for the extension. The working branch contains the same set of source files initially, but as you work on your extension, this branch will contain the changes that you make. Using the **Switch Branch/Sandbox** option, you can commit your changes and then switch branches, or create a new branch from a parent branch that you select.

Note:

You don't have to switch to another workspace to configure another page in your App UI, as long as the target for your change is in your current Git repository. It's much faster to switch branches within a workspace than to switch workspaces. For example, suppose you're working in an extension (that is, a Git repo) for the Sales App UI. Let's say you're configuring the Accounts page, but then get an urgent request to fix something on the Opportunity page and deploy it to your production instance. Rather than create a new workspace, you simply create a new branch, make the fix and publish it, then switch back to the Accounts page code line and resume your work.

You use the **Switch Branch/Sandbox** dialog to switch branches or create a new branch from a parent branch that you select, or, optionally, switch from one sandbox to another.

1. Click the Git menu in the header or in the Git panel.

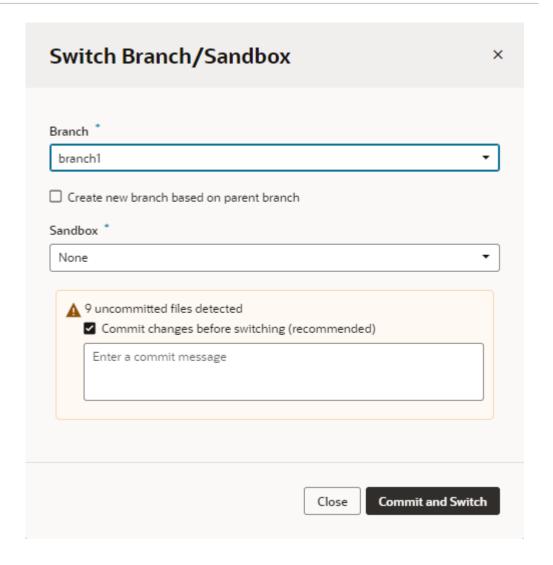
You can also click the Sandbox menu in the header.

2. Select Switch Branch/Sandbox.

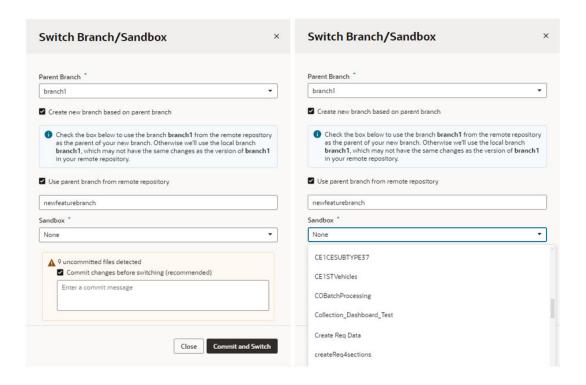
The Switch Branch/Sandbox dialog displays the current branch in the Branch drop-down list and the number of uncommitted files, if any.

- 3. Choose the appropriate option:
 - To switch branches, select the branch that you want to switch to.





- To create a new branch, select the Create new branch based on parent branch checkbox, enter the name for the new branch.
 If the selected parent branch exists locally as well as in the remote repository, select Use parent branch from remote repository to use the remote branch as your new branch's parent. You'd typically want to do this if the remote branch is more current than the local branch. Otherwise, the local branch will be used.
- To associate this branch with a new sandbox, choose one from the **Sandbox** dropdown. Some sandboxes might be marked "Unpublishable", which means they either have dependencies that haven't been resolved or were created without the ability to publish. Avoid choosing an unpublishable sandbox, as doing so is likely to break your extension.



 Click Commit and Switch (or Switch if you cleared the Commit changes before switching (recommended) checkbox).

If you chose to create a branch, take note that the newly created branch only exists locally —until it is pushed to the remote repository. To do this, you can either push the branch to the remote repo or merge it into the main branch in the remote repo.

Rename a Branch

When you want to rename a Git branch, you can use the **Rename Branch** option to change the name of your local branch, then push the renamed branch to the remote repository.

Note:

Exercise caution when renaming branches, especially if other team members are still using those branches. Renaming the default branch (main, for example) might break build scripts that your repository uses. Before you rename the branch, you'll want to update references to the old branch name in your code or scripts.

Renaming a branch changes your current branch's name while keeping its history and pushes the renamed branch to remote, but the current remote branch is not renamed or deleted. Suppose your current branch is mybranch and you want to change its name to newbranch, the rename first replaces the local mybranch with newbranch, then pushes the local newbranch to the remote repository. No changes are made to mybranch, the previous existing remote branch. You'll have to manually delete this remote branch if you want to.

If the branch you want to rename is local (that is, it isn't tracking a remote branch), the branch is renamed, but isn't automatically pushed to the remote repository—the branch stays local even after the rename.

To rename a branch:

- 1. Click the Git menu in the header or in the Git panel, and select **Rename Branch**.
- 2. In the Rename Branch dialog, enter a new name for the branch. Make sure you're not using the name of an existing branch.
- 3. If the current branch has any uncommitted changes, you have the option of committing those changes.
 - To commit your changes before renaming the branch, enter a commit message (with the Commit changes before renaming branch checkbox selected).
 - To rename the branch without committing your changes, deselect the Commit changes before renaming branch checkbox.
- 4. If you chose to commit changes, click **Commit All and Rename**; otherwise, click **Rename Branch**.

When the rename is successful, close the confirmation message that appears.

The renamed branch shows up in the header as your current branch.

Delete a Local Branch

You can delete a local Git branch that you no longer need to keep your workspace free of clutter. Typically, after your work on a branch is done and you've pushed your changes to a remote repository, it's good housekeeping to delete your local branch.

If your work required you to create a merge request, you can delete the branch immediately following the merge. For all other scenarios, you can use the **Delete Local Branch** option in the Git menu.

If you delete a branch that hasn't been pushed, its unique commits are likely to be lost. Make sure you push any commits you want to keep.

To delete a local branch:

- 1. Click the Git menu in the header or in the Git panel, and select **Delete Local Branch**.
- 2. In the Delete Local Branch dialog, select the local branch you want to delete.
 - A list of local branches that you can delete are shown. This list will not include your current working branch. If you want to delete the current working branch, you'll need to switch to another branch to make your original current working branch available for deletion.
- 3. Select Yes, I want to delete the branch to confirm your action.
- Click Delete Branch.

When the delete is successful, close the confirmation message that appears.

Merge Remote Branches

Use the Merge Requests tool in VB Studio to merge remote branches, such as when you want to merge a remote branch with your changes into the main branch. Your team members can use the tool to review your changes, leave comments, and to approve or reject your merge request. If your request is rejected, you'll need to fix any problems and create a new merge request.

A merge request is created automatically when you use the Merge After Review option in the Publish Changes dialog box in the Designer. To publish the extension, you'll need to open the Merge Requests page in VB Studio and merge the branch into the main branch.

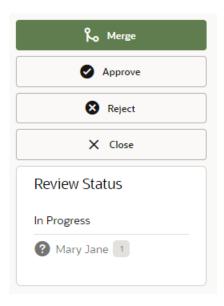


You can also create merge requests in the Merge Requests page. You'll need to create a merge request when you want to merge two remote branches. For details on how to create and work with merge requests, see Review Source Code with Merge Requests in *Using Visual Builder Studio*.

It's good practice to ask some project members to review your changes, but it isn't mandatory to get approval from all reviewers before you merge the review branch. Note that you can't merge the review branch if the target branch is locked. If it's locked you need to contact the project owner to unlock the target branch. For details on how to merge branches, see Merge Branches and Close the Merge Request in *Using Visual Builder Studio*.

To merge two remote branches:

- 1. Open Merge Requests in the main menu in VB Studio and open the merge request.
- On the right side of the page, click Merge.



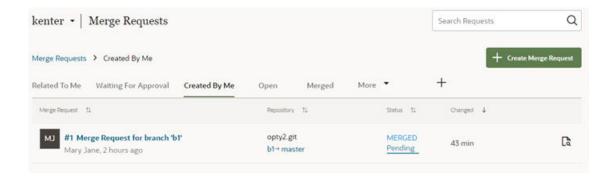
In the Merge dialog box, type a description of the merge and click Create a Merge Commit.

If changes in your sandbox are not yet published, it's recommended that you publish the sandbox before merging your application extension changes to avoid potential problems.

To delete the review branch after the commits are merged with the target branch, you can select Delete Branch in the dialog box.

When your merge is complete you'll see a summary of the merge request in the Conversation tab. If you didn't select Delete Branch in the Merge dialog box, you can delete it now by clicking **Delete Branch**. After merging a branch to the main branch, the branch is automatically closed and you can no longer use it. To make additional changes you'll need to create a new branch in your workspace or open an existing branch.

In the main Merge Requests page, you can see the status of merge requests in the table, and locate requests using the built-in filters, for example, Created By Me.



Push a Scratch Repository to a Remote Repository

If you use a scratch repository for your extension, your teammates cannot work with your changes until you create a remote Git repository. To share your scratch repository with your team, you'll need to push its content to a remote repository. A remote repository is also required to fully use Git operations within the Designer.

- 1. Click the Git menu in the header or in the Git panel, and select **Push**.
- 2. In the Push Scratch Repository to Remote dialog, enter a name (and optionally, a description) for the Git repository that VB Studio will create for you.
- 3. By default, VB Studio creates packaging and build jobs and adds them to a CI/CD pipeline, allowing you to build and publish your changes when you click Publish. A CI/CD pipeline gives you the option to publish changes to more than one environment and to perform some lifecycle operations. If you don't need these capabilities, click CI/CD Pipeline to toggle off the setting and not create these jobs and pipelines. Your changes will then be merged directly to your Git repository's main branch and deployed immediately to your Development environment when you click Publish.



The CI/CD pipeline is an extension-level setting that impacts everyone collaborating on the extension, including those using it in Express mode. Before you enable or disable the pipeline, make sure it's the optimal configuration for all those who work on the extension. Once you decide on a setting, it's a good idea not to change it unless your requirements change significantly.

If you don't see the **CI/CD Pipeline** option, it means an environment isn't associated with your workspace. To add an environment with a suitable deployment target, see Set Up an Environment.

- Enter a commit message.
- 5. Click Push Repository.

Using Branches with a Sandbox

Within your workspace, you can use separate branches for changes you want to make for two different sandboxes, or for different versions of a feature that use the same sandbox.

For example, when modifying a table component in your extension, you might want to work on two different versions of the table. By creating a branch for each version, you can work on one

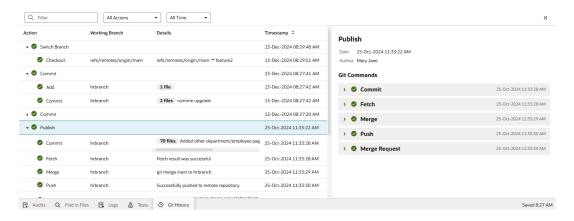
version in one branch with Workspace A mapped to it, and then switch to another branch by using Workspace B to work on the other version. This way you can use your workspaces to help you isolate the branches with your changes. After you decide which version you want to use, you can share the branch with others and delete the branches you no longer need. See How Do Sandboxes Relate to Git Branches? .

View Git History

As you manage changes in your workspace, you can use the Git History panel to view your Git actions and keep track of what you've done in the workspace.

The Git History panel lists all your Git actions (the last 400 to be precise), along with the results of each action. Accessing this panel is useful when you want to check the sequence of recent actions and their details, especially when troubleshooting issues with version control. You can also filter the actions by various criteria (commit message, revision ID, branch name, action type, and so on) to quickly find events and check details to understand its history.

- 1. Click **Git History** at the bottom of your workspace.
 - A panel opens to show the Git actions you've performed in the workspace (starting with the most recent one).
- 2. Click any action to view its details on the right. Say you're looking for the revision ID of a commit; click the commit and look for the revision ID in the details pane. If the action you select has other actions triggered internally (for example, a Publish action that triggers other actions), you'll see a sub-list of those actions, as shown here:

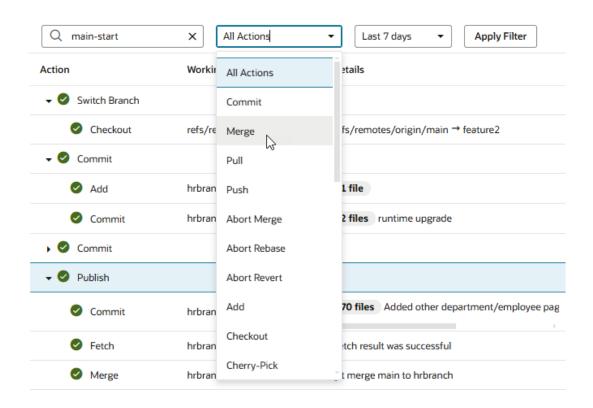


In the details pane, you can click each sub-action to view more information.

3. When you know what you're looking for, use the search and filter options to find it quickly.

In the Filter search box, enter text (say, a file name, branch, or message) to find the item that matches what you enter. Case is important as text-based search is case-sensitive. You can also filter by action type or by date. Use a combination of all three options to quickly find what you're looking for:





How Do Sandboxes Relate to Git Branches?

There is no strict association between an Oracle Cloud Applications sandbox and a particular Git branch in your local repository. In fact, you can associate any sandbox to any Git branch in your repo while you're working on your extension in your private workspace.

As described in How Do I Use My Sandbox in Visual Builder Studio?, a sandbox hosts unpublished changes made to an Oracle Cloud Applications data model. Once you or another team member publish the sandbox, the changes are committed to the data model, so the sandbox is no longer needed and no longer available. If you want to refer to unpublished data in your extension, you simply associate your current Git branch with that sandbox. The typical workflow looks like this:

- In your Oracle Cloud Application, you create a sandbox to make a change to the data model, like adding a new column to a database table.
- 2. When you create a workspace in VB Studio, you specify the sandbox you want to access while making UI changes in your extension.
 If you jump over to VB Studio using the Edit Page in Visual Builder Studio option, we automatically add the sandbox association to your workspace for you. When accessing a workspace via this option for the first time (something that is typical when you create an Application Extension project and are yet to open the workspace), your workspace will be set to your Oracle Cloud Application's active sandbox if one isn't associated with your workspace.
- 3. Also while creating your workspace, you can create a new Git branch to house the UI changes you plan to make. Or, if you've previously made UI changes on a branch and want to continue with that branch, you can select it here.

Switch a Sandbox

You might need to switch sandboxes in your workspace if the one you're using was deleted or published. You can also switch when you want a sandbox that defines an alternative data model, or if yours is using the mainline data model (indicated by the **None** sandbox in the header).

Before you switch a sandbox, see How Do I Use My Sandbox in Visual Builder Studio? for best practices on working with sandboxes.

1. Click the **Sandbox** menu in the header:

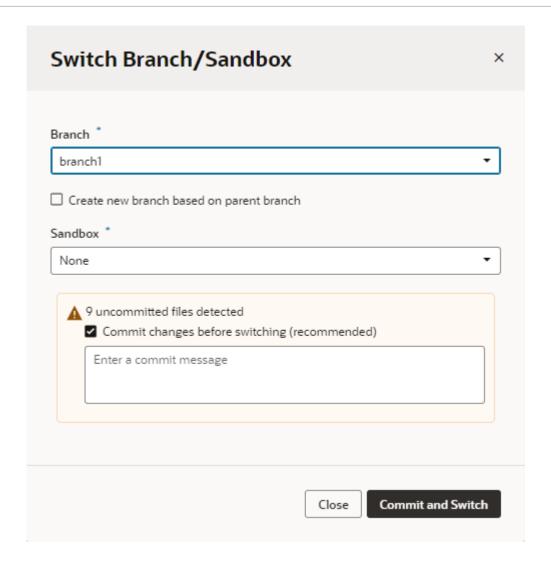


You can also click the Git menu, next to the name of the current branch, in the header.

2. Select Switch Branch/Sandbox.

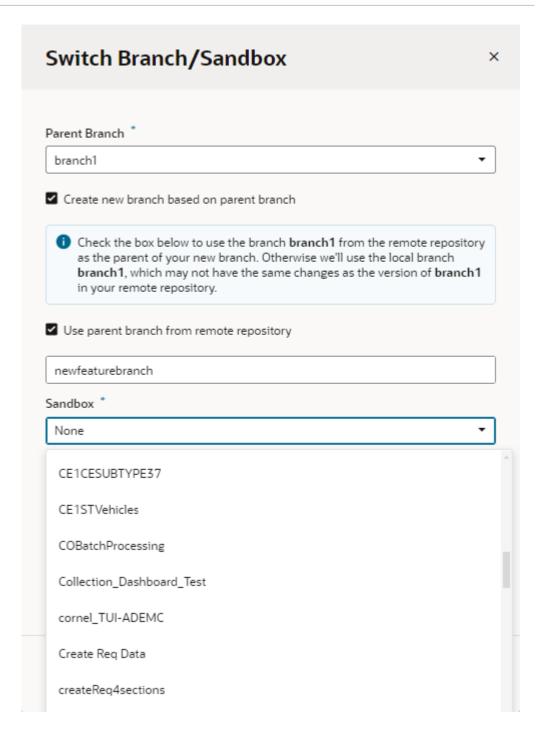
The Switch Branch/Sandbox dialog displays the current branch in the Branch drop-down list and the number of uncommitted files, if any.





3. Choose the appropriate option:

- To switch branches, select the branch that you want to switch to.
- To create a new branch, select **Create new branch based on parent branch** and enter the name for the new branch.
 - If the selected parent branch exists both locally as well as in the remote repository, select **Use parent branch from remote repository** to use the remote branch as your new branch's parent. You'd typically want to do this if the remote branch is more current than the local branch. Otherwise, the local branch will be used.
- To associate this branch with a new sandbox, choose from the Sandbox drop-down.
 Some sandboxes might be marked "Unpublishable", which means they either have dependencies that haven't been resolved or were created without the ability to publish. Avoid choosing an unpublishable sandbox, as doing so is likely to break your extension.

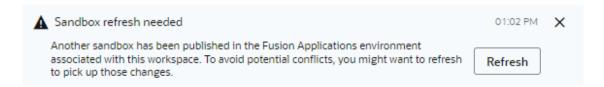


 Click Commit and Switch (or Switch if you cleared the Commit changes before switching (recommended) checkbox).

If you chose to create a branch, take note that the newly created branch only exists locally —until it is pushed to the remote repository. To do this, you can either push the branch to the remote repo or merge it into the main branch in the remote repo.

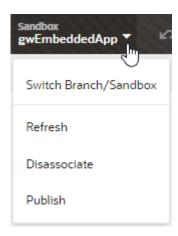
Refresh a Sandbox

You might need to refresh a sandbox associated with your workspace to pick up the latest changes from other published sandboxes. VB Studio will notify you if this is required, for example, as shown here:



Clicking **Refresh** in the message will refresh your sandbox. You can also do this using **Refresh** in the Sandbox menu:

Click the Sandbox menu in the header:

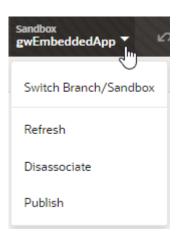


2. Select Refresh. This option is enabled only when your sandbox needs to be refreshed.

Disassociate a Sandbox

You can remove the sandbox currently associated with your workspace if you no longer need it. Removing a sandbox breaks the relationship between the Git branch and the sandbox and will likely impact your app's logic as well as the user interface. Proceed with caution.

1. Click the **Sandbox** menu in the header:



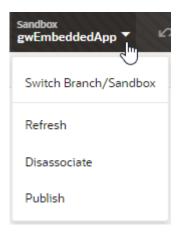
- Select Disassociate.
- 3. In the Disassociate Sandbox dialog, click **Disassociate** to confirm your action.

Once the sandbox is disassociated, your workspace is set to the **None** sandbox (which uses the mainline data model). Make sure you resolve any issues reported in the **Audits** pane.

Publish a Sandbox

After you're done making changes to your extension, publish the sandbox to make your changes available to all users. Remember that you can't make changes to a sandbox once it's published, so make sure you adequately test and verify your extension's code before publishing your changes.

1. Click the **Sandbox** menu in the header:



- 2. Select Publish.
- 3. In the Publish sandbox dialog, click **Publish** to confirm your action.

Once the sandbox is published, your workspace is set to the **None** sandbox (which uses the mainline data model, now with your changes).

Any workspaces associated with the published sandbox must be switched to another sandbox.

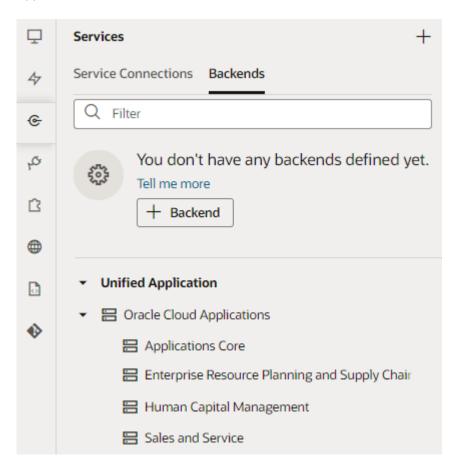
Work With Services

Many of the UI components in your App UI 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 extension 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 extension level, which means that all the App UIs in your extension can use them. In addition, you can use service connections provided through a dependent extension as a source of data for other App UIs in your extension, whether you're creating the App UI from scratch or configuring an existing App UI.

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. The Unified Application provides a backend called "Oracle Cloud Applications for all services", which points to the servers hosting the Application Core API, the Sales and Services API, and more:



While you can't modify the service connections or backends that come from dependent extensions (including the Unified Application), you can create your own and reuse them across all the App UIs in your extension.

Important: When creating your own service connection or backend, you must mark it as "Accessible to application extensions" if you want to allow their use by dependent extensions.

Manage Service Connections

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

What Are Service Connections?

A *service connection* provides basic information about a service, including connection details, properties, and the REST endpoints provided by the service.

You can create a service connection by:

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

Each of these mechanisms create a representation (metadata) of the external REST service, which is required for you to use the REST endpoints as data sources in your extension.

Let's take a look at how you might use service connections in your extension:

- Let's say you're configuring an Oracle-provided App UI, which has a dynamic container, and you want some of the content in the container to be sourced from a new custom object you defined in Oracle Cloud Applications. You create a service connection in your current extension, choose the Catalog option, select the relevant Oracle Cloud Application pillar, then select the custom object REST endpoint you want to use. Finally, after you've created and tested the service connection, you use it to source the content in your dynamic container.
- Suppose you're creating a brand new App UI which needs to show population data from a
 publicly-available countries API. In the extension containing your new App UI, you'd create
 a backend (with no authentication), as well as a service connection to the countries API.
 You'd then configure the UI elements on the App UI's page to show data from this service
 connection.
- Suppose you want to display data from a third-party system, like LinkedIn, in your App UI. To achieve this, you'd first create a backend to represent LinkedIn, using a *local server* configuration that uses the OAuth 2.0 Client Credentials for the authentication method. Next, you'd create a service connection to the LinkedIn REST API, using the "Define an Endpoint" option. Finally, for each Oracle Cloud Application instance to which you plan to deploy your extension, you'd need to configure the Manage Backend for Visual Builder Studio page, as described in Connect to OCI Process Automation Services.
- Suppose you want to create a companion application (that is, an App UI) for your
 HelpDesk app, which relies on a custom object you created in Application Composer. You
 could simply grab the URL for the REST Describe in Application Composer, then create a
 new service connection using the **Define by Specification** option in VB Studio. See
 Display SaaS Data In Your App UI for more details.
- Instead of creating a new service connection, you might add an extension containing the service connection you want to use as a *dependency*, assuming that the "accessible" flag



has been set. You could also use any of the service connections or backends defined in the Unified Application.

When you define a service connection, you supply metadata for it that can include request and response payload structures in JSON, query and path parameters, headers, and so on. This metadata helps to shape the raw data that you get back from the REST call, to narrow the scope of the data that is returned.

When you create a new service connection, you must also create a new *backend*, unless you use an existing backend as part of the service connection. Backends provide a way to bundle common attributes across service connections into a single entity you can refer to later. For example, you could create a backend (perhaps called "myBackend") with the URL and the authentication information, then create two service connections based on "myBackend" for / employee and /department resources.

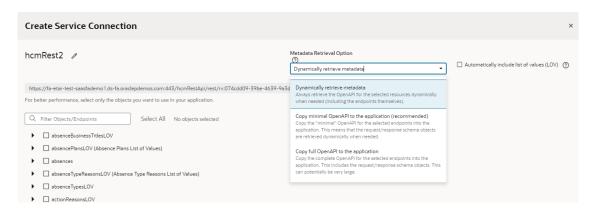
Service Connections: Static Versus Dynamic

Service connections in VB Studio 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 VB Studio.

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 App UI, because the



service metadata is fetched every time the App UI 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 <code>openapi3.json</code> 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.
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) Available only for ADF Describe-based services like Oracle Cloud Applications	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

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 Manage Service Endpoints.

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, VB Studio 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:

```
de
Sales and Service vb-catalog://backends/base:crmRest
                          .oraclecorp.com:443/crmRestApi/rest/rv:8053295d-5425-47da-
https://
9d2c-3d59d603c2fe/en
Applications Core vb-catalog://backends/base:fndRestApi
                          .oraclecorp.com:443/fscmRestApi/applcoreApi
Enterprise Resource Planning and Supply Chain vb-catalog://backends/base:fscmRest
                          .oraclecorp.com:443/fscmRestApi/rest/rv:8053295d-5425-47da-
https://
9d2c-3d59d603c2fe/en
Default Server vb-catalog://backends/Extension:hcmIndexSearch
                          .oraclecorp.com:443/hcmRestApi/indexSearch
Human Capital Management vb-catalog://backends/base:hcmRest
https://
                          oraclecorp.com:443/hcmRestApi/rest/rv:8053295d-5425-47da-
```

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.





All URLs in backends and service connections should be HTTPS with a valid certificate.

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.

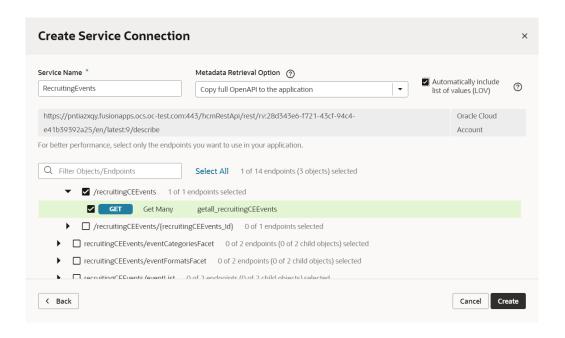
To create service connections from one of these catalogs:

- Click Services in the Navigator.
- 2. In the Services pane, click the + sign and select Service Connection.
- 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 dropdown list, then select the endpoints that you require for your application. This way, you create a static service connection:



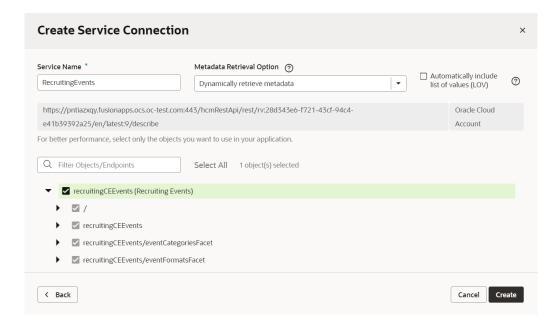


\bigcirc

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:







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.

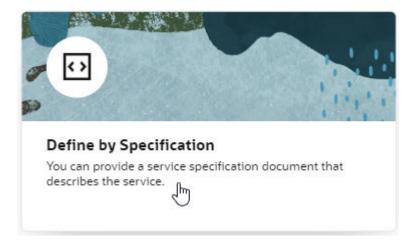
- 6. Optional: If you want all list of values (LOV) for the selected objects/endpoints in an Oracle Cloud Applications catalog to be automatically included in the service metadata, select Automatically include list of values (LOV). Take note that selecting this option may increase the size of the openapi3.json file that holds service metadata and potentially impact performance.
- 7. Click Create.

Create a Service Connection from a Service Specification

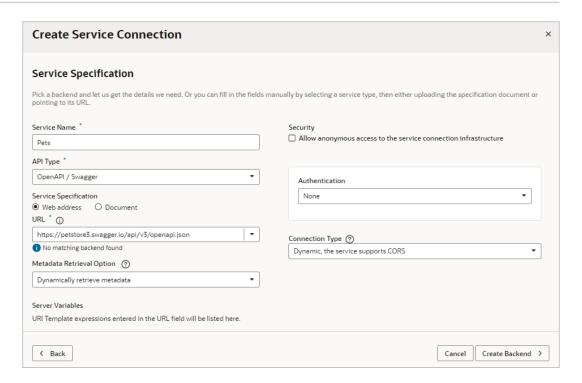
You create a connection from a service specification when you know the URL of the OpenAPI/ Swagger or Oracle ADF Describe file that describes the service. You can also upload the specification file from your local system.

To create a connection from a service specification:

- 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:

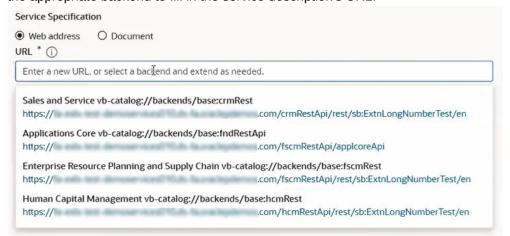


4. Enter a name for the connection in the **Service Name** field.



The name you specify will be the connection's display name in your application.

- Click the API Type dropdown and select the appropriate 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 (cache-enabled) if you need to create a service connection to an Oracle Cloud Applications REST API that has a Describe URL.
 - Select ADF Describe for any other REST API that has an ADF Describe file.
- 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>:<REST-Framework-version>/
describe.openapi/<resource-name>

For example, for the HCM absences resource, you'd add /11.13.18.05:9/ describe.openapi/absences to the HCM URL you selected in the text box, where 11.13.18.05 is the REST API version and 9 is the REST Framework 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>:<REST-Framework-version>/
describe?resources=contracts,expenses

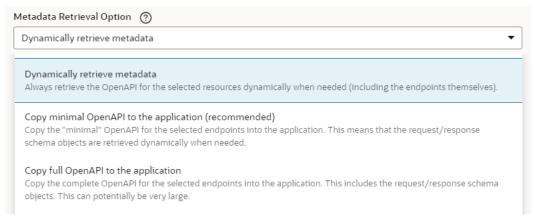
- If you have an external URL representing the service specification (for example, an OpenAPI3 URL), select Web address and enter the service descriptor's URL in the URL field.
 - If you're using an IP address instead of a proper DNS-based URL in a production environment, you're probably using self-signed 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**, 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.

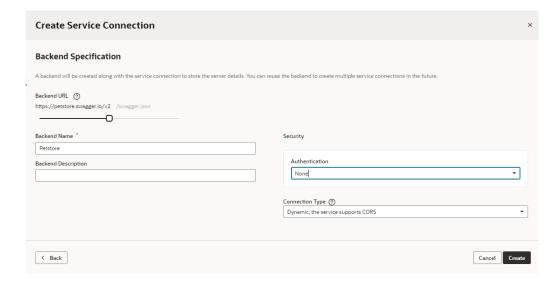
7. Select a Metadata Retrieval Option.



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 or ADF Describe (cache-enabled), 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:
 - Update the Security and Connection Type settings as needed.
 - b. Click Create Backend.
 - c. If you are creating a backend that uses credentials (like Basic Authentication), or one that has different URLs for your dev/test/production instances, check the Local Server Definition checkbox. If your backend doesn't need authentication, leave Local Server Definition unchecked. You can see an example of how to work with different authentication methods here: Connect to OCI Process Automation Services.
 - d. In Backend URL, use the slider to specify which part of the URL you want to use for the backend.

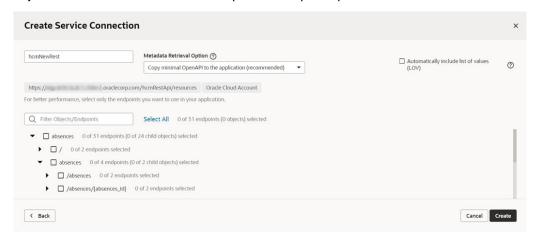




- e. Add a name and optional description for the backend.
- f. Update the Security and Connection Type settings for the backend as needed.
- 9. Do one of the following:
 - If you have an ADF Describe or ADF Describe (cache-enabled) API type, continue to the next step.
 - If you have an OpenAPI/Swagger 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.
- Go to the Test tab and, under Response, enter an integer for a query parameter.
- 3. Click Send Request.
- Check the Send Request status. A status of 200 indicates success.

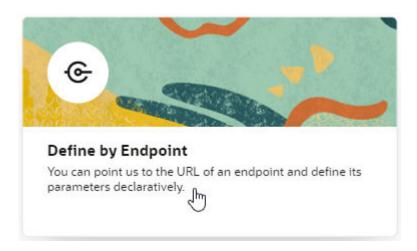
Create a Service Connection from an Endpoint

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:

- Click Services in the Navigator.
- 2. In the Services pane, click the + sign and select Service Connection.
- Click Define by Endpoint in the Select Source step of the Create Service Connection wizard.



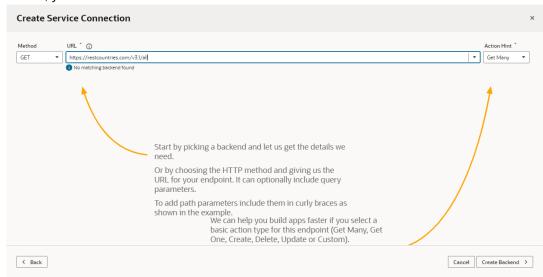


Select the HTTP method and enter the endpoint URL.



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.

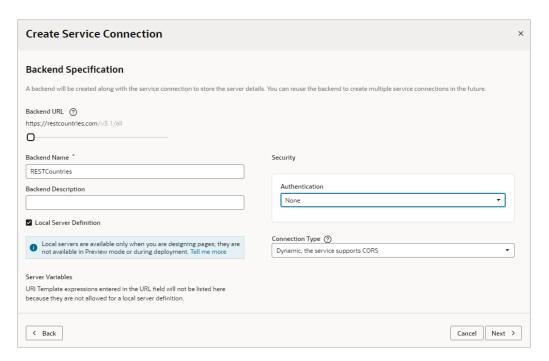
If you know the result expected from the endpoint URL, select it in the **Action Hint** drop-down list to indicate what the endpoint does. For example, when you select **GET** as the **Method**, select **Get One** as the Action Hint if the endpoint URL will retrieve a single record, or **Get Many** if 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.

- If you did not have a backend to match your service specification URL, click Create Backend.
 - If you are creating a backend that uses credentials (like Basic Authentication), or one that has different URLs for your dev/test/production instances, check the Local Server **Definition** checkbox. If your backend doesn't need authentication, leave **Local Server Definition** unchecked. You can see an example of how to use different authentication methods here: Connect to OCI Process Automation Services.
 - In Backend URL, use the slider to specify which part of the URL you want to use for the backend.



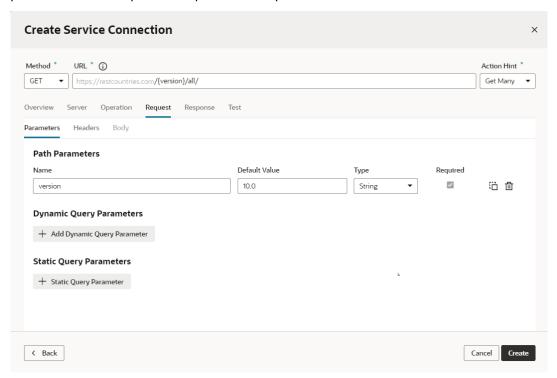
- Add a name and optional description for the backend.
- (Optional) Change the **Security** and **Connection Type** settings for the backend.
- Click Next.
- In the **Overview** tab, add a Service name, and confirm the title and version. Optionally, enter a description.
- Click the **Server** tab. If 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:
 - 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 or Always use proxy, irrespective of CORS



support. If there's a typo or a mismatch in the service's CORS configuration, you're likely to run into an error.

- 8. Click the **Operation** tab to view the Endpoint ID that VB Studio will use to identify the REST API endpoint you specified at the start of this task.
- 9. Click the **Request** tab to add headers and URL parameters to the request.

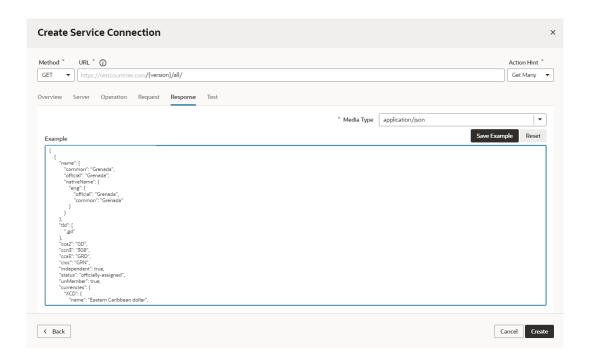
Depending on the endpoint, you might want to add custom headers or path or query parameters that are passed as part of the request.



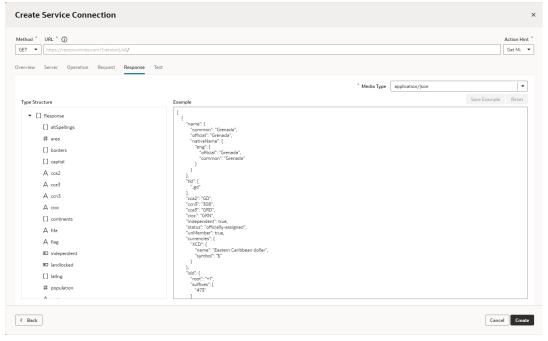
10. Click the **Response** tab and enter the response body for the endpoint.

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.



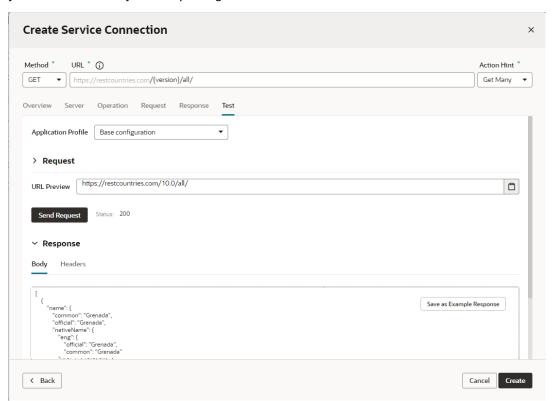
If you don't have an example to add, you can use the **Test** tab to send a request to the service, then save and use the response that is returned as your example (or edit it as needed) in the tab's text area.

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.
- 11. 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.

12. 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.
	 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.

Manage Service Endpoints

You may want to add new endpoints to your static or dynamic service connections or modify the endpoints you already have.

See:

- Add Endpoints to a Static Service Connection
- Add Endpoints to a Dynamic Service Connection
- Edit a Static Service Connection's Endpoints

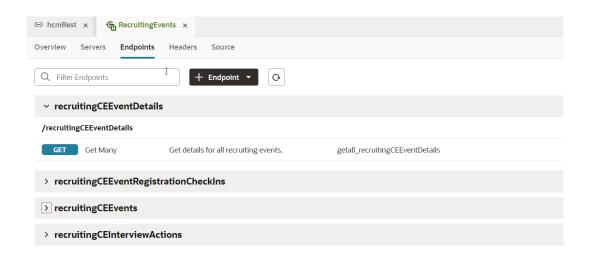
Add Endpoints to a Static Service Connection

After creating a static service connection, you can add endpoints from the same source or add custom endpoints.

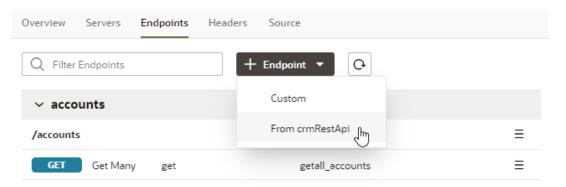
To add endpoints to a static service connection:

1. Open the static service connection's **Endpoints** tab:



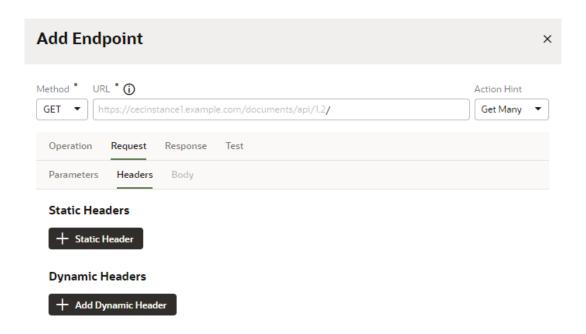


2. Click + Endpoint. If your service connection was created for a service from the catalog, select Custom or From original_service in the drop-down list:



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:



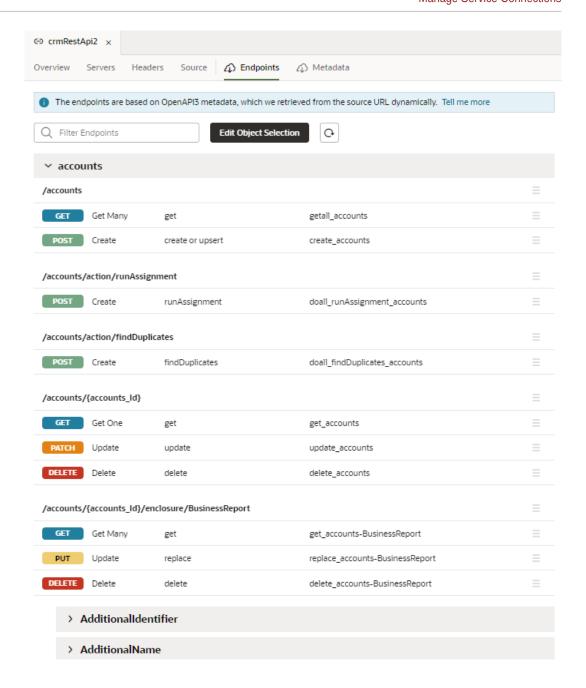
- 4. (Optional) If you chose to add an endpoint from the original service, you can change how your connection's service metadata is retrieved in the Metadata Retrieval Option dropdown. Select Dynamically retrieve metadata to convert your service connection to a dynamic one. For ADF Describe-based services, select Copy minimal OpenAPI to the application (recommended) to leverage both static and dynamic capabilities for your connection.
- 5. Click Save.

Add Endpoints to a Dynamic Service Connection

To add endpoints to a dynamic service connection:

1. Open the dynamic service connection's **Endpoints** tab:





- 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.
- Click Add.

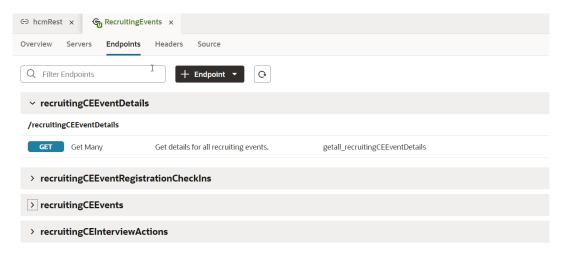
Edit a Static Service Connection's Endpoints

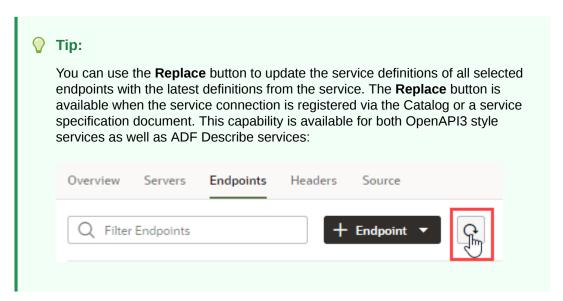
After you create a service connection and select the service endpoints, you can edit the endpoint settings, request parameters, and the response for each endpoint in the **Endpoints** tab. Endpoints can be edited only for static service connections (identified by the icon that appears in front of the service name).

If you edit an endpoint after you have created a type from it, you'll need to manually edit the type to use any of the changes to the endpoint. A type created from an endpoint isn't updated automatically when the endpoint is modified.

To edit a service endpoint:

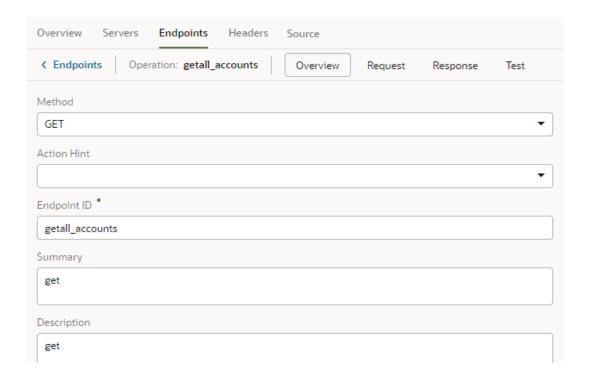
1. Open the service connection's **Endpoints** tab.





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 <code>example</code> or <code>examples</code> keys, the examples will show up within the **Request** and **Response** tabs.

The examples key takes precedence over example and, if multiple examples exist, just 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.

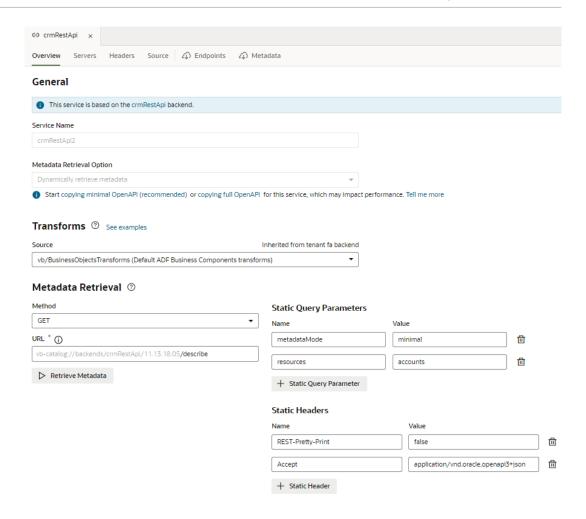
Retrieve Service Metadata for a Dynamic Service Connection

After creating a dynamic service connection, you can retrieve the service's most recent metadata, ensuring that your extension is using 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 for retrieving 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/base:crmRest/11.13.18.05:{REST-Framework-Version}/describe.openapi Server.

2. Click Retrieve Metadata.

The **Endpoints** tab displays a read-only view of the endpoints for the service connection's objects dynamically retrieved from the service's OpenAPI3 document corresponding to the source URL.

3. Click the **Metadata** tab.

The metadata file's contents, including endpoints and type definitions, are displayed.

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.



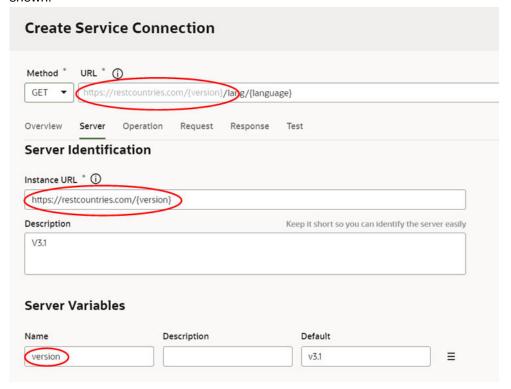
For service connections created after the 23.10 release, server variables must be updated from the backend associated with the service connection. For more information, see Add Server Variables for Backends.

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.
- 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 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

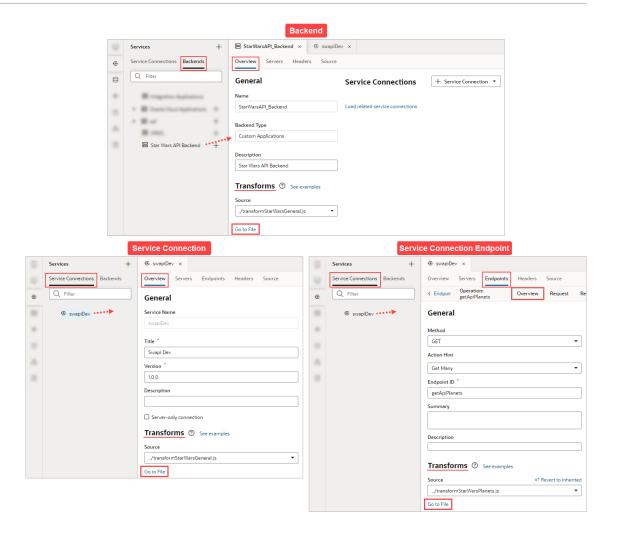
Service Level Transforms

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**.

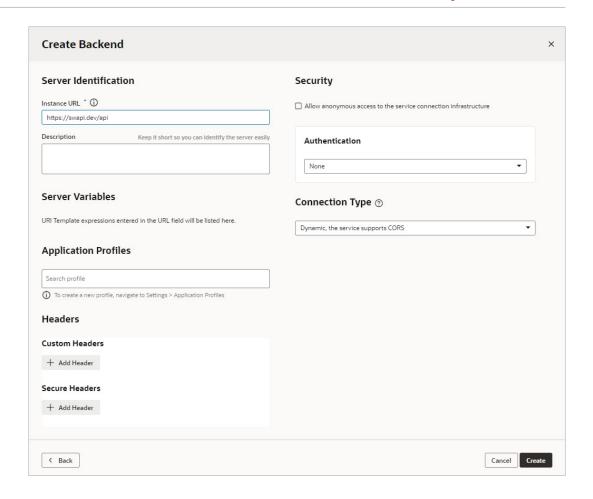




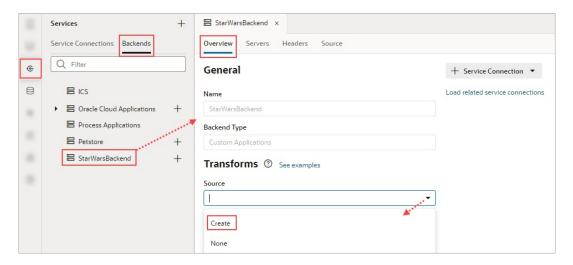
Example of Creating Service Level Transforms

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:

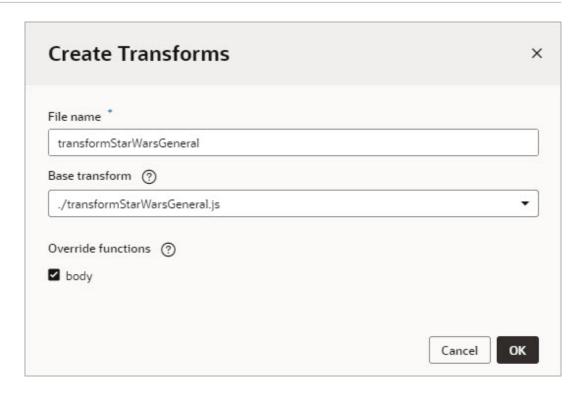
- Open the Services panel, select the Backends tab, click the Add (+) icon and select Backend.
- 2. In the Create Backend window, select **Custom** to create a custom backend.
- 3. Provide a name for your custom backend, and optionally, enter its description.
- 4. Enter the instance URL and any other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



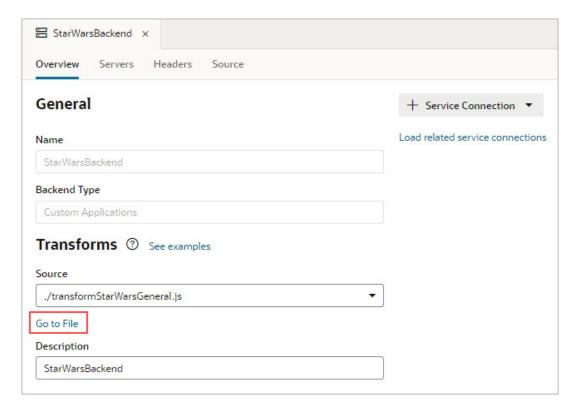
5. You can now create the file with the template transform functions for your new backend. On the Overview tab, in the Transforms section, click the Source field's down-arrow and select Create:



6. Provide the file name for the transforms file:



- 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.

```
JS TransformStarWarsGeneral.j ×
          class Response {
236
               * @typedef {Object} PaginateResponse
237
               * @property {number} totalSize optional what the totalSize of the result is (the total count of the records in
238
239
               * the service endpoint).
               * @property {boolean} hasMore usually required, the paginate response transform function is relied upon to
240
               * inform the ServiceDataProvider when to stop requesting to fetch more data. Indicates whether there are more
               * @property {String} pagingState optional. This can be used to store any paging state specific to the paging
243
244
               * capability supported by the endpoint. This property can be used in the response paginate transform function
245
               ^{st} to set an additional paging state. This will then be passed as is to the request paginate transform function
               * for the next fetch call.
246
247
248
249
              * paginate is called with the response so this function can process it and return an object with
250
              * properties set.
251
252
               * @param {object} result
              * @param {TransformsContext} transformscontext
253
               * @return {PaginateResponse}
255
256
              /*paginate(result, transformscontext) {
257
258
                  const tr = {};
259
                  if (result && result.body) {
260
261
                      const cb = result.body:
                      if (cb.totalCount) {
262
                        tr.totalSize = cb.totalCount;
263
                      if (cb.totalCount >= 0) {
                          tr.hasMore = !!cb.hasMore;
266
                      } else {
267
268
                        tr.hasMore = false;
269
279
271
                  return tr:
272
273
274
               st body is called last, after all the other response transforms have been called. It is a hook for authors
275
276
               st to transform the response body or build an entirely new one.
277
               * @param {object} result
               * @param {TransformsContext} transformscontext
278
279
               * @return {object}
280
                                                      transform function for
281
              body(result) {
                                                      body of response
282
                  // TODO: Replace example code
                  let tr = {};
283
284
                  if (result.body) {
285
                    tr = result.body;
286
287
288
                  // as a example store some random aggregation data
                  tr.aggregation = { example: 4 };
290
291
                  return tr;
292
```

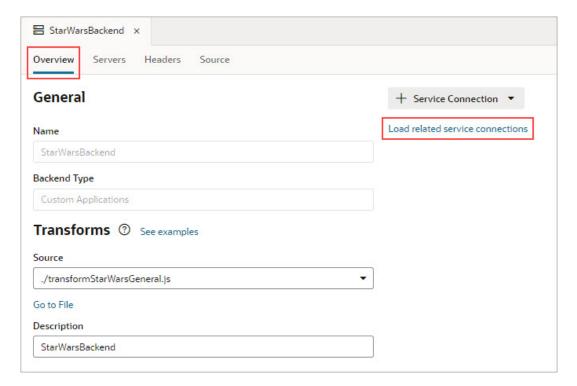
Here's an example for the <code>body(result)</code> transform function, which adds text to all of the <code>mass</code> and <code>height</code> fields:

```
body(result) {
```



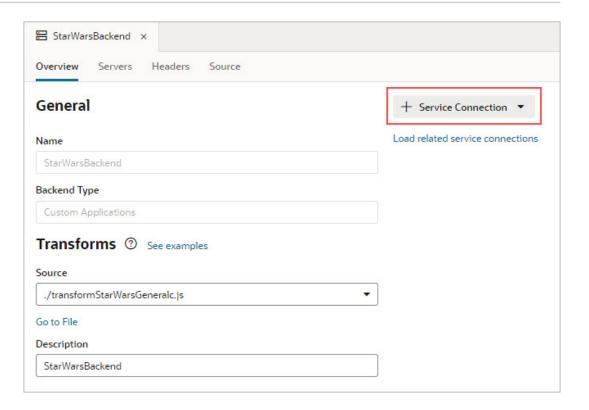
```
let tr = {};
if (result.body) {
    tr = result.body.results;
    for (let i = 0; i < tr.length; i++) {
        tr[i].mass = tr[i].mass + " kilos (kg)";
        tr[i].height = tr[i].height + " cm";
    }
}
// return tr;
}</pre>
```

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:

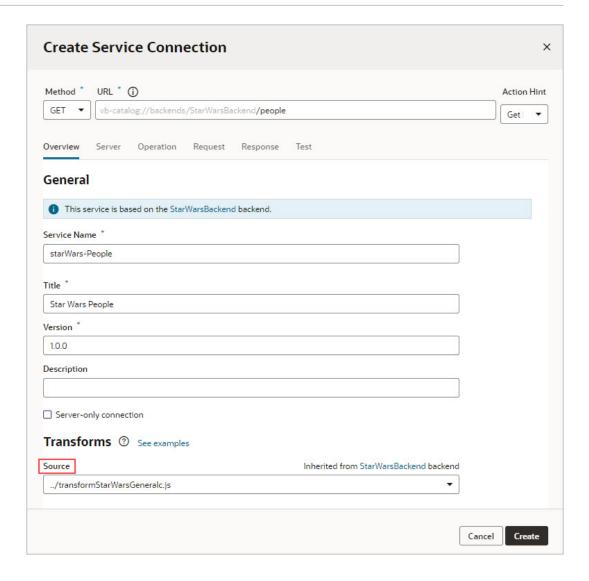


For details about creating a service connection from an endpoint, including how to get an example of the body of the response, see Create a Service Connection from an Endpoint.

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=${options.offset}`;
 c.url = newUrl;
 return c;
};
sort
 * Sort Transform Function Implementation
 * @param configuration
 ^{\star} @param options the JSON payload that defines the sortCriteria
 * @param context an object to store/retrieve any contextual information
 * current request lifecycle.
 * @returns {object} configuration object. the url looks like ?
orderBy=foo:asc
 * /
PageModule.prototype.sort = function (configuration, options, context) {
 const c = configuration;
  if (options && Array.isArray(options) && options.length > 0) {
    const firstItem = options[0];
    if (firstItem.name) {
     const dir = firstItem.direction === 'descending' ? 'desc' : 'asc'
     let newUrl = c.url;
     newUrl = `${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;
};
```

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.

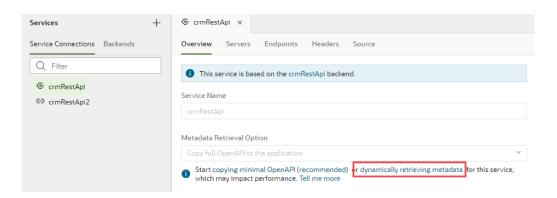
Typically, it's useful to create a dynamic service connection during an application's development stage, when endpoints defined in the service metadata are still evolving. A dynamic connection fetches the service's OpenAPI3 metadata from the source URL whenever the app is opened, enabling your application 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:

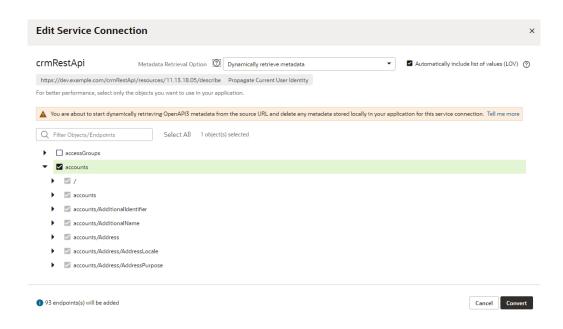


- To convert your service connection to dynamic:
 - Open the Overview tab of a static service connection, then click the dynamically retrieving metadata link.



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.

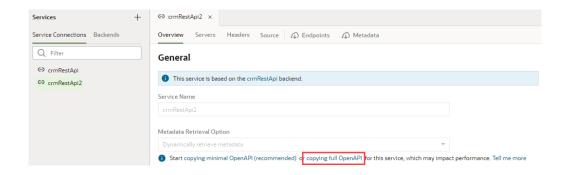
If you want all LOVs for the selected objects/endpoints in an Oracle Cloud Applications catalog to be automatically included in the service metadata, select **Automatically include list of values (LOV)**.



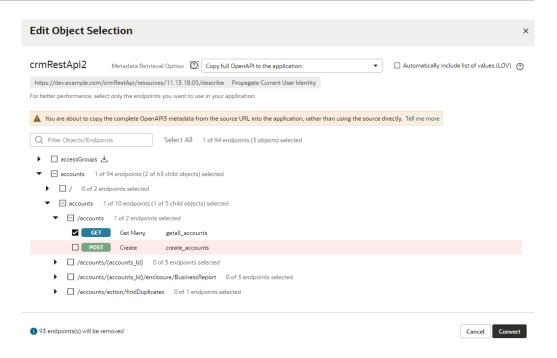
Click Convert.

The static service connection becomes dynamic.

- To convert your service connection to static:
 - Open the Overview tab of a dynamic service connection, then click the copying full OpenAPI link.



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.

Click Convert.

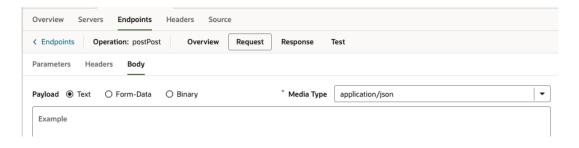
The dynamic service connection becomes static.

Test Service Connection Responses

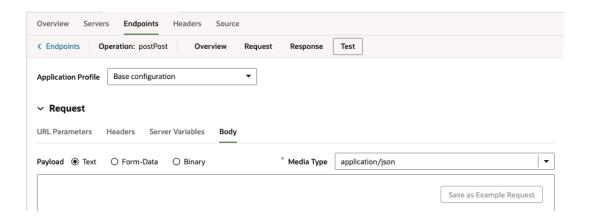
You can configure a POST/PATCH/PUT REST API with application/octet-stream or multipart/ form-data format and test it from the service connection's endpoint's Test tab before proceeding with it. You can also use the form-data's schema when you call the REST endpoint.

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:



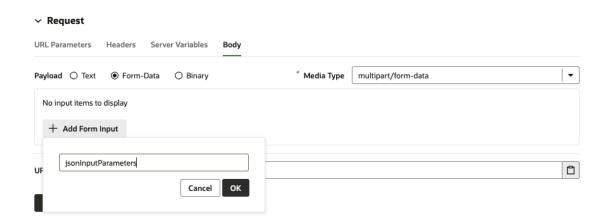
From the Body panel under the **Test** tab:



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:

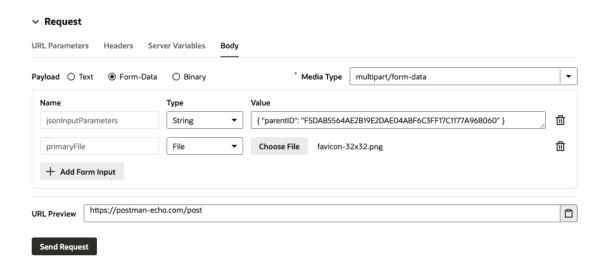


Two types of input elements can be created:

- String, for providing a user-entered text value.
- File, for selectiing a file to provide a binary data value.

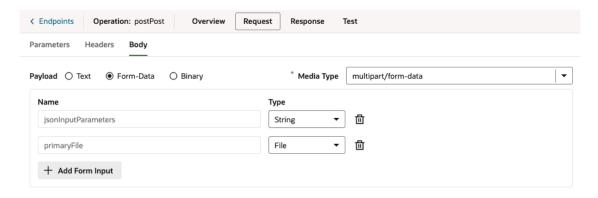
As elements are added to the form, corresponding properties will be added to the media type's schema in the OpenAPI metadata:





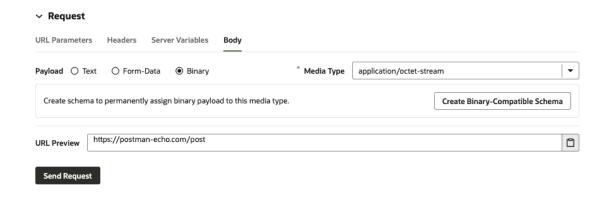
When the request is submitted from the **Test** tab, the names and values of the inputs are used to generate a FormData object that defines the multipart payload.

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

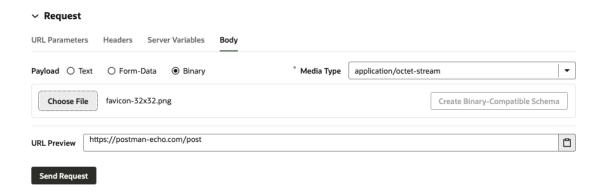
When you select the Binary payload option, the media type defaults to application/octetstream, although you could manually enter any binary content type you want:



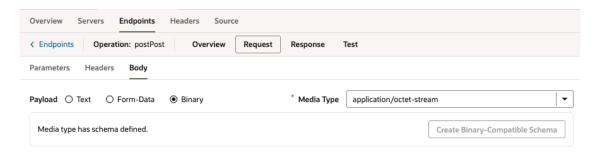


After you click the **Create Binary-Compatible Schema** button, the corresponding schema will be generated.

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:

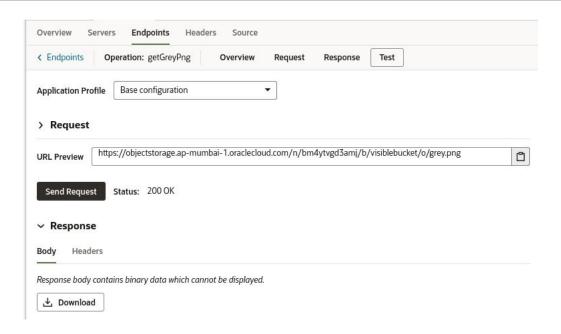


The file picker for the Binary payload type isn't shown in the Request tab:

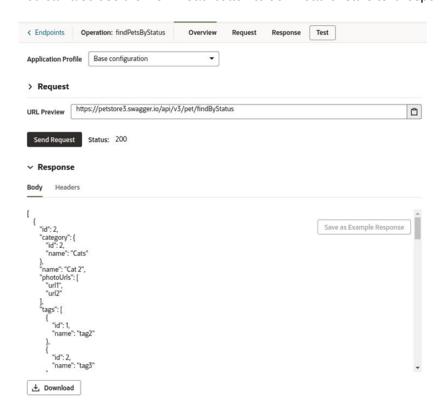


Instead, a message indicates that the schema has been created.

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:



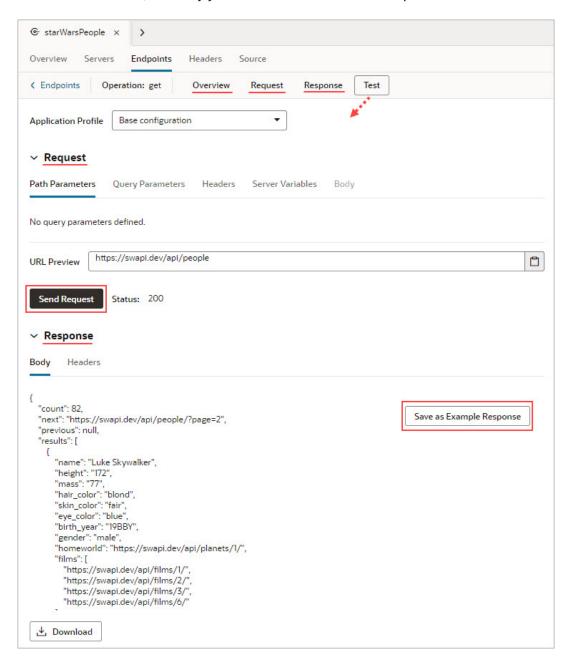
You can also use the **Download** button to download or save text responses:



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.





Manage Backends

Your extension has automatic access to some out-of-the-box backends, like Oracle Cloud Applications and its child backends. You can also create your own backends, as well as add another extension as a dependency and gain access to its backends.

What Are Backends?

Simply put, a backend describes the server that hosts a system you want to connect to. For example, the "Oracle Cloud Applications" backend represents the details of the Oracle Cloud Applications system, including the URL, authentication method, and so on, all of which are needed to gain access to that system's REST APIs.

All new service connections require a backend. If a backend doesn't exist, you'll be prompted to create one. You'll know when services are "derived" from a backend because you'll see something like this:



This service is based on the crmRestApi backend.

Click the backend link to open the Backends editor, where you can view the backend's details.

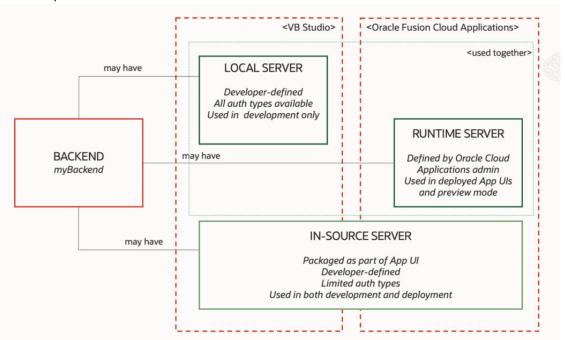
You can create your own backends to represent the systems you want to connect to. Creating backends for commonly used systems helps keep the information in one place, rather than duplicating it across various service connections. For example, instead of having multiple service connections for your company's LinkedIn account, each with the same URL, authentication method, and so on, you could define the details in one backend, perhaps called the "LinkedIn Backend".

A backend server can have two types of definitions, which reflect the URL and authentication method you want to use:

- An *in-source server* definition provides a backend's details for use during both development and deployment. Its details live in an extension's source code (specifically in the catalog.json file) and is therefore packaged up as part of the extension itself. This means that the backend always points to the same URL and authentication method, irrespective of where you deploy the extension.
- A *local server* definition provides a backend's details for use during development only. Because its details aren't stored in an extension's source code, you can go beyond the limitations of an in-source server by defining the URL and authentication method for the particular instance you're deploying to. Typically, a local server is used in conjunction with a runtime configuration, specified in the Oracle Cloud Applications instance, which provides the backend's details for the deployed extension (see Provide a Backend's Runtime Server Details). You can also use a local server with an in-source server to override its details during development, although this is far less common. In VB Studio, a local server's details are used by the Page Designer's Live view, while the server details provided by the runtime configuration and in-source server are used by the Preview action.

For example, suppose that during the development phase you want to use a mock URL. While testing you want to use a different URL, located in the test instance, and when you finally deploy you want to use the actual URL. You can achieve this using local servers, but not with in-source servers. The caveat is that local servers must be supplemented with a runtime configuration on the Oracle Cloud Application instance you are deploying to. The relationship

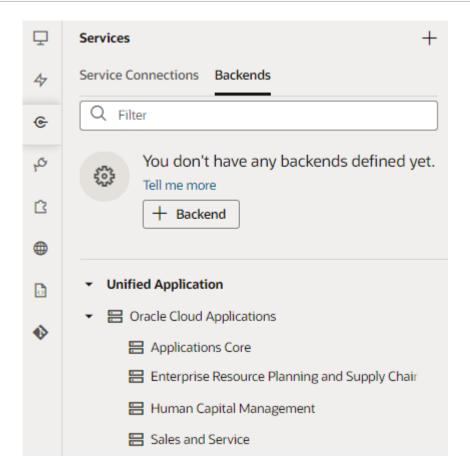




between potential servers is shown here:

In addition to defining your own backends, you can also add another extension as a *dependency* and gain access to its backends as well (so long as those extensions have been marked as accessible to other extensions). Your extension also has automatic access to some out-of-the-box backends, like Oracle Cloud Applications and its child backends. These backends are provided by the Unified Application, as shown here:





These backends are provided so that you can use them to quickly create service connections to the REST APIs exposed by Oracle Cloud Applications. You'll need these service connections so you'll have data for things like configuring a dynamic container, or adding other components to the page that require a data source.



If you don't see the Unified Application as a dependency, contact your administrator to confirm that you have the proper credentials and that your user role is authorized.

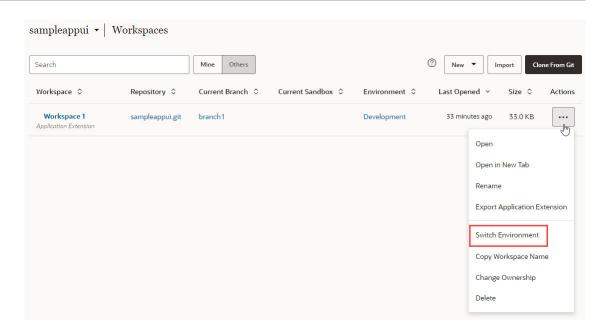
You can manage backends on the **Backends** tab, which you access from the **Services** tab in the Navigator. When you click a backend, you'll see four tabs on the right, each focusing on a different set of details:

Tab	Description	
Overview	Displays the name and type of the backend, which can be 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 the services connections that were defined from the backend. 	
	If you have the backend configured with a local server, you might also see a section that shows the runtime configuration on the Oracle Cloud Application instance you're deploying to. See What Are Local Servers? for more on this.	
Servers	Displays the server, including the instance URL, associated with the backend. You can display details for a backend server that comes from a dependency, but you can't edit it. You can edit the details for a backend server that you defined for your extension. The server can be either an in-source server, which is packaged within the extension, or a local server. Local servers are indicated by a Local server definition tag.	
Headers	Add and edit static headers, which pass information to the service, for the backend at the server level.	
Source	Displays the backend's description, which is stored in the <code>catalog.json</code> file of the extension or dependency that the backend is defined in. For backends that you define, the location is <code>services/self/catalog.json</code> , and the entries are editable. Local server details are not stored in the source code.	

The artifact that stores all the backend definitions for the extension is an OpenAPI-compliant file called *catalog.json*, which contains the list of backends and their details. If you want a backend to be usable by other extensions (that is, extensions who have added your extension as a dependency), the backend **must** be marked as "Accessible to application extensions" on the backend's **Overview** tab. The complete service catalog for your extension is a combination of all the backends that you have defined in your extension, plus any backends exposed by dependencies.

If your App UI needs access to a particular Oracle Cloud Applications instance—say, one that's running a different version of Human Capital Management—use your project's Environment's tab to add that instance to your Oracle Cloud Applications environment, then use **Switch Environment** to point your workspace to that environment (or just create a new workspace):





Create a Backend

By default, your extension has access to Oracle Cloud Applications and its child backends. You can't override or modify these, but you can use them to create service connections based on these backends. You can also create custom backends in your extension.

To create a service connection to your backend, click + Service Connection. See Create a Service Connection.

Create a Custom Backend

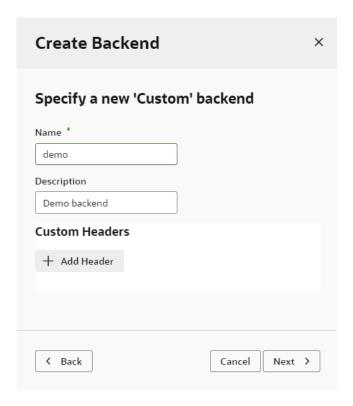
You can create your own backend to map to a custom server other than the built-in Oracle Cloud Applications backend.

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.
- In the Services pane, click + sign and select Backend.
- 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.





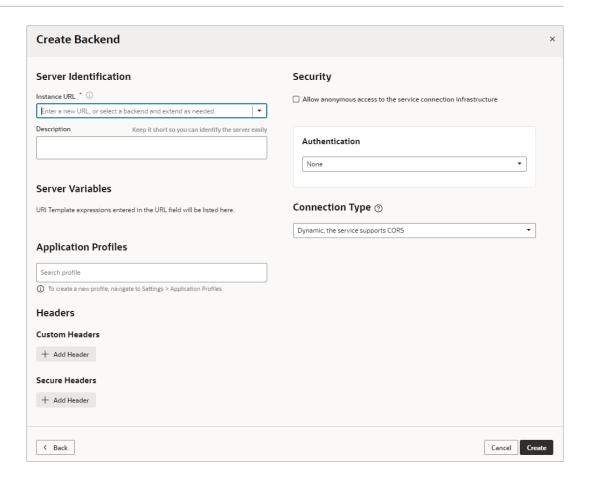
Optionally, click + Add Header under Custom Headers to add a static header that is
passed from the browser to the service. Enter a name for the header and its value, then
click OK.

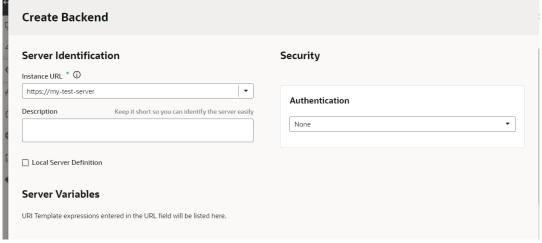
Headers can exist at both the backend level and at the server level. The difference is that backend headers are guaranteed to be applied to REST API calls, whereas server headers are applied to the call only if that particular server is in play. For this reason, using headers at the backend level is recommended.

Custom headers become available to you from the browser's Developer Tools console.

6. Click **Next** to display the second page in the wizard:







- Determine whether this backend needs a local server (see What Are Local Servers?) or insource server (see What Are In-Source Servers?). If you need a local server, click the Local Server Definition checkbox. (Most extensions will need a local server.)
- 8. Enter the instance URL and other information that your extension 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.

- 9. If you need more information about the options in the Authentication drop-down, see Set the Backend's Authentication Method and Connection Type.
- If you're using an in-source server and need help in determining server variables, see Add Server Variables for Backends.

11. 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.

If this backend was created with a local server, you'll need to provide the corresponding runtime server details in Oracle Cloud Applications so that the backend will work for the Preview action. See Provide a Backend's Runtime Server Details.

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 provided by the Unified Application.

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 <code>vb-catalog://backends/</code> 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 <code>vb-catalog://backends/fa/crmRestApi/rest,</code> where fa represents the parent backend.

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

Edit a Backend

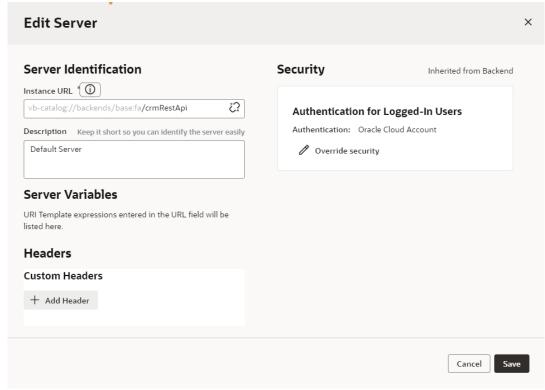
To edit a backend:

- 1. Open the **Backends** tab from the **Services** tab in the Navigator.
- 2. Select the backend you want to modify to display the **Overview** tab.
- 3. Change the description, if desired, or change the transform attached to the backend (see Add Transforms for more information.)
- On the Headers tab, use + Add Header to add static headers that are passed from the browser to the service.
- 5. On the **Servers** tab, select an in-source or local server to modify. Note that local servers are flagged with an extra "Local server definition" tag.





6. Click the **Edit** icon of to display the Edit Server dialog:



Modify the backend server details. These options are the same as those you configure when you define a backend. Depending on whether it's a local or in-source server, you'll see slightly different fields:

- In Server Identification, under Instance URL, change the modifiable portion (the part
 that isn't grayed out) of the instance URL or the description. If this is an in-source
 server, you can also change the server variables (described in Add Server Variables
 for Backends).
- In Headers, under Custom Headers, use + Add Header to add static headers that are
 passed from the browser to the service (although it's preferable to use the backend's
 headers rather than those on this tab—see step 4 in this procedure).
- In Security, you can change the authentication type used to connect to the backend. If this is a local server, you can also change the connection type (see Set the Backend's Authentication Method and Connection Type).
- 7. Click **Save** when you're done making changes.
- 8. Click the **Source** tab to view the contents of the services/self/catalog.json at the extension level. Edit the file if needed.

You can also access this file under services in the Navigator's Source View tab.

Remember, any changes you make here for a local server will **not** be reflected in the server's source code.

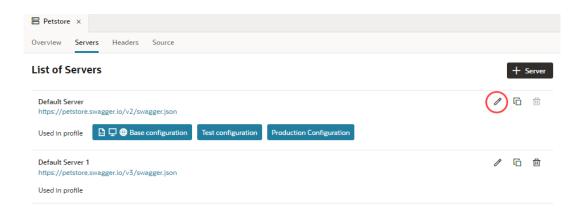


Edit a Backend Server's Authentication Details

After a backend is created, you can edit the server's authentication details from the **Servers** tab.

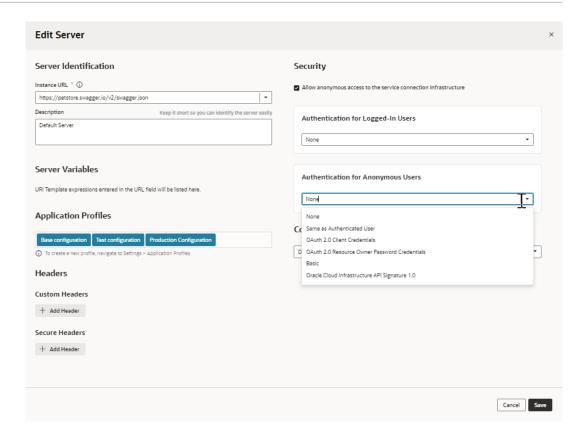
You might want to edit the authentication details when the authorization requirements of your app change, for example, if you you need to override the settings for the backend service. If you have multiple servers added to your backend, you may need to make changes in more than one server.

- Open the backend's Servers tab.
- 2. Click the **Edit** icon beside the server instance you want to edit:



- In the Edit Server dialog, you can use the Authentication drop-down list to change the authentication mechanism.
 - 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 mechanism for anonymous users in the Authentication for anonymous users drop-down list.





Add Server Variables for Backends

When creating or editing a backend, you can add server variables to the instance URL on the Servers tab for an in-source server. Server variables are not available for local servers.

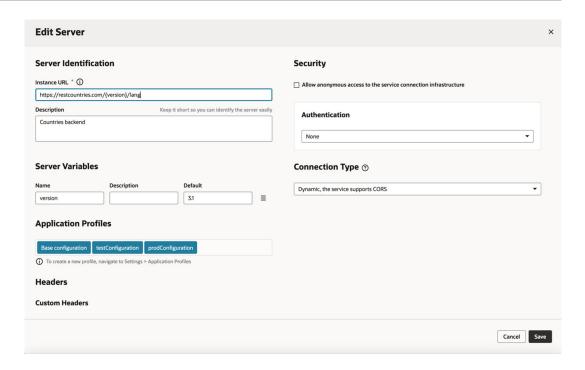
When you enter a valid URI template expression, such as {version}, in an instance URL, a server variable will be created automatically and displayed in the Server editor's **Server Variables** section. You can also create, edit, or reorder a list of variables. A default value must be set for each server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

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

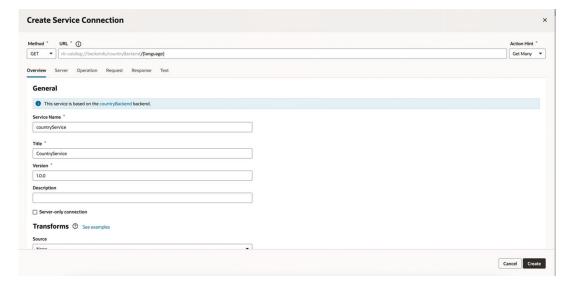
1. The server instance URL can be represented as https://restcountries.com/ {version}/lang.

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**:



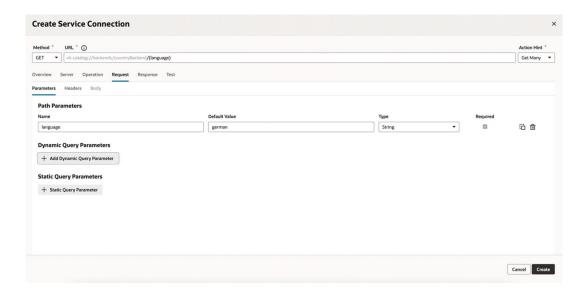


2. The endpoint is / {language}.
We define the service connection on the countryBackend by defining the endpoint and adding / {language} to the URL path.



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.
- 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.
- 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**.
- Click Save.



Add a Local Server to Use a Different Backend Definition During Development

Note:

Local servers have limited scope in this release, so it's best not to use them.

A local server is used to override the default server details for a backend, so that you can provide a URL and authentication type for the backend and its related service connections that are more applicable during development, without those details going into the source code.

Local servers are useful when you want to:

- Use mock service connections with a different set of credentials.
- Use the latest, work-in-progress service connections during development, without affecting the deployed App UI.

For example, suppose while developing the latest version of an App UI, you need to use the latest corresponding service connections—which contain new attributes important for development—rather than the older service connections used in the deployed App UI. To do this, you would:

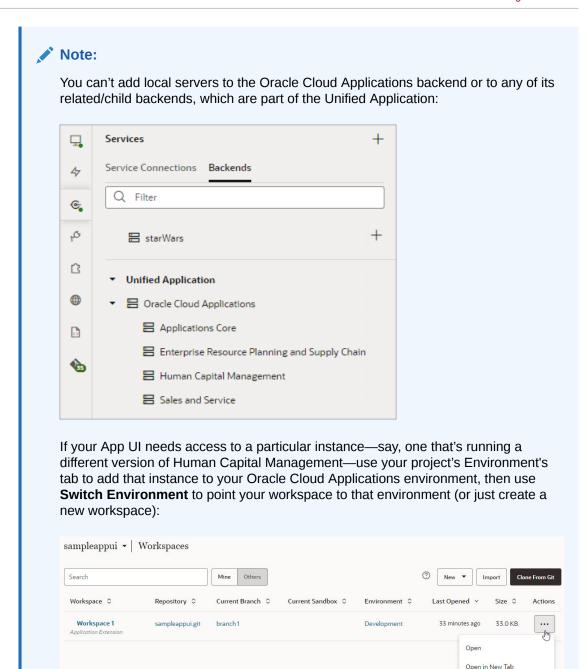
 Add a local server to the backend, providing details for the latest version of the service connections.

Note:

To ensure that your local server functions properly, use the value <code>Always Use proxy</code>, <code>irrespective of CORS support in the Connection Type field</code>.

- 2. Develop your App UI using the new attributes available with the latest service connections, working the new attributes into the pages and introducing logic around them. (Ensure that the application doesn't break if the attributes aren't present.)
- 3. When you're ready to deploy the updated UI, change the details for the backend's default server—which is always used for the deployed version of the App UI—to the details for the server with the latest service connections..
- 4. Remove the local server if it is no longer needed.





While server details for the default backend are stored in the source code (specifically in the catalog.json file), local server details are not, so you don't have to worry about them getting packaged with the App UI for deployment.

Rename

Export Application Extension

Switch Environment

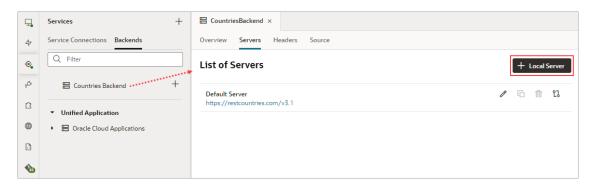
Copy Workspace Name

Change Ownership

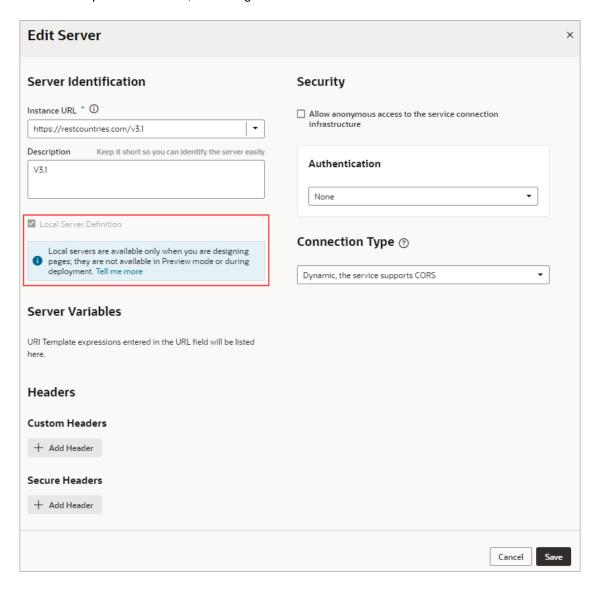
Delete

Once you've added a local server, the Page Designer's Live button uses the connection services related to that server instead of the default server. The Preview option, however, continues to use the backend's default server.

To add a local server to a backend, click + Local Server:



Provide the server details to use during development, and note that the **Local Server Definition** option is checked, indicating that this is the backend's local server:





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 App UI , 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* (relevant to local servers only) 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:

- Manage the credentials for accessing the service (if credentials are required).
- Manage identity propagation of the end user logged into the App UI (if the service supports the standard IDCS OAuth flows).
- Manage how your application connects to the service (via proxy or via Direct call).

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 App UI 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). (Fixed credentials are never available for in-source servers.)

To provide flexibility in how service connections are established, you can use different authentication methods for your extension's development cycle, as well as for each Oracle Cloud Application instance to which your extension is deployed. If needed, you can use methods of different types between development phases.

Note:

Before you connect to a service, an administrator may need to configure the VB Studio environment settings for backend services, or the settings for the external service or identity provider.

For certain authentication types, your REST APIs might need CORS support for both VB Studio and Oracle Cloud Applications domains (as reflected in the tables shown in the next few topics.) If you don't have this setting in place, you'll see a CORS Preflight error when the REST API is called. The backends that are inherited from the Unified Application must be pre-configured for CORS to allow access to VB Studio.

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 VB Studio and extract the user (or subject) from it. VB Studio 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 <heathermal / header.body.signature > and looks like this:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY30DkwIiwibmFtZSI6IkpvaG4
gRG91IiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJ
SMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c

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.

Identity propagation authentication methods are available for both *local* and *in-source* servers. Use local servers if you need to vary the URL or scope between different instances, like Test and Production. You can see an example of this in Connect to OCI Process Automation Services.

Here are the authentication methods that use identity propagation:

Authentication method	Description	Available for:
Oracle Cloud Account	This method is used in extensions to communicate with Oracle Cloud Applications. Since you will always have an out-of-the-box backend (available through the Unified Application) representing Oracle Cloud Applications and its child instances, you will probably not need to use this authentication type.	In-source servers Local servers
OAuth 2.0 User Assertion	Select this method to call an external system's REST API that can accept a token from the IDCS or OCI IAM identity domain attached to VB Studio. Such external services are represented in IDCS or OCI IAM as a resource application with a particular scope. You must be signed in as a user that is present in the relevant IDCS or OCI IAM identity domain.	In-source servers Local servers Runtime server configurations
	If you have an Oracle Process Automation or Oracle Integration instance in the same identity domain, you can use this authentication type. See Connect to OCI Process Automation Services for details.	
	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 <i>own</i> scope, rather than using the service's URL.	



Edit a Server's Authentication Details

After creating a service connection, you can edit its server's authentication details from the **Servers** tab.

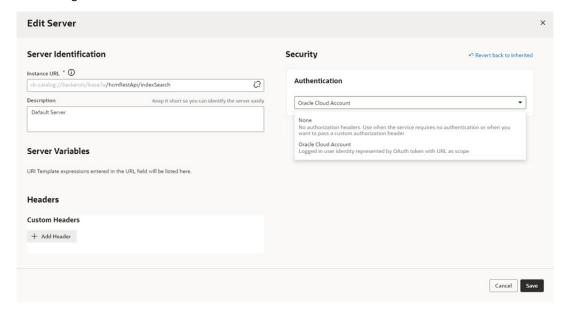
You might want to edit the authentication details when the authorization requirements of your extension change or you need to override the settings for the backend service.



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 Edit a Backend Server's Authentication Details.

To edit a service's authentication settings:

- 1. Open the **Servers** tab of the service connection.
- Click the Edit icon next to the server instance whose authentication details you want to modify.
- 3. If the connection is to a service in your Service Catalog, in the Edit Server dialog, click Override security to display the Authentication drop-down list, where you can change the settings inherited from the backend:



4. Click Save.

Sample Use Case: How to Use Authentication to Connect to Oracle Integration Cloud REST APIs

We've provided one end-to-end example that walks you through the steps for setting everything, including authentication, up to connect to an Oracle Integration Cloud REST API. As VB Studio provides additional authentication mechanisms, we'll include more examples like this one, using other services and other authentication mechanisms.



For our example, we'll be using Oracle Integration Cloud (OIC) REST API as the target REST API but the process is largely the sane for other REST APIs that are protected by the same IDCS with a JWT Assertion OAuth type grant.

Example Scenario

Your Oracle Cloud Applications test instance is preconfigured with VB Studio. You want to consume an OIC REST API in your extension. You have an OIC test instance, say https://oic-test.integration.mycorpcloud.com, that has a REST API $/ic/api/integration/v1/flows/rest/HELLO_WORLD/1.0/names/that you want to connect to the extension that's deployed on the test instance. However, when deploying to production, you'd like to use your OIC production instance, say <math>https://oic-prod.integration.mycorpcloud.com$, in your service connection.

Prerequisites

You need to satisfy these prerequisites before you can set up the connection:

- The OIC test instance REST API must:
 - Accept an IDCS-issued token (that is, the OIC test instance and the Oracle Cloud Applications instance should reside in the same IDCS/Identity domain).
 - Have CORS support for the test VB Studio and the corresponding Oracle Cloud Applications domain.
- The production instance REST API must:
 - Accept an IDCS-issued token (that is, the OIC production instance and the Oracle Cloud Applications instance should reside in the same IDCS/Identity domain).
 - Have CORS support for the production Oracle Cloud Applications domain.

Steps

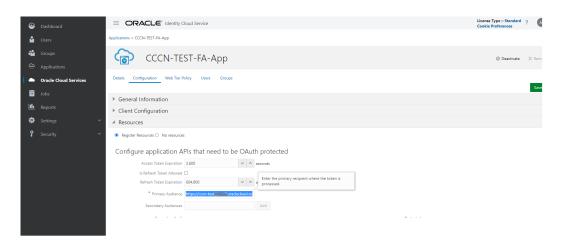
Here's the process you follow to set everything up:

- Locate the Oracle Cloud Applications OAuth Client application in IDCS
- 2. Add the required scope of the OIC instance to the Oracle Cloud Applications OAuth Client application
- 3. Set up a backend and a service connection in the extension
- 4. Deploy the extension to the Test instance
- 5. Deploy to the Production instance or to other instances

Locate the Oracle Cloud Applications OAuth Client application in IDCS

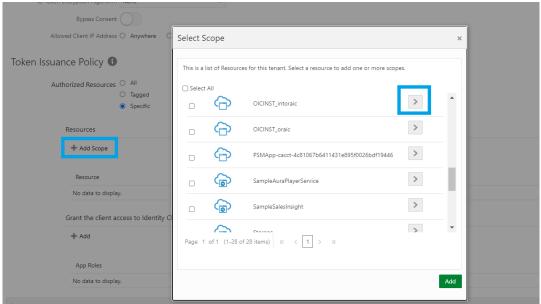
- 1. In IDCS (connected to the Test instance), navigate to "Oracle Cloud Services" from the hamburger menu.
- 2. From the list of all the services shown, find the one that represents the Oracle Cloud Applications Test instance and select it.
- To confirm this, navigate to the Configuration > Resources section.
 The primary audience or the secondary audience URL is the Oracle Cloud Applications
 Test instance's base URL.





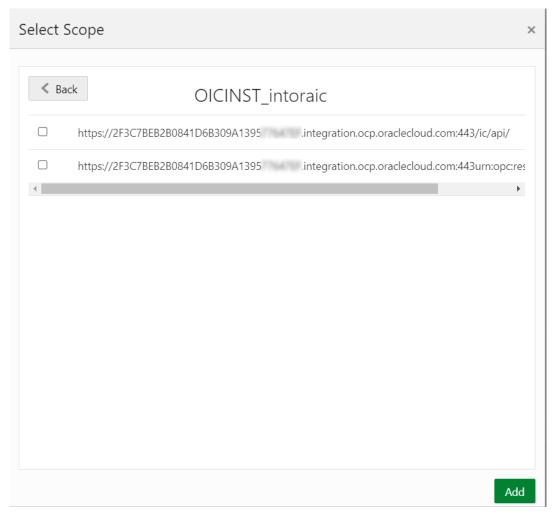
Add the required scope of the OIC instance to the Oracle Cloud Applications OAuth Client application

- In the Oracle Cloud Applications OAuth client, navigate to the Client Configuration >
 Token Issuance Policy > Resources Add scope.
 A popup opens.
- 2. Choose the scope from the existing resource applications in IDCS.



- 3. Navigate through the list of resources and locate the resource application that represents the OIC instance.
- Click the > button to select this resource app. The available scopes are displayed.
- 5. Select the scope that represents your REST API.

 For OIC, this will usually be the urn:opc:resource:consumer::all scope.



- 6. Note the scope https://<test uuid>/ic/api because you'll be adding it later in the VB Studio extension.
- 7. Click **Add** to add this scope to the Oracle Cloud Applications OAuth Client application.
- 8. Click **Save** to persist your changes.

Set up a backend and a service connection in the extension

In VB Studio, create a custom backend.
 See Create a Backend for more information.

Set the following attributes:

- a. Specify a valid name, like oicBackend, for example, for your custom backend.
- **b.** Make the type custom.
- c. In the Server details page:
 - Set URL to the service connection's base URL (https://oic-test.integration.oraclecloud.ocp.com in our example).
 - Set Authentication to OAuth 2.0 User Assertion.
 If you can't see this option, see the Tip below.
 - Set **Scope** so it corresponds to the IDCS OIC REST application (for example https://<test_uuid>/urn:opc:resource:consumer::all).



0

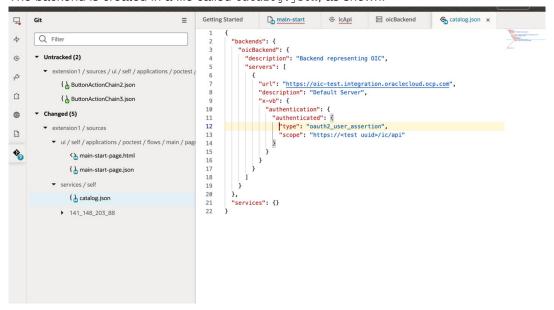
Tip:

If the version of VB Studio you're using is older than 22.04), you may not be able to see the **OAuth 2.0 User Assertion** option in the **Authentication** selection menu. Follow these steps to assign the authentication mechanism and scope:

- i. Create the backend and set **Authentication** to **None**.
- ii. From **Backends**, navigate to the **Source** tab and locate the "servers" section for the oicBackend in the catalog.json file.
- iii. Add the authentication mechanism and the scope in the "servers" section:

2. Click Create.

The backend is created in a file called catalog. json, as shown.



Now that we've created a backend that represents the OIC system, we'll create a service connection that uses this backend. The advantage of using a backend is that you can

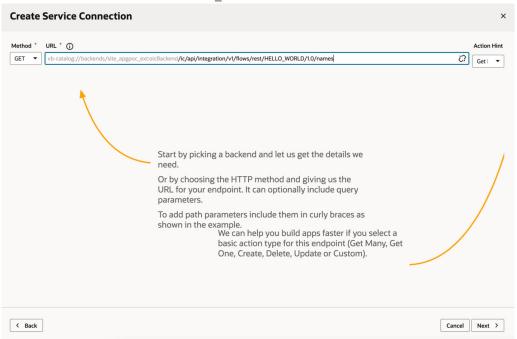
create multiple service connections based on it, without having to add the Server and Authentication details each time.

3. Add the service connection to the extension.

See Create a Service Connection to learn about multiple ways you can create service connections. The example we show in this step uses the **Define by Endpoint** option, which is explained in greater detail in Create a Service Connection from an Endpoint, but you could use one of the other options to create the service connection too.

Create the service connection by endpoint:

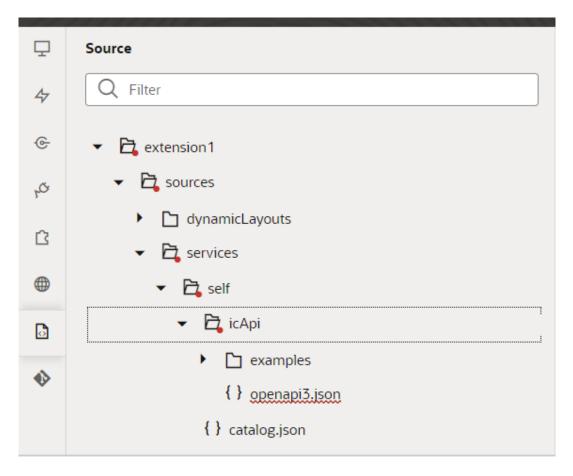
- a. Click + Service Connection.
- b. Click on the **URL** field to display a list of existing backends.
- c. Select the backend you just created (for example, oicBackend) and the URL will become vb-catalog://backends/<extn>:oicBackend.
- d. Add the rest of the URL path to the service connection (for example, /ic/api/integration/v1/flows/rest/HELLO WORLD/1.0/names.



- e. Click **Next** and add an appropriate **Service Name**.
- f. Then add other details, like headers, guery parameters etc., if any are needed.
- 4. Navigate to the **Test** tab and test the service connection.
- 5. When you're done, click Create.
- 6. Note the name of the service connection. You'll be using it later.

 Let's use the name "icApi" in this example. The source for this service connection is in the extension1/sources/services/self/icApi/openapi3.json file. You can confirm this by looking at the source tree.





Deploy the extension to the Test instance

- Follow the instructions in Preview Your App UI to run (preview) the extension and check if it was successful.
- If it was successful, commit the sources and push (or merge) them to the branch you'll be packaging from.
- 3. If the extension hasn't been configured already, follow the instructions in Set Up the Project for Testing to configure it.
- Deploy the extension on the test instance.
- 5. Test the deployed extension and ensure that it works correctly in the test instance.

Deploy to the Production instance or to other instances

 Repeat the steps in Locate the Oracle Cloud Applications OAuth Client application in IDCS and Add the required scope of the OIC instance to the Oracle Cloud Applications OAuth Client application for the production IDCS, production Oracle Cloud Applications pod, or production Oracle Integration Cloud instance.



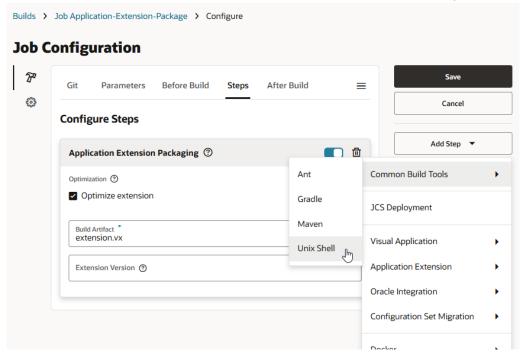
Since you hardcoded the URL and the scope in the Backend source, you'll need to change them before deploying them to production pods if the OIC production instance is different than the test instance.



- 2. To do so, in the pipeline, add a preprocessing step that changes the URL and scope to that of the OIC production instance:
 - **a.** Follow the instructions in Create and Configure Production Build Jobs to create the production build pipeline.
 - b. Open the Packaging Build job in the pipeline and click **Configure**.



c. From the menu Common Build Tools select Unix Shell and add a new step.



d. • In the script area, add the command to change the OIC test instance URL and scope to that of the production <code>catalog.json</code> file. Use this format for the commands:

```
sed -i 'originalText+modifiedText+g' filename with fullpath
```

For our example, we'll use this:

```
sed -i 's+https://oic-test+https://oic-prod+g' extension1/sources/
services/self/catalog.json
sed -i 's+<test uuid>+<prod uuid>+g' extension1/sources/services/self/
catalog.json
```

e. Click the up arrow ^ to move the newly added step to the top of the job, before the Application Extension Packaging step.





- f. Save the job.
- g. Run the production pipeline and check the deployed extension.
- Perform the same steps for any other instances where the extension needs to be deployed.

Note:

In this case specifically, we created custom service connections with hardcoded URLs and scopes in the <code>catalog.json</code> file. Therefore, when refreshing Test extensions with Production versions, the URL won't be automatically changed to the one used in Test. You need to redeploy the production branch to the test instance to have the correct URLs/scopes after refreshing the Test extensions with the Production versions. This redeployment isn't needed if you don't have custom service connections to other services.

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 issuing system, Oracle Cloud Applications (service) infrastructure ensures that the token is reissued when the current one expires, provided the user is still logged into Oracle Cloud Applications and the calling REST API complies with the OAuth specification. If the user is no longer logged into Oracle Cloud Applications, any attempts to issue a new token will redirect the user to the login page.



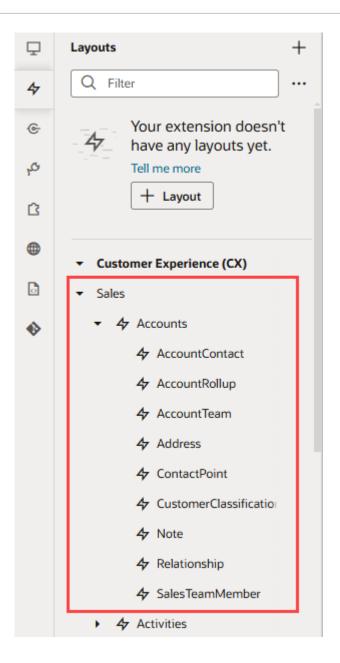
6

Work With Layouts in Your Extension

A Layout is where you configure your extension's dynamic tables and forms, everything from the component's data source—usually an Oracle Cloud Application service endpoint—to how the component is rendered and behaves on a page.

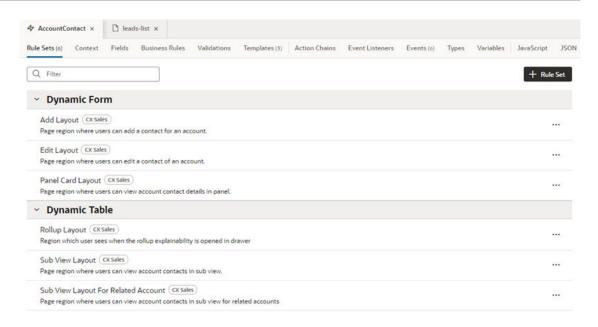
You can see the Layouts available in your extension in the Navigator's Layouts pane, which shows any Layouts you've created, and a list of Layouts in dependencies, grouped by pillar, that you can browse and access. The list contains Layouts in dependencies you have added to your extension, as well as other Layouts in dependencies in the same pillar as your extension's dependencies (except in the case of dependent extensions belonging to the CX or Other pillars, where only layouts from the dependencies you've added show). Each Layout is associated with a data source, which can be a parent object, like Accounts, or a child object, like AccountContact:





Exploring a Layout

A Layout represents both the *data source* for one or more dynamic components, as well as the *rule sets* that control the component's behavior on the page. Let's look at the Layout associated with the AccountContact object:



In this Layout, AccountContact is the data source, and there are six rule sets defined, all of which come from a dependency. (You can tell this from the **CX Sales** badge beside each rule set.) If you created your own rule set based on the AccountContact data source, it too would appear on this tab. In other words, if a dynamic component displays data from the AccountContact object, its rule set will appear on this tab. However, a rule set can be listed here and *not* used, meaning there might not be any components using the rule set at the moment.

All the rule sets from CX Sales shown in this image are *extendable*, which means you can customize them in your extension. You know they are extendable simply because they are listed in the **Rule Sets** tab—if they weren't marked as extendable, they wouldn't appear here. When you extend a rule set, your changes affect every dynamic component that uses that rule set. You can tell which rule sets you've extended because they have a gray vertical bar next to its name.

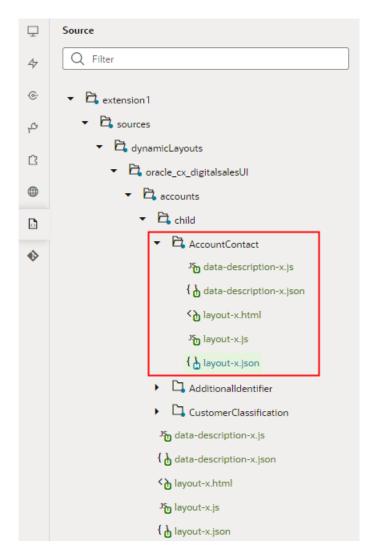
Extending a Layout in a Dependency

While you can't edit the artifacts in a dependency directly, you can *extend* them. When configuring dynamic tables and forms, this usually involves changing one or more rules sets in a Layout, or using business rules to override field properties.

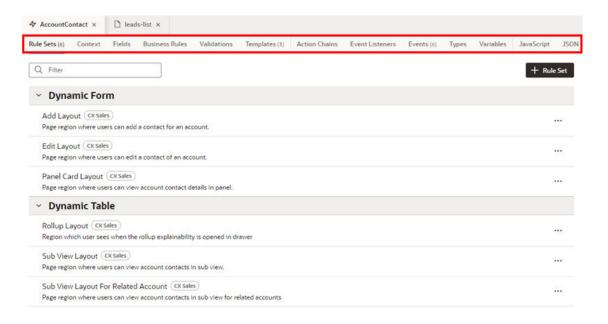


You can only extend the artifacts that are marked extendable in the dependency. Some rule sets defined in the dependency might not be extendable, in which case they won't appear on the Layout's Rule Sets tab.

When you modify a Layout defined in a dependency, you'll see the impacted files in your extension's **Source** pane in the Navigator. Files that extend a dependency are appended with -x. In this example, the layout-x.json file extends the layout.json file in the dependency, and so on:



To extend a rule set you start with the Rule Sets tab, but there are other tabs for configuring how the Layout's dynamic forms and tables behave as well. Let's take a quick look at these tabs now:



Description
Shows all the extendable rule sets available in the Layout.
Use this editor to decide which fields in the Layout's data source will be exposed in the component (which you do in a <i>layout</i>), and the conditions under which a given layout is displayed (which you do by setting the <i>display logic</i> .) See Create a Rule Set in a Layout and Create a Layout for a Dynamic Table or Form.
Lists parameters in the object context that can be passed to the dynamic layouts from the page. The Unified Application's developer at Oracle or a dependency's developer can define context parameters that might otherwise be inaccessible to a dynamic component. See Use Context Parameters in Extensions.
You can use the context parameters when building conditions for rules and creating expressions.
Lists all the data object's fields that can be used in the Layout's rule sets.
Use this editor to configure some field properties, and to create virtual fields that can be used in the dynamic components. See Override Field Properties and Create Fields For a Layout.
Use the business rules editor to create rules to override field properties in dynamic forms and tables at runtime, based on conditions such as the user's role, or other field values. See Control Your Display with Business Rules .
Use the validations editor to create rules for displaying 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. See Display Messages in a Form.
Lists the field and form templates that can used in the Layout.
Use this editor to create new templates, and to edit the templates you've created. See Use Field and Form Templates
Lists the action chains defined in the Layout that can be used in the dynamic components.
Use this editor to create action chains in the Layout, and to edit the action chains you've created. See Work With JavaScript Action Chains.



Editor	Description
Event Listeners	Lists the event listeners defined in the Layout.
	Use this editor to create event listeners that listen to events in the dynamic component, and to edit the event listeners you've created. See Create Event Listeners for Events.
Events	Lists the events defined in the Layout.
	Use this editor to create events used to start action chains in the dynamic component, and to edit events that you've created. See Define Events in Your App UI.
Types	Use this tab to create and edit the types used in the Layout. See Create Types.
Variables	Use this tab to create and edit the variables and constants used in the Layout. See Create Variables.
JavaScript	Use this tab to edit the JavaScript used in the Layout. See Work With JavaScript.
JSON	Use this tab to view and edit the Layout's JSON file. See Work With JSON.

For more about extending and configuring Layouts in a dependency, see Customize Dynamic Tables and Forms.

Create a Layout

When you add a dynamic component to a page, the Configure Layout wizard automatically creates a Layout and rule set in the extension for you, if needed. If there's already a Layout for the data source you selected, your new rule set is added to the existing Layout. See Add Dynamic Components to Pages.

You can create a Layout even if you're not quite ready to add a dynamic component to a page. For example, say you want to display data from the AssetActivities object in your application, but the dynamic form component displaying it will be added by another team member. You could create a Layout for the object, then do one or more of the following:

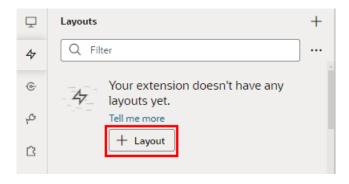
- Create a rule set for determining the fields that will be displayed in the form;
- Create field and form templates for rendering the form;
- Create virtual fields used in the form; and
- Add validation rules to the fields.

Your team member could then add the dynamic form where needed, select the Layout's data source, then select the rule set that you created in the Layout. In addition, as long as you click the **Enable Extensions** check box when you create your Layout, all the Layout's other artifacts may be used by your team member's extension as well.

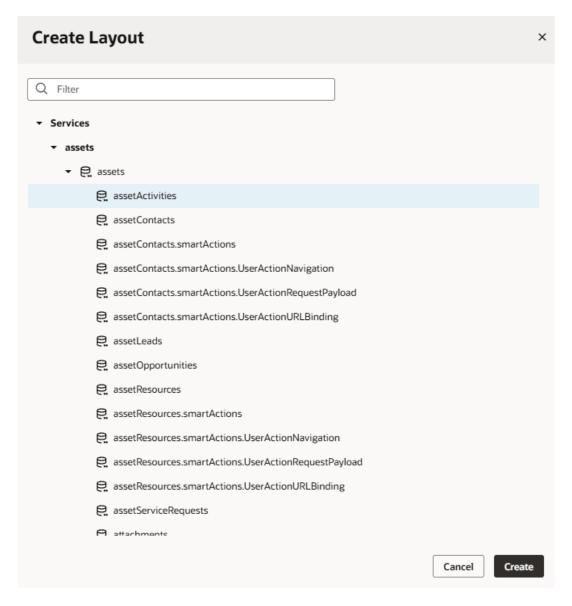
To create a Layout:

1. Open the Layouts pane in the Navigator, then click + Layout



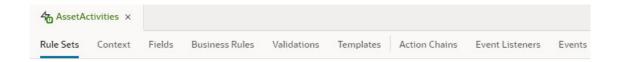


2. Select the data source in the Create Layout dialog, then click Create:



Your extension *must* have access to a service to create a Layout. This might be from a service connection you've added to the extension, or a connection defined in one of the extension's dependencies.

After you create the Layout, the Rule Sets tab opens in the Layout, but it won't have any rule sets:





You don't have any rule sets defined yet.

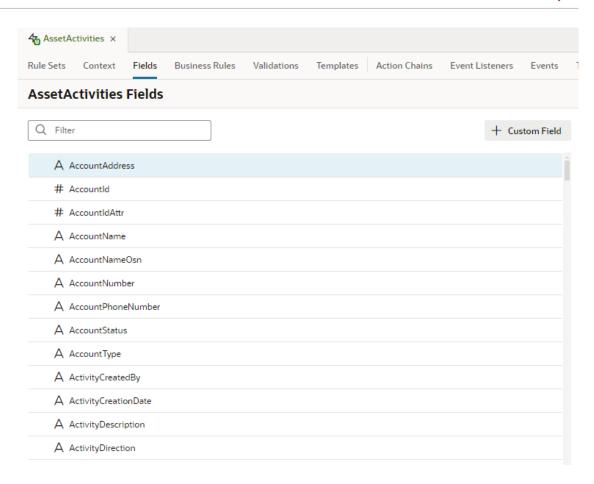
Rule sets define rules for choosing and displaying fields in dynamic form and table components.

Tell me more



At this point, you could create a rule set to define the fields that should be displayed in a dynamic component, or just open the **Fields** tab to explore the list of the fields you can choose from:





Create a Rule Set in a Layout

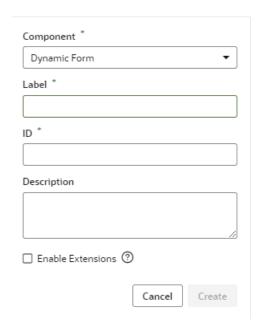
When you add a dynamic form or table to a page, you first select a data source (usually through a Quick Start), then either create a new rule set, or select one that already exists the Layout for the data source.

You can create a rule set in a Layout at any time, even if you don't have a dynamic component ready to use it yet. You might want to do this in preparation for dynamic components that will be added to your extension—or someone else's—later.

To create a rule set in a Layout:

- Open the Rule Sets tab in the Layout, then click + Rule Set
 The Rule Sets tab might already contain some rule sets, or it might be empty if the Layout was created before any rule sets were created.
- Select the type of component that the rule set can be used by, either Dynamic Form or Dynamic Table:





- Type the name of the rule set in the Label field and, optionally, add a description.
 The description should be meaningful, so that it's clear when the rule set should be used.
- Select Enable Extensions if you want to allow other extensions to use this rule set. Click Create.

If **Enable Extensions** is not selected, the rule set won't be able to be used in any extension that adds your extension as a dependency.

After creating a rule set, the next step is to define which fields should be displayed in the component, and the conditions governing when they should be displayed.

- Determine What's Displayed at Runtime With Rule Sets
- Create a Layout for a Dynamic Table or Form

Override Field Properties

The Layout's Fields tab lists all the fields in the data source that can be used in the Layout's rule sets. In the Fields tab, you can view properties of the fields, and override the values of some of the properties.

When you modify the property value of a field in the Field's tab, the new value is applied every place that field is used in the Layout. For example, if you set a field's Read Only property to True, the field will be Read Only in every rule set where the field is used. You can see a field's properties in the Properties pane when you select the field. The field's properties are defined in the data source's service definition.



Note:

You can't create or modify the fields in your Oracle Cloud Application, or the service definition used by a Layout, but you can override some field properties, such as "Read Only" and "Required". So if a field's Required property is set to False in the service definition, you can override the property to make it more strict and set it to True. This won't change the description in the service definition, where the property will still be set to False. However, you can't override a property to make it less strict, meaning you can't set a Required property to False if it is already set to True in the service definition.

To override field properties based on conditions, such as a user's role, or the screen size of the user's device, you can override field properties at the object level using business rules and override field properties in a component using rule sets.

Depending on the field, you might be able to modify the following field properties in the Fields tab.

To override	You can
Required	Set the property to True, False, or to an expression. See Set a Field as Required.
Read Only	Set the property to True, False, or to an expression. See Set a Field to be Read Only.
Default Value	Enter a value or expression in the Default Value field.
Value Precedence	Specify what happens when a user enters a value for a field that has a calculated value. See Specify How a User Input Overrides a Field Value.
Layout Discriminant	Map a field to a global field template using its Layout Discriminant ID. See Map a Field to a Global Field Template.
Validator	Specify and configure an existing or custom validator for the field. See Add Converters and Validators to a Field.
Converter	Specify and configure an existing or custom converter for the field. See Add Converters and Validators to a Field.
Label Hint	In the Label Hint field, enter the value you want displayed in the page as the field's label.
Format	In the Format dropdown list, select how you want the field's value rendered in the page. For example, to obfuscate the value of a string, you might choose the 'password' format.

In addition to modifying the properties of fields defined in the service definition, you can also create virtual fields that can be used in the Layout's rule sets. See Create Fields For a Layout.



7

Preview, Share, and Publish Your Extension

As you design and develop your application, you'll want to test it and see how its pages look and function. You can do this using Live view and Preview, but both these views are visible only to you. When you're ready for others to see your work, share your application to give others a chance to try it out. Then, when you're done making changes, publish your application to make your changes public.

VB Studio provides different options to view and test your extension at any point during the development process.

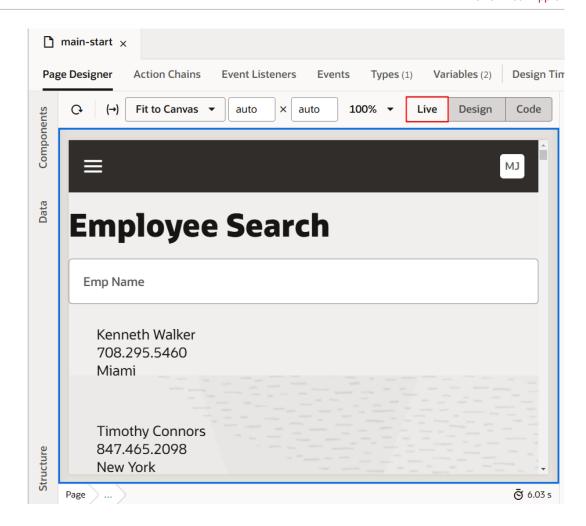
- When you're still working on your application's pages, use Live view to make real-time edits (for example, to adjust layout and styles on a page), test interactions (for example, to see what happens when a button is clicked), and visualize changes right as you're making them in the Page Designer. When your app is more or less complete, use Preview to open your application in a separate browser window and test overall layout and functionality, without affecting your actual users.
 See Preview Your App UI.
- When you're ready to share your changes with others, use Share to generate a URL that your team members can use to test your application.
 See Share Your App UI.
- Once your application has been thoroughly tested, use **Publish** to make your changes public. The Publish action commits your changes to the Git repository, and either deploys directly to your Oracle Cloud Applications instance or kicks off the package and deploy jobs for your extension. (These jobs were created for you automatically when you created your workspace.) You'll see a few messages pop up in the Designer telling you when each job starts and finishes.

See Publish Your Extension and View Your Deployments.

Preview Your App UI

There are two ways to preview and test your application before deploying it. Use the Preview action to see your application in a separate browser tab, as it would appear in a deployed environment. This method is ideal for checking overall layout and functionality, especially once your application is further built out. On the other hand, to quickly check your changes as you make them, *directly* in the Page Designer, use Live view.

 To quickly check how a page behaves, you can use Live view in the Designer. Live view is an in-tool preview that you can use to view and debug the page you're currently viewing in the Page Designer. To view a page in Live view, open a page in the Page Designer and select Live in the toolbar:



Clicking Live switches the page to an interactive mode in the canvas area, where you can click components and see changes instantly as you edit. For example, after creating a business rule, switch to Live view and enter values to confirm that the rule is triggered. In this way, Live view enables fast, iterative design because you can immediately check the impact of changes without waiting for a separate preview.

Live view also gives you the option to tweak application code and render pages even when they're missing the required context. For example, when a page is missing input parameter values provided by a flow-level variable, you can populate the missing context by calling REST in an action chain to look up the best available object record for the page to work. To do this, update an action chain that depends on the missing flow-level variable to add a check for \$env.dt and provide a mock value that allows the page to render, for example:

```
define([
  'vb/action/actionChain',
  'vb/action/actions',
  'vb/action/actionUtils',
], (
  ActionChain,
  Actions,
  ActionUtils
) => {
  'use strict';
```

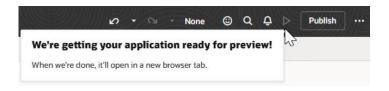
```
class vbEnterListener extends ActionChain {
    /**
    * @param {Object} context
    */
    async run(context) {
      const { $page, $flow, $application, $constants, $variables } = context;
    if (context.$env.dt === true) {
         $variables.heading = 'Hello from this side';
         return;
    }
    $variables.heading = 'Hello from that side';
}
return vbEnterListener;
});
```

Any time a page is running in Live view, the \$env context is passed into action chains, with dt set to true. You can define your own action chains (even use existing vbEnter ones) to run only in Live view simply to render pages without the required context.

Note that some behavior in Live view, such as how pages are rendered, or using links to navigate from the current page, might not work as expected. When you're ready to more fully test your application, use Preview.

Preview displays your application in a separate browser tab, so you can see how your
pages look and function before publishing. Use this method to quickly test UI and layout
changes, navigation between pages, and what happens when you add and edit data, all
without affecting your actual users.

To preview your application, click **Preview** in the header:



When you use Preview, you'll see the pages as they look based on your user role. This means that if the App UI has any rule sets that evaluate the user's role, this will affect the data you see in the pages.

You can't share an App UI preview URL with your team members. If you want team members to see your changes, you'll need to commit the changes to a branch, then use the Share action to get a URL you can share.



Tip:

If your extension also contains another App UI, you can preview its latest changes too, simply by replacing the App UI's name in the URL: https://<hostname>/<FusionAppID>/redwood/<App UI name>/<flow name>

Note:

The option you choose is a matter of preference—though the form factor is a bit more pleasing when using Preview. Live view lets you immediately test interactions on a page, right from the Designer itself, without waiting for a preview. However, Live view isn't a substitute for Preview, and some behavior in Live view might not work as expected. For this reason, you'll likely end up using both methods to preview and test your application, depending on your testing needs at any particular moment.

Share Your App UI

While App UI previews can only be seen by you in your workspace, you can use the Share action to allow others to test your work. The Share action generates a URL for the App UI that you can give to your team members.

The URL generated for each branch in your workspace is unique, so you can give distinct URLs to your reviewers to get feedback on the changes in each branch. Suppose you're working on <code>branch_A</code> and share your changes with a reviewer; while waiting for feedback, you can switch to <code>branch_B</code>, make changes to a new set of pages, and share an entirely new URL with these changes alone. This helps you isolate changes in distinct branches and share multiple versions — one from <code>branch_A</code> and another from <code>branch_B_with</code> your reviewers.

Note:

The URL you're sharing might not be the same as the URL you see when previewing the App UI in the Designer.

Tip:

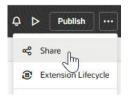
Sharing your extension triggers optimization to improve performance. You may want to define your own optimization configuration to specify the resources you want to include and exclude in the optimized resources bundle *before* you share the extension. See Customize Optimization.

Further, if you use the <code>vb-optimize</code> Grunt task to optimize your extension and have defined custom settings in the <code>Gruntfile.js</code> file, you must migrate your configuration from <code>Gruntfile.js</code> to <code>build.json</code> before you share the extension. See <code>How to Migrate Gruntfile.js</code> Configuration Into <code>build.json</code>.

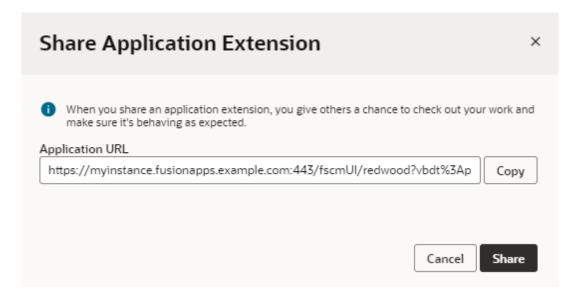
To share your App UI:

- Open your workspace in the Designer.
- 2. If you're using a sandbox, confirm that your current sandbox is the one you want to share.
- 3. Click **Menu** in the upper right corner and select **Share**:





4. Click Copy in the Share Application Extension dialog to copy the URL generated for the App UI, which you can then share with your reviewers.



If you're configuring an Oracle-created App UI and your changes involve updates to the data model, you'll also need to share the name of your sandbox with your reviewers. They will need the sandbox so they can check your changes using the correct data model.

Keep in mind that the format of the shared URL changes depending on whether your workspace is associated with a sandbox or not.

Click Share.

When the URL is ready to be shared, **Share** will be replaced by **Done**.



If your application cannot be shared, click **Open Logs** in the Share Application Extension dialog to view and fix build-related errors.

6. Send an email with the generated URL to your team members, taking care to include the name of the sandbox if applicable.

Share is one way to open up your App UI to team members. But if you actually want to share your *code* with a co-worker, you can export your workspace as an archive. Your co-worker can then create a new workspace and import the archive to create a new Git repository containing your files. See Export Your Workspace as an Archive and Import an Extension Archive.



Customize Optimization

When your extension is shared or deployed, all its resources are optimized to improve performance. If you want to specify the resources you want to include and exclude in the optimized resources bundle, you can define your own optimization configuration in a build.json file.

Optimization, by default, includes all resources that match the regular expression .* (\\.(js|json|html|css))\$ in an optimized resources bundle named vb-app-bundle. During this process, stylesheets, HTML, and JSON files are minified, and RequireJS bundles created in the embedded build server that prepares your extension to be shared or deployed. To customize this configuration to include and exclude resources, you'll need to create a build.json file and define your own configuration before sharing or deploying your extension.

- 1. Click **Menu** in the upper right corner and select **Settings**.
- 2. Click Create Build Configuration under Optimization to create the build.json file. If the file already exists, click Open Build Configuration.
- 3. Define your build configuration in the build.json editor to specify the resources you want to include in and exclude from the optimized resources bundle.

Here's the schema you must use to define optimization in build.json under the optimize element:

Property	Description
optimize.suppress	(Boolean) If set to true, optimization is disabled. If the property is not defined or is set to false, optimization is determined by the Optimize Application setting defined in a Packaging build step, or by individual optimizer settings (for example, optimize.require-js.enable). Default is false, meaning resources are always optimized before the extension is shared or deployed. It's simpler to disable optimization from the Settings editor, instead of manually editing build.json. See Suppress Optimization.
optimize.require-js	(Object) Configuration of the RequireJS optimizer that generates resource bundles.
optimize.require-js.enable	(Boolean) Whether RequireJS module bundles should be created. Enabled by default.
optimize.require-js.vb- bundles-config	(Object) Custom RequireJS module bundles, defined using this schema:
	<pre><bundle_name> find</bundle_name></pre>
	ids
	modules
	find
	ids
	exclude
	find
	ids



Property	Description
<pre>optimize.require-js.vb- bundles- config.bundle_name.modules</pre>	Modules that should be added to the bundle, defined using this schema:
	ids
optimize.require-js.vb-bundles-config.bundle name.exclude	Modules that should not be added to the bundle, defined using this schema:
, <u> </u>	find ids
	Exclusions are applied to all bundle modules, including those added following transitive module dependencies.
optimize.require-js.vb- bundles- config.bundle_name. [exclude modules].find	List of regular expression patterns used for matching optimized application resources. Regular expressions starting with an exclamation mark (!) are considered negative; resources matching these patterns won't be included.
<pre>optimize.require-js.vb- bundles- config.bundle_name. [exclude modules].ids</pre>	List of module IDs
optimize.require- js.emptyPaths	(Array) Comma-separated list of RequireJS paths that are set to empty. This way, you indicate that this dependency must be ignored (and also that its sub-dependencies must not be recursively traversed).
optimize.css	(Object) Whether stylesheets should be optimized. Enabled by default.
optimize.css.enable	(Boolean) Set to false to disable stylesheet optimization.
optimize.html	(Object) Whether HTML files should be optimized. Enabled by default.
optimize.html.enable	(Boolean) Set to false to disable HTML file optimization.
optimize.json	(Object) Whether JSON files should be optimized. Enabled by default.
optimize.json.enable	(Boolean) Set to false to disable JSON file optimization.

For default and other example configurations, see Optimization Configurations for build.json.



Tip:

If you use Grunt tasks to optimize your app, you must migrate your existing Grunt $configuration \ from \ {\tt Gruntfile.js} \ into \ {\tt build.json}, \ so \ your \ custom$ configuration can take effect during resource optimization when your extension is shared or deployed. The configuration in build.json is similar to what you define in Gruntfile.js. See How to Migrate Gruntfile.js Configuration Into build.json.



Optimization Configurations for build.json

When you create the build.json file (located under /extension1/sources/settings and accessible from the extension-level Settings editor), you can specify the resources you want to include and exclude in the application's optimized resources bundle.

Include all resources and exclude specific ones

You can extend this configuration, for example, to exclude some third-party libraries that are part of the application's sources, say, bcrypt.min and crypto-js (defined for the deal-registration flow). In this case, here's how you might define build.json:

```
{
    "optimize": {
        "require-js": {
             "vb-bundles-config": {
                 "vb-app-bundle": {
                     "modules": {
                         "find": [
                              ".*(.(js|json|html|css))$",
                              "!bcrypt.min",
                              "!^flows/deal-registration/crypto-js"
                         ]
                     }
                 }
            }
        }
    }
```

The find element specifies a list of regular expressions that are matched against your application. Regular expressions starting with an exclamation mark (!) are considered negative, so resources matching these patterns are not included.

Include and exclude specific resources

Here's an example that adds specific resources to vb-app-bundle, but excludes the fontawe some file:

```
{
    "optimize": {
        "require-js": {
            "vb-bundles-config": {
                 "vb-app-bundle": {
                     "modules": {
                         "find": [
                             "app-flow.js",
                             "^build",
                             "!^build/components/oj-odcs/2.5.1/fontawesome",
                             "^flows/",
                             "^pages/",
                             "^resources/strings",
                             "^resources/css",
                             "^services/"
                         ]
                     }
                }
            }
        }
    }
}
```

How you organize your app's resources into bundles depends on your application's structure as well as what you deem critical for application start. Try to bundle resources in such a way that what's really needed by the application is placed in one bundle while the rest goes into one or more other bundles.

Here's an example of configuration defined to create three specific RequireJS module bundles:

- One for all resources that belong to the application flow dashboard. This bundle includes all files that match the flows/dashboard pattern (in other words, all pages, modules, resources, and nested flows stored in the flows/dashboard directory). It, however, excludes the helpers module, referred by one of the page modules.
- Two for resources that belong to the application flow customers. In this case, nested
 flows (which are placed into a separate bundle later on) are excluded—as is the referred
 helpers module.
- Three for resources that belong to nested flows of the customers flow.

In addition to these three bundles, a fourth base bundle includes application resources, such as shell pages, libraries, and styles. This bundle explicitly adds the referred helpers module.



```
"flows/dashboard"
                 1
             },
              "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 using emptyPaths

Sometimes, when your app includes a custom web component that refers to a third-party library (for example, resources/components/mycomponent that references ckeditor), you might want to include the component in your bundle but not the third-party library. To do this, you can use "!^resources/components/mycomponent/ckeditor" in build.json. But the library will still be included if code in your bundle references it. It may be code in the custom web component itself, or something in a file that refers to this library, for example, code in the app-flow.js file included in the bundle that refers to 'resources/components/

mycomponent/ckeditor'. In such cases, you can add the emptyPaths element to build.json to ignore the dependency and also not recursively traverse its sub-dependencies:

```
"optimize": {
    "require-js": {
        "emptyPaths": [
            "ckeditor"
        "vb-bundles-config": {
          "dashboard": {
            "modules": {
                "find": [
                     "flows/dashboard"
            },
             "exclude": {
                "ids": [
                     "helpers"
         "base": {
             "modules": {
                "find": [
                     "app-flow.js",
```

Here's an example emptyPaths definition that ignores multiple libraries:

How to Migrate Gruntfile.js Configuration Into build.json

If you use Grunt tasks to optimize your application, it's important to migrate configuration you've already defined in the <code>Gruntfile.js</code> file into <code>build.json</code>. Without this definition in <code>build.json</code>, resource optimization that occurs by default when your extension is shared or deployed won't include your customizations.

- 1. Click **Menu** in the upper right corner and select **Settings**.
- Click Open Build Configuration under Optimization to open your extension's build.json file. If you don't see this option, click Create Build Configuration to create the build.json file (under /extension1/sources/settings) and open it from within the Settings editor.

3. Add this snippet to build.json:

- 4. Open your extension's Gruntfile.js file (located at the root of your extension in Source view).
- 5. Move the following configuration from Gruntfile.js to build.json:
 - Move the content of vb-require-bundle.options.bundles from Gruntfile.js into optimize.require-js.vb-bundles-config in build.json.
 - Move the content of vb-require-bundle.options.emptyPaths, if defined, from Gruntfile.js into optimize.require-js.emptyPaths in build.json.

Remember to fix the syntax for JSON schema. For example, here's a side-by-side comparison of the configuration in Gruntfile.js and build.json:

'use strict'; module.exports = (grunt) => { "optimize": { "require-js": { require('load-grunt-tasks') (grunt); "emptyPaths": ["ckeditor" grunt.initConfig({ "vb-require-bundle": { 1, options: { "vb-bundles-config": { emptyPaths: ["vb-app-bundle": { 'ckeditor', "modules": {], "find": ["bundles": { "vb-app-"^flows/", bundle": { "modules": { "^pages/", "find": "^resources/", [**"** ! "^flows/", ^services"] "^pages/", } } "^resources/", } " <u>!</u> } ^services/", }], }, **}**, }, **}**, }, }); };

build.json

6. Remove vb-require-bundle from Gruntfile.json, so your file looks something like this:

```
module.exports = (grunt) => {
    require('load-grunt-tasks')(grunt);
};
```

Debug and Audit Your Code

Gruntfile.js

As you work on your extension—and especially before you share or publish it—it's important to test and debug your App UI to ensure a smooth end-to-end experience for your users.

You have several options to ensure your App UI's code is error-free:

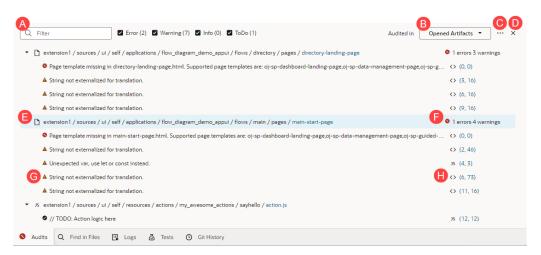
 Browser tools: Because your VB Studio extensions are essentially client-side HTML apps written in JavaScript, you can use your browser's development tools for debugging. When you preview your extension in Chrome (for example), you can use the Developer Tools option to diagnose problems quickly:

- 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 view console output and get a complete picture of what's going on in your extension. If your console only shows errors, you might need to set a different log level for your Oracle Cloud Application instance. See Change the Log Level.
- Audits pane: When you use Audits, VB Studio scans and runs checks on your extension's 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 are also reported.

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 App UIs icon in the Navigator toolbar as well as against the particular artifact in the App UIs pane. The badge persists until the error is fixed or until the session ends.

1. Click **Audits** to view the Audits pane.

If you see 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 check boxes to focus on errors.
В	Scope of the audit. By default, only artifacts that are open are audited, for example, the directory-landing-page, main-start-page, and action.js. 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.
С	Menu with options to edit audit settings or revalidate the entire extension.
D	Option to close the Audits pane.



Label	Description
E	Path to the artifact that contains issues, for example, the main-start-page under extension1/sources/ui/self/applications/appname/flows/flowname/pages.
F	Number and type of issues in the artifact, for example, 1 error 4 warnings.
G	Message about the issue in the artifact, for example, String not externalized for translation.
Н	Line and column number of the issue, for example, (6, 73) indicates that the String not externalized for translation issue exists in line 6, column 73 of the main-start-page artifact in Code view.

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 menu option 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.
- 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.



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.

3. 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) for Oracle JET at https:// static.oracle.com/cdn/jet/. Every Oracle 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 VB Studio backend; it runs on the client.

Every Oracle JET release includes the JAF utility and JAF metadata, plus the JAF metadata for previous releases of JET. You can configure built-in JAF rules in the Source editor of the Audits feature. Custom JAF configurations that deal with the file system, custom rule plugins, and so on will not be evaluated because JAF does not execute on the Visual Builder backend. JAF runs on the client.

Here's how to enable a built-in rule that is disabled by default:

- a. In the Audits pane, click Menu (***) and select Edit Settings.
- b. 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 Understand the JAF Audit Engine in *Using and Extending the Oracle JET Audit Framework*.

Troubleshoot Build Issues

If you run into errors when you try to share or deploy your application, you might need to check the build logs to troubleshoot build-related issues that prevent your app from being shared or deployed.

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 shared or deployed, but you are not explicitly notified. However, when build issues prevent your app from being (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 Share Application Extension dialog, with a pointer to open the build logs.

Typically, most build errors occur when you share your app and must be resolved before you publish it. Here's what to do when you encounter build errors:

- Click Open build logs in the dialog.
- 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.

After resolving the issues, share or deploy your app again.

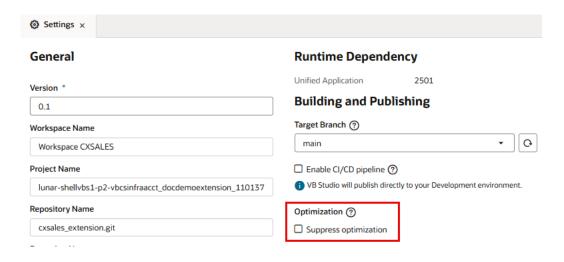
Suppress Optimization

When your extension is shared or deployed, its resources are optimized to reduce load times and network traffic. Optimization is recommended to improve your App UI's performance before it's shared with others, but at times you may need to disable optimization, say to troubleshoot file access issues that occur because of optimizing your extension.

Optimization minifies application stylesheets, HTML, and JSON files, and creates RequireJS bundles in the embedded build server that prepares your extension to be shared or deployed. If you don't want these resources to be optimized, you can turn off optimization before sharing or deploying your extension.

To disable application optimization:

- Click Menu in the upper right corner and select Settings.
- Select Suppress optimization under Optimization in the General tab.



With this setting, optimization is disabled in the build.json file as follows:

```
{
  "optimize": {
    "suppress": true
  }
}
```

Click Open Build Configuration to open this file (located under /extension1/sources/settings in Source view) from within the extension-level Settings editor and confirm that optimization is disabled.

To re-enable optimization, deselect **Suppress optimization** in the Settings editor. Optimization automatically takes effect the next time your extension is shared or deployed. To explicitly specify the resources you want to include and exclude in the optimized resources bundle, see Customize Optimization.

Change the Log Level

When using your browser to debug and resolve issues in your extension, you might want to change the log level to increase or decrease log messages.

You do this by adding <code>vb_logConfig_level</code> as a Local Storage key in your browser's developer tools section. Here's how to do this in Chrome:

- Open Chrome DevTools.
- 2. Click the **Console** tab in the DevTools panel.
- Enter the following code:

```
localStorage.setItem('vb logConfig level', 'loglevel')
```

where *loglevel* can be none (least verbose), info, warning, error, or log (most verbose). For example:

```
localStorage.setItem('vb logConfig level', 'info')
```

4. Refresh your browser (or close and reopen it) to make the new log level take effect and save it to session storage.

5. It's also possible to dynamically set the log level in your browser's current session to temporarily troubleshoot issues. You do this using the <code>sessionStorage.setItem()</code> 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. See Migrate Runtime Dependencies.

- a. Click the **Console** tab in the DevTools panel.
- **b.** Enter the following code in the console:

```
sessionStorage.setItem('ojet.logLevel', 'loglevel')
```

where *loglevel* can be none (least verbose), info, warning, error, or log (most verbose). For example:

```
sessionStorage.setItem('ojet.logLevel', 'error')
```

- c. Refresh your browser (or close and reopen it) to make the new log level take effect.
- d. To reset the log level for your browser's current session, enter sessionStorage.removeItem("ojet.logLevel") in the console, then refresh your browser.

Publish Your Extension

The Publish operation combines Git operations (commit, push, merge) to merge the changes in your local repository branch to the target branch you specify in the remote repository.



If you need to review the relationship between your workspace, your project, and the Oracle Cloud Applications development environment, you may want to watch this

video: Video

There are a few important details to attend to before you publish your extension:

- If you used an Oracle Cloud Application sandbox in this extension, publish it first.
 Publishing the sandbox before merging your extension changes can help avoid potential problems resulting from using two different data models. For more about using and publishing sandboxes, see Sandboxes in Configuring and Extending Applications.
 - It's also a good idea to check and verify your extension's code before you publish. See Debug and Audit Your Code for more.
- Make sure your extension has a suitable name and description. This helps users who may
 want to configure something in your extension later on, as both the extension name and
 description appear in the Dependencies list. To check what's currently set for your
 extension, click Menu in the upper right corner and select Settings.

You specify a target branch in the remote repo when you click **Publish**. The Publish action packages and deploys whatever is in that target branch to your environment's Oracle Cloud Applications instance. For example, let's say you're publishing to the remote repo's main branch. Under the covers, Publish does a commit of your *current* branch to your local main branch, then pushes it to the remote main branch. The changes are then deployed to your Oracle Cloud Applications instance, either *directly* or by kicking off package and deploy jobs in a *CI/CD pipeline*.

Configuration changes you make in your Oracle Cloud Application (through the **Edit Page in Visual Builder Studio** link) are, by default, deployed directly to your target instance. If you prefer, you can change the default by enabling the **CI/CD Pipeline** setting for the target branch you're merging to.

Here's a brief rundown of the two options: Publishing your changes directly to your Oracle Cloud Applications instance allows you to quickly view your changes—but because deployment happens immediately, you won't have a chance to create a merge request as part of the publishing process, so you'll need to do that *before* publishing your changes. Publishing as part of a CI/CD pipeline gives you the flexibility of attaching additional jobs to the out-of-the-box configuration, but you'll need to wait for the entire process to complete before you can see your changes. For details on when and why you may want to use one option over another, see Use CI/CD Pipelines for Deployment in *Administering Visual Builder Studio*.

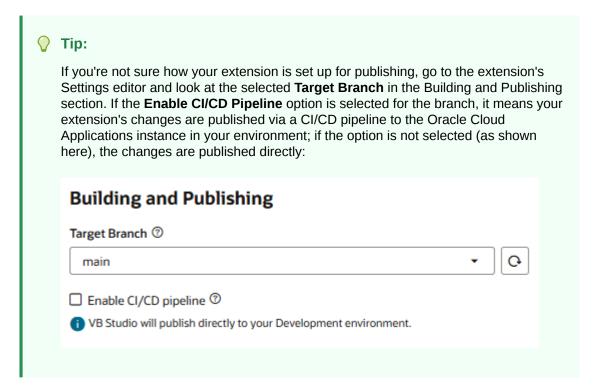
No matter how your changes are deployed, if you want to deploy everything in your extension and you've been working in several different branches, you must commit and push the changes from *all* branches *before* clicking **Publish**. For example, suppose you're working on MyNewAppUI in branch A. You then get word that you need to quickly fix something in ProdAppUI. You create a new branch in your repo (or switch to the branch devoted to ProdAppUI), make the change, then click Publish. Only the changes in that branch are deployed; your changes to MyNewAppUI aren't deployed, unless you explicitly committed them to the desired target branch, like main, then pushed them to the remote repo before clicking **Publish**.

• To ensure your changes load successfully, the target instance must be running the same Oracle Cloud Application release version as your development instance. If you develop an extension on, say, 23D in your Test environment, then want to deploy the extension to your 23C Prod environment, you should wait until your Prod instance has been upgraded to 23D before you deploy the extension. If your instances are out of sync, you'll see this message: "Dependent Extension with Extension id: xxxxand version >=xxxxx doesn't exist". In most cases there shouldn't be more than a two-week gap between pod upgrades. See Oracle Applications Cloud – Fusion Applications Update Policy (Oracle account signin required).

When you're ready to publish your extension:

- See Publish Your Changes Without CI/CD Pipelines if the CI/CD Pipeline setting is disabled for the target branch you're merging to.
- See Publish Your Changes Through CI/CD Pipelines if the CI/CD Pipeline setting is enabled for the target branch you're merging to.





Enable or Disable the CI/CD Pipeline for Publishing

When your extension is published, your changes can be deployed to the Oracle Cloud Applications instance in your Development environment either directly or via continuous integration and delivery (CI/CD) pipelines.

CI/CD pipelines give you the option of deploying an extension to multiple environments as well as automating some lifecycle operations (for example, you can deploy dependent artifacts to the target environment in parallel or delete extensions to clean up the environment). If this extended functionality is not required, you can opt to disable the CI/CD pipeline and simply publish the extension to your environment.

Configuration changes made in your Oracle Cloud Application (through the **Edit Page in Visual Builder Studio** link) are, by default, merged to your project's main branch and then directly deployed to your target instance without a pipeline. You can change both the default target branch to merge to (something other than main), as well as the CI/CD pipeline setting for the target branch. (If you change the target branch and you want to deploy with a pipeline, you'll have to create a pipeline for that branch first.) The default setting takes effect when you create a new workspace in version 25.01.1 or later. Existing extensions continue to use the CI/CD Pipeline setting that was previously set in the extension's Settings editor, until you change it.



Caution:

The CI/CD pipeline is a branch-level setting that impacts everyone who publishes to that branch when collaborating on the extension, including those working on the extension in Express mode. Before you enable or disable the pipeline, make sure it's the optimal configuration for all those who work on the extension. Once you decide on a setting, it's a good idea not to change it unless your requirements change significantly.

- Click Menu in the upper right corner and select Settings.
- 2. Under Building and Publishing, select a default **Target Branch** in your workspace's remote repo. (Click to refresh the list of remote branches, as needed.)
 When others publish to this remote branch, you'll be notified so you can refresh your workspace to pick up their changes. This target branch is also preselected in the Publish dialog when you publish your extension, but you can change it.
- If a pipeline exists for the selected target branch, the Enable CI/CD pipeline checkbox displays.

Select or deselect **Enable CI/CD pipeline**:

- The option is enabled when at least one packaging job is enabled for the target branch. Deselect the option to stop using CI/CD pipelines for publishing.
 - With the CI/CD pipeline disabled, your extension's sources are merged to the target branch of the workspace's Git repository, selected when you click **Publish**, then built and deployed directly to the Oracle Cloud Applications instance within your Development environment.
- The option is disabled when all packaging jobs for the target branch are disabled.
 When you click **Publish**, your extension is merged to the target branch, then built and deployed directly to your Development environment's Oracle Cloud Applications instance.
 - Select the option to start using CI/CD pipelines for publishing. Selecting the option enables *all* pipelines associated with the repository's selected target branch.

As with most other actions, any change you make is logged to the activity stream on the Project Home page.

When you're ready to merge your changes, see Publish Your Changes Through CI/CD Pipelines or Publish Your Changes Without CI/CD Pipelines.

Publish Your Changes Without CI/CD Pipelines

Follow these steps when your extension's **CI/CD Pipeline** setting is disabled for the remote branch you're merging to, or when a pipeline doesn't exist for the branch. With this setting, your changes are deployed immediately to your environment's Oracle Cloud Applications instance.

Note:

Configuration changes made when you click the **Edit Page in Visual Builder Studio** link in your Oracle Cloud Application are, by default, deployed *directly* to your target instance. This default takes effect when you create a new workspace in version 25.01.1 or later. Existing extensions continue to use the CI/CD Pipeline setting that was previously set in the extension's Settings editor, until you change it.

When publishing changes without CI/CD pipelines, your extension's sources are merged to the specified target branch of the workspace's Git repository, then built and deployed directly to the Oracle Cloud Applications instance within your Development environment. Because deployment happens immediately, you won't have a chance to create a merge request as part of the publishing process. So if you want to get your changes reviewed, create a merge request before publishing your changes.

When you're ready to publish your extension:

When you're ready to publish your exten

1. Click **Publish** in the header:





If **Publish** isn't enabled, you're likely using a scratch repository instead of a real Git repo. See Push a Scratch Repository to a Remote Repository to create a remote repo.

- 2. If changes exist, enter a comment in the **Commit Message** area to describe your changes and commit them. A commit groups the changed files and provides a description to identify the group.
- 3. In the **Target Branch** field, select the branch in the remote repo to merge to.

The first branch in this list is always the repository's default branch, main, conveniently followed by the last 10 branches you created merge requests for (if any). Any remaining branches follow in alphabetical order. This flexibility lets you more easily publish and test your new feature development work, if you're not yet ready to merge those changes into your repository's main branch.

4. The New Working Branch Name field displays with an automatically populated new name for your working branch. You can accept the default, but it's recommended to change to a name that's more meaningful.

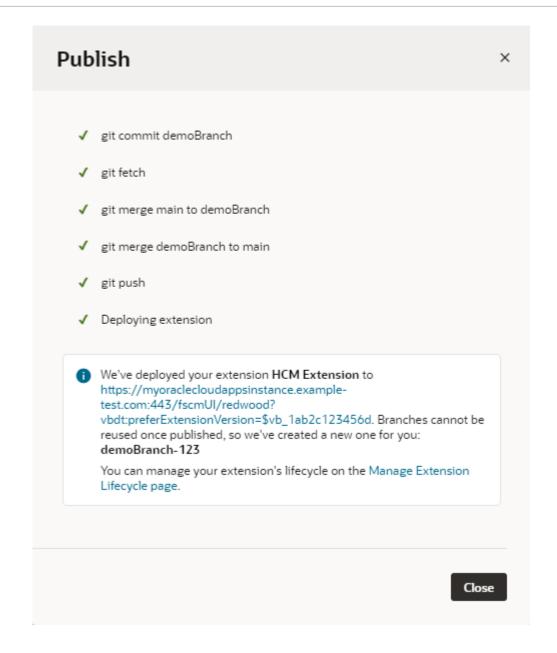
Once you publish a branch, you can no longer use it (or your local copy of it). Instead, VB Studio automatically creates a new working branch using the value in this field, and switches your workspace to it for any future changes.

Click Publish.

Your changes are merged from your working branch to the selected target branch, then immediately deployed to the Development environment associated with your workspace.

Click the link in the Publish dialog to see your published changes:





Note:

Make sure you copy and paste the deployment URL to your clipboard before you click **Close**. You won't have access to the URL after the Publish dialog is closed. Also, users of your Oracle Cloud Application will have to sign out of the app, then sign back in again to be sure they're seeing the latest.

After testing the changes deployed to your Development environment, you might want to deploy it to other Oracle Cloud Applications instances from the Manage Extension Lifecycle page.

Publish Your Changes Through CI/CD Pipelines

Follow these steps when your extension's **CI/CD Pipeline** setting is enabled for the remote branch you're merging to. With this setting, your changes are deployed via CI/CD pipelines to your environment's Oracle Cloud Applications instance.

To successfully deploy your extension via a CI/CD pipeline:

You must be authorized to run the build job (if the job is private).
 Project owners can configure protection settings for build jobs so that only authorized project members can run the jobs. If you can't run a build job, ask the project owner or an administrator to add you to the build job's list of authorized users. If you're the project owner, check the job's protection settings. See Configure Job Protection Settings in Administering Visual Builder Studio.

If you can't access a project, use the **Request Membership** option in the project's **Update project** menu on the Organization page to request access. A project owner can approve your membership so that you can access the project, and authorize you to run build jobs. For more, see Request Membership in a Project You Can't Access in *Using Visual Builder Studio*.

- The pipeline's deployment job (*gitRepoName-Deploy*) must be configured to deploy to the Oracle Cloud Applications instance in your environment.
 - If the deployment job is set up for OAuth (recommended), authorization must be provided for OAuth tokens to be created with the credentials of a user who has permissions to connect and deploy to the Oracle Cloud Applications instance. You won't be able to publish without this authorization. It is recommended that you authorize your connection during initial job configuration—though you'll be prompted to authorize missing or expired OAuth tokens when you click **Publish**.
 - If your deployment job uses Basic authentication, credentials of a user who has permissions to connect and deploy to the Oracle Cloud Applications instance must be provided. The credentials must be Oracle Cloud Application credentials, as opposed to Visual Builder Studio or SSO credentials. If the project owner hasn't provided this information, you'll be prompted to provide it when you click Publish. If your credentials don't allow you to deploy to the Oracle Cloud Applications instance, talk to the project owner or an administrator to get credentials that you can use.

See Optional Administrator Tasks in *Administering Visual Builder Studio* for more about setting up projects to publish extensions.

When you're ready to publish your extension:

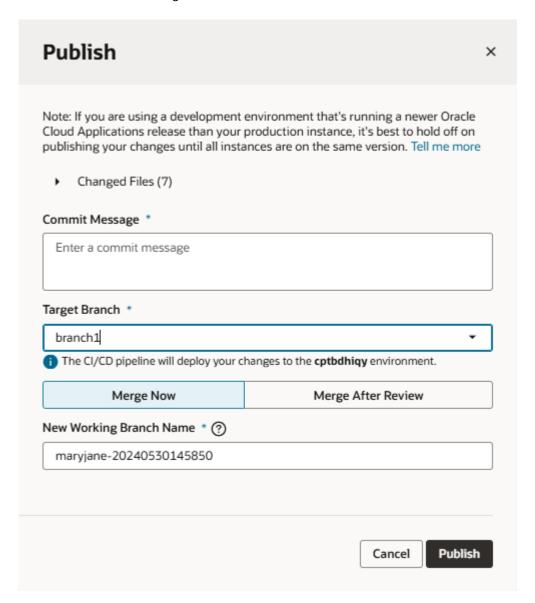
1. Click **Publish** in the header:



Note:

If **Publish** isn't enabled, you're likely using a scratch repository instead of a real Git repo. See Push a Scratch Repository to a Remote Repository to create a remote repo.

The Publish operation combines Git operations (commit, push, merge) to merge the changes in your local repository branch into the target branch you select. Your screen should now look something like this:



- 2. If changes exist, enter a comment in the **Commit Message** area to describe your changes and commit them. A commit groups the changed files and provides a description to identify the group.
- 3. In the **Target Branch** field, select the branch in the remote repo to merge to.
 - The first branch in this list is always the repository's default branch, main, conveniently followed by the last 10 branches you created merge requests for (if any). Any remaining branches follow in alphabetical order. This flexibility lets you more easily publish and test your new feature development work, if you're not yet ready to merge those changes into your repository's main branch.
- 4. If you land on the **Merge Now** tab, the **New Working Branch Name** field displays with an automatically populated new name for your working branch. You can accept the default, but it's recommended to change to a name that's more meaningful.

Once you publish a branch, you can no longer use it (or your local copy of it). Instead, VB Studio automatically creates a new working branch using the value in this field, and switches your workspace to it for any future changes.

5. If you'd rather get someone to review your changes via a merge request, click the Merge After Review tab. After entering a comment summarizing your changes, you can then specify other details such as reviewers and linked issues as needed. You can also enter a description of all your changes (not just this commit) in the Merge Request Description field, using the convention established for the template (Markdown, Confluence, or Textile). If you need help, use the link to the associated reference.

Note that the **New Working Branch Name** field doesn't display because, if a merge request is created, VB Studio won't automatically create a new working branch and switch your workspace to it. After the merge request is approved and your extension is published, however, you should manually switch the workspace to a new working branch.

Note:

If you see only the option to commit and merge your changes via a merge request (that is, there's no **Merge Now** tab), that means the project owner or an administrator has set things up so that all changes must always be reviewed and approved via a merge request before they are merged to the target branch. In this case, summarize your changes in the Publish dialog, add at least one reviewer (if no one is specified), and specify the related issue in order to create a merge request. (Once the request is created, you can look for it on the Merge Requests page. For details on how to work with merge requests, see Review Source Code with Merge Requests in *Using Visual Builder Studio*.)

If a merge request is required but a CI/CD pipeline hasn't yet been set up, the dialog will display a message indicating that the Publish action will only merge your changes; to fully deploy your extension, you'll need to first set up a pipeline for the remote branch. See Deploy Changes from a Different Branch in Administering Visual Builder Studio.

6. Click Publish.



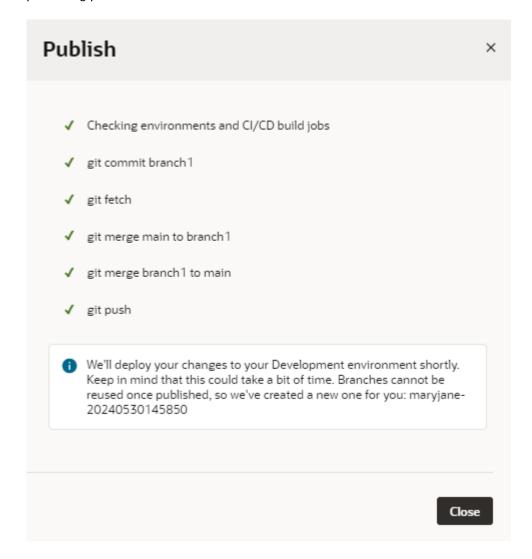
Tip:

Use the details in the Publish dialog to take note of the environment that your changes will be published to, which is typically the deployment target specified in the pipeline that's enabled for the selected target branch.

- 7. If the deployment job in your CI/CD pipeline is missing the credentials required to connect and deploy to your environment's Oracle Cloud Applications instance, you'll be prompted to complete some additional steps:
 - If you see a message that authorization is required, it means the deployment job is set up for OAuth and requires authentication for the OAuth connection to the target Oracle Cloud Applications instance:
 - a. Click OK.
 - b. If prompted, enter the credentials of a user who can access the target Oracle Cloud Applications instance.

- If you land on the Authorize Jobs page, your Deployment job may have missing or expired OAuth tokens. Click **Authorize**, enter the credentials of a user who can access the target Oracle Cloud Applications instance, and click **Authorize**.
- c. Once authorization is complete, click **Publish** again in your workspace's Publish dialog.
- If you see a message about missing deployment credentials, it means the deployment
 job uses Basic authentication and requires the credentials of a user who has
 permissions to access the target Oracle Cloud Applications instance. Enter valid
 credentials and click Add Credentials and Continue.

You'll now see the Git commands that are automatically performed for you as part of the publishing process:



You should then see notifications in the bottom right corner of your workspace, indicating that the publish process has started. (If you miss these notifications or want to track progress, look for them under **Notifications** in the header.) If a build job fails, you'll see something similar to this message and will need to take action to resolve the issue (click **Open job** to view the job's details and take action):





Your project is probably set up so that each time changes are merged into the main branch, the extension artifacts are built from the repository and automatically deployed to the Oracle Cloud Application's Development environment, or to a Test environment for further tests. When ready, you can deploy your extension to other Oracle Cloud Applications instances as well, either by modifying your project's build pipeline or by using the Manage Extension Lifecycle page.

IMPORTANT: If your extension contains configurations for an Oracle Cloud Application, users of that application will have to sign out of the app, then sign back in again to be certain they're seeing the latest.

Deploy to Test and Production Instances

If you're a developer who's fairly comfortable with Git and the development lifecycle, you'll probably want to set things up so that you have control of moving your extension from the development phase into test, and finally to production.

When a project is first created and configured, it's typically set up so that changes committed to the main branch automatically kick off a pipeline that packages up the extension and deploys it to whichever Oracle Cloud Applications instance was named as the project's Development instance. You can modify this pipeline so that your extensions are deployed to other non-production Oracle Cloud Applications instances as well, like a test pod or another Development environment. This involves the following steps:

- Create a VB Studio environment for <u>each</u> Oracle Cloud Applications instance to which you want to deploy.
- 2. Create a deploy job for each instance you defined in step 1.
- 3. Set up the pipeline to run the package and deploy jobs you created in step 2. In most cases you'll want to have this pipeline automatically triggered when anyone commits to main, but you have the option to run the pipeline manually. (As an individual developer working alone or with just a few others on a project, it's unlikely that this option would appeal to you, but it does exist.)

Note:

If your Oracle Cloud Applications instance is not in the same identity domain as your VB Studio instance, you can use OAuth tokens to access the instance. All you need to do is provide credentials of a user that can access the instance, when prompted. If needed, talk to your administrator to get Oracle Cloud Application credentials that you can use. See Create or Renew OAuth Tokens to Deploy Your Extension.

When you're ready to move your extension to production, the process is similar, with a few additional steps:

- 1. Create a VB Studio environment for the Oracle Cloud Applications production instance.
- 2. Create a new branch in your project's Git repo for changes that are ready for production. After creating this branch, any changes pushed to the main branch won't automatically be added to the production branch. Instead, you must create a merge request or manually push the changes to the production branch, as a means of protecting the branch against untested or unwanted additions.
- 3. Create a production packaging build job, which generates an extension artifact that ready to deploy to the mainline.
- Create the production deployment build job, which deploys the artifact to the Oracle Cloud Application's production instance.
- 5. Set up the production build pipeline.

Step-by-step instructions for both of these processes are available here:

- Set up the Project to Deploy to Other Development and Test Instances
- Set Up the Project to Deploy to Production



Tip:

Just like you have the option to bypass the pipeline when initially publishing your extension, you can also bypass the pipeline when deploying it to other Oracle Cloud Applications instances. To do this, use the Manage Extension Lifecycle page.

Create or Renew OAuth Tokens to Deploy Your Extension

When the Oracle Cloud Applications instance in your environment is not in the same identity domain as your VB Studio instance, you use OAuth tokens to securely access the Oracle Cloud Applications instance and deploy your extension.

VB Studio leverages three-legged OAuth 2.0 flows to secure programmatic access to your Oracle Cloud Applications instance. OAuth tokens eliminate the use of passwords in service-to-service REST interactions and centralize trust management between clients and servers.

OAuth tokens for an instance in a different identity domain can be created while adding the
instance to an environment: simply add the instance with the Authorization Type set to
OAuth 2.0. See Add an Oracle Cloud Applications instance to your environment.

Before any OAuth tokens can be created, a one-time authorization must be provided to handle OAuth requests for your environment's Oracle Cloud Applications instance. VB Studio can automatically detect an instance being added from a different identity domain and will prompt you for authorization. Click **Authorize**, when prompted, then sign in with credentials to access the Oracle Cloud Applications instance.



Note:

It is recommended that you authorize your OAuth connection during initial configuration. If you skip this step, you won't be able to publish your changes from the Designer until the required authorization is provided.

In addition to OAuth set up from the Environments page, OAuth can be configured in build jobs that deploy artifacts to your environment's instance, for example, in the deploy build job used by the Publish action in the Designer to deploy your extension. To configure this

- build job for OAuth, see Create a Deployment Build Job. OAuth is also supported in the delete build job used by an extension.
- 2. OAuth tokens (access and refresh) are cycled during regular use. A refresh token is used to obtain an access token whenever a user accesses the target instance. This refresh token is typically valid for seven days. (The token expiration time is set in the IDCS resource app and may be different based on your security requirements.) If the user authenticates with the target instance within the seven-day period, the active refresh token generates a new access token and a new refresh token. This cycle continues indefinitely as long as the refresh token stays valid. If the refresh token expires during extended periods of inactivity (say, when you're away on vacation), you'll need to renew the access and refresh tokens.
 - To renew OAuth tokens from the Environments page, click Actions *** and select Renew OAuth Access.
 - To renew OAuth tokens from the Builds page, locate your deploy or delete build job, then on the Steps tab, click Renew Authorization. You can also run the job manually, so you'll be prompted to authorize any expired OAuth tokens.



Service administrators can control the OAuth access or refresh token's expiration in the IDCS resource app. To change this for Oracle Cloud Applications, you set the value in the Fusion Applications Cloud Service resource app under Oracle Cloud Services in your identity domain's resource app. See Edit High-Level Information for Oracle Applications.

Impact of P2T on Extensions

When it comes time to test your extensions, or new versions of your extensions, you might want to use the Oracle Production-to-Test (P2T) refresh process to copy data from your Production environment (the source environment) to a Test or Development environment (the target environment). This allows you to thoroughly test your extensions without disrupting the Production environment. (You can also use the process to move data from one Test or Development environment to another Test or Development environment (T2T)).

The P2T process copies the contents of the database in the Production environment to the database in the target environment. Here's what happens during the process:

- All extensions on the target environment are replaced by the extensions from your Production environment. If the Production environment has older versions of extensions, they are copied over and effectively replace the versions on the target environment, even if the versions on your target environment are newer.
- Similarly, all metadata related to your extensions stored in the target environment's database is overwritten by the metadata info from the Production instance's database.

The P2T/T2T process copies only source environment's **database content**, which means that none of these are affected:

- The source for your extension, which lives in your project's Git repo
- Your CI/CD build jobs and pipelines
- The passwords to access the Test or Development environment.



During the P2T process, you won't be able to access VB Studio on the target TEST or DEV instance for up to two days. However, you can use the instance of VB Studio associated with a different DEV or TEST instance in your environment family, assuming your user account has been added to the project you want to work in.

Deploy an Extension From Your Local System

After an extension is packaged as a .vx archive, you can use a REST request to install the extension on an Oracle Cloud Application environment instead of using a Visual Builder Studio deploy build job.

For example, you might want to deploy an extension to an Oracle Cloud Application production environment that has restricted IP access, so the environment is not accessible from Visual Builder Studio. To get around this, you can use curl to install the extension from an IP address that's able to access the environment. To install the extension, you'll need a valid .vx extension archive and the URL and credentials for the Cloud Application environment.

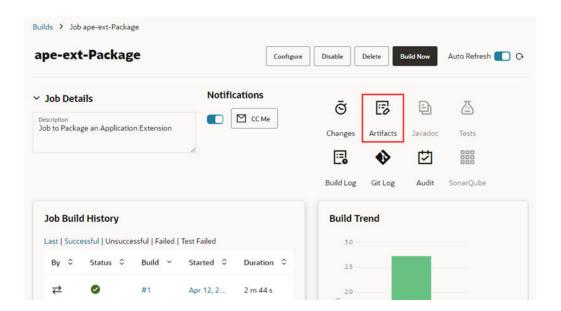
The Package build job for your extension in VB Studio will build a valid extension archive for you. After running the build job, you can download the archive to your local system.



You can also package your .vx extension archive locally using Grunt.

To install an extension from your local system:

- Make sure you have curl installed on your local system, and that the system can access the Cloud Application environment.
- 2. Download the extension archive to your local system.
 - a. On the project's Builds tab, open the Package build job for the extension.
 You can use the same Package build job used to build the extension for testing.
 - b. Click Artifacts to open the Artifacts Archived page.





c. Click the extension archive to automatically download it to the default location on your local system.

The extension will have .vx appended to the name as the file type (extension.vx).



If you don't see an extension archive, click **Build Now** to run the build job.

- 3. Use curl to install the extension on the Cloud Application.
 - a. Open your command terminal.
 - **b.** In the terminal, navigate to the location of the extension archive on your local system.
 - **c.** Type the curl request at the command prompt to install the extension.

In the curl request you'll need to provide:

- The USERNAME and PASSWORD for the Oracle Cloud Application instance.
- The name of the extension archive (for example, EXTENSION NAME.vx).
- The path to the v2/extensions directory of your Cloud Application, which will include the URL and port (INSTANCE_URL: PORT/fscmRestApi/vx/v2/extensions).

Your curl request will look similar to this:

```
curl -v --noproxy '*' -X POST -L -H Content-Type: application/octet-
stream -H 'X-OAM-AUTHN-HINT: 0'
  -u USERNAME: PASSWORD --data-binary @EXTENSION_NAME.vx
http://INSTANCE_URL: PORT/fscmRestApi/vx/v2/extensions
```

The curl request contains a POST method to add the extension archive to the extensions directory.

After installing the extension, you can use the following curl request to see a list of installed extensions:

4. (Optional) To delete an extension using curl:

To delete a specific version of an extension:



To delete all versions of an extension:

The extension is automatically installed and registered when it's added to the directory.



Extensions deployed from a local system using curl will not be in the list of deployed application extensions in the project's Environments tab.

Specify a Different Version for the Extension

The packaging job generates the extension's build artifact from the source files in the Git repository's main branch.

The extension's version is defined in the extension.json file. If you want the build artifact to use this version, leave the packaging job as it is.

To specify a different version for the extension, without explicitly modifying extension.json, follow these steps:

- In the VB Studio left navigator, click Build.
- 2. In the **Jobs** tab, click the packaging job for the extension.
- Click Configure.
- Click Configure (with the hammer).
- **5.** Specify the new version in the **Extension Version** field.

When the build runs, VB Studio overwrites the version defined in the extension.json file with the version specified here.

Click Save.

What If Dev and Prod Instances Aren't On the Same Release?

When publishing an extension, your development and production instances must be on the same Oracle Cloud Applications release version. If they're not, you may see this message: "Dependent Extension with Extension id: xxxxand version >=xxxxx doesn't exist".

Here are some recommendations to follow when your instances are out of synch:

Deploying to Production

When the Oracle Cloud Applications instances in your development and production environments are running different releases, we recommend that you wait until they are on the same release to deploy your changes. If you've created an extension on a development environment that's running a different release than your production environment, **do not** deploy the extension to production until your production environment is updated to match your development environment.



Publishing to Development

If you've already deployed a prior version of your extension to production, **do not** publish your extension to deploy to the development environment that's running a newer release. Publishing changes when your development instance is on a newer release than your production instance (say, when you're developing an extension on 24D and your production environment is on 24C) could merge changes that may not work with the release that your production instance is on. Even if you don't plan to promote those changes to production, you—or someone in your team—may need to update your extension in production. By publishing your changes, you run the risk of including changes made on the newer release with the updates meant for production. For this reason, we recommend that you wait to publish your changes until your production instance is running the same release as your development instance. In most cases, there shouldn't be more than a two-week gap between upgrades.

Check Your Instance's Release Version

To check the release version that your development and production instances are running on:

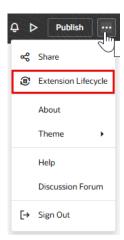
- Go to the Environments page and look for the environment's Oracle Cloud Applications instance in the Service Instances tab.
- 2. Under Details, click the instance's **Service URI** to access the instance in a new tab.
- 3. From the **Settings & Actions** menu, click **About this Application** to view the release version the instance is on.

View Your Deployments

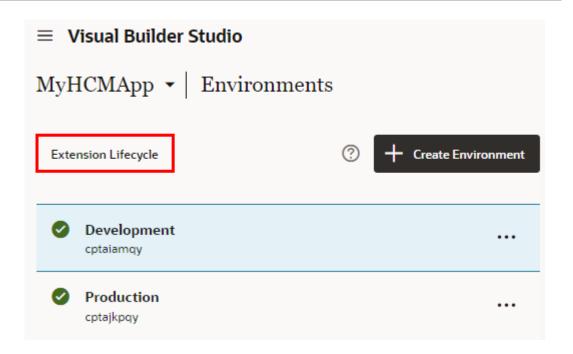
After you've published your extension, you can check the Manage Extension Lifecycle page to see if it's been deployed to your development environment.

To view your deployed extensions:

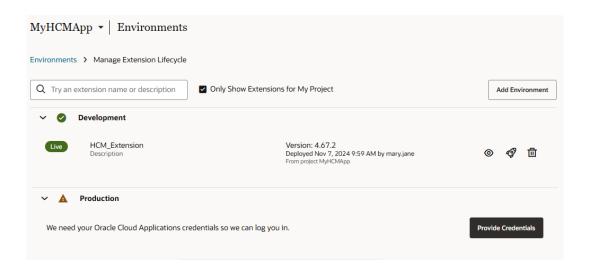
From the menu in the header, click Extension Lifecycle.



Or, in the VB Studio left navigator, click **Environments** to open your project's Environments page, then click **Extension Lifecycle**.



You should now see the Manage Extension Lifecycle page.



- Expand each environment to view a list of the extensions already deployed. Each entry includes the extension name plus additional details, such as extension version, deployment date, who deployed the extension, and project name.
- 3. Select to view even more details, such as the extension's extension ID, App UIs, dependencies, and version history.

Extension Details

Name

HCM_Extension

Ιd

HCM Extension 1

Version

4.67.2

Deployed

Wed at 12:51 PM

Deployed By mary.jane

App UIs

☐ HCM App 🔼

Dependencies

oracle_hcm_benefitscommonUl >=2501.0.8

oracle_hcm_employmentcommonUI >=2410.0.2

Version History

Nov 26, 2024 12:36 AM

Version: 4.67

Deployed by mary.jane

• If you added a new App UI to your extension, you can open it by clicking oxdiv .

All App UIs open in their own browser tab, with a dedicated URL. By default, this URL is in the form https://UnifiedAppName/redwood/AppUIName. However, if you specified an App UI ID when you created your App UI, that ID will be substituted for yourAppuiName instead.

An Open icon is not provided for an App UI that you configured, nor are configured App UIs listed here explicitly. In other words, there is no way to tell whether <code>oracle_hcm_timeUI</code> has been configured by this extension, only that it has been added as a dependency.



Dependencies are listed by their extension IDs. (The Unified Application, which is a dependency for all App UIs, is not listed.)

Extension IDs are assigned in the **New Application Extension** dialog when the extension was created (either site_extensionName by default or user-specified), or assigned automatically if you used the **Edit Page in Visual Builder Studio** option to jump over to Visual Builder Studio from an Oracle Cloud Application.

The version number of the dependency appears after the extension ID.

4. Use the **Only Show Extensions for My Project** checkbox to toggle between viewing extensions deployed only from the current project, or from all projects.

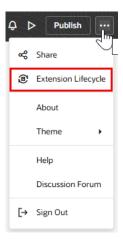
In addition to *viewing* your deployed extensions, you can also use the Manage Extension Lifecycle page to centrally *manage* them, as well. For example, you can deploy a published extension to a new Oracle Cloud Applications instance, and delete them when no longer needed. To learn more about what you can do on this page, see Manage Your Published Extensions.

Manage Your Published Extensions

When working with multiple environments (say, Development1, Development2, Test, and Production), you can use the Manage Extension Lifecycle page to centrally manage published extensions across environments (and across projects). Use this page to deploy a published extension to a new Oracle Cloud Applications instance, and delete them when no longer needed.

Before you can deploy an extension to another instance from this page, make sure the extension is already published to your default Oracle Cloud Applications instance. You must also have a valid user name and password for the target instance you want to deploy to.

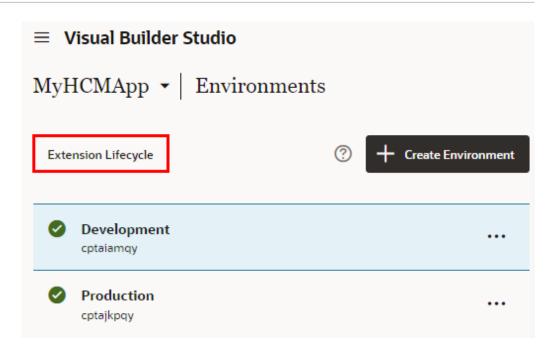
- Access the Manage Extension Lifecycle page:
 - From the menu in the Designer's header, click Extension Lifecycle.



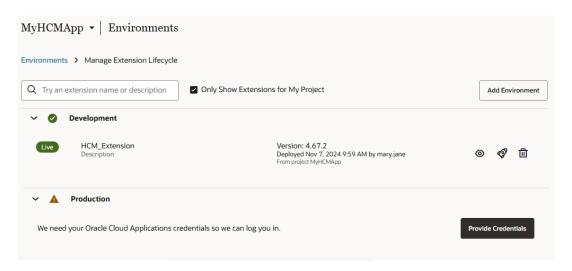
Or, from the VB Studio left navigator, click Environments, then click Extension
 Lifecycle. (If you're in the Designer, click at the top left of the header to get to the VB Studio navigator.)

You won't see the **Extension Lifecycle** button if you haven't yet added an Oracle Cloud Applications instance to your environment.





You should now see the Manage Extension Lifecycle page, which lists all the environments that are available for you to deploy to.



You can work with an environment on this page as long as the authentication method for its Oracle Cloud Applications instance is OAuth. If there's an environment listed whose authentication method isn't OAuth, then you'll first need to add a new environment with the same base URL, but using OAuth as the authentication method, before you can work with it.



If you see a **Provide Credentials** button next to an environment, you must first sign in before you can deploy your changes to it.

Expand each environment to view a list of the extensions already deployed. Each entry includes the extension name plus additional details, such as extension version, deployment date, who deployed the extension, and project name.

Use the Only Show Extensions for My Project checkbox to toggle between viewing extensions deployed only from the current project, or from all projects.

- To deploy an extension to another Oracle Cloud Applications instance:
 - a. Find the extension that you want to deploy and click



Choose the environment that points to the instance you want to deploy to.



Your project administrator should have already created any additional environments you might need to deploy to. If you don't see an expected Oracle Cloud Applications instance or if that instance is unavailable, check with your project administrator or add a new environment right from this page (see step 3).

c. Click Deploy.

Deployments made from this page bypass any CI/CD pipelines previously set up.

IMPORTANT: To see the newly published extension, your users will have to sign out of their Oracle Cloud Applications instance, then sign back in again to be certain they're seeing the latest.

- 3. If you'd like to deploy to an Oracle Cloud Applications instance but it's not listed on this page, you can add it.
 - Click Add Environment.
 - b. In the Add Environment panel, enter basic environment details, such as the environment's base URL, environment name, and user name and password for the target Oracle Cloud Applications instance.

You can add any environment as long as the authentication method for its Oracle Cloud Applications instance is OAuth.

c. Click Add.

You also have the option to add an environment from the Environment's page. See Add the Oracle Cloud Application's Production Instance to an Environment.

To delete an extension from an Oracle Cloud Applications instance, click Deleting an extension restores your application to its original state.



Roll Back a Published Extension

Sometimes after publishing your changes to an Oracle Cloud Applications instance, you might need to roll them back. When you "roll back" an extension, what you're actually doing is redeploying a previous extension version. To do this, you'll need to locate the previous version



in VB Studio, and then move it to your extension's remote repository so that you can redeploy it.

It's always best to redeploy the previous version's actual build artifact that was originally packaged and deployed. If you have access to this build artifact in Advanced mode and you'd like to redeploy it, see this blog.

But, if build artifacts from previous extension versions aren't available, here's what you can do instead:

- Make sure you're working in VB Studio's Advanced mode.
- 2. In the Designer, switch your branch to the previous extension version.



Tip:

Look for the branches named <user name>-<timestamp>; these are the branches that are created each time you publish an extension (if you didn't rename them).

When switching, select the **Create new branch based on parent branch** checkbox.

- 3. Push the new branch to your remote repository.
- From the VB Studio left navigator, click **Builds**, then configure your extension's packaging job with a new "Branch" parameter.
 - Set the parameter's default value to main so that, when you use the **Publish** button in VB Studio, everything still functions as usual.
- 5. Run the packaging job manually, entering your newly created branch (instead of main) in the parameter you just added.

The packaging job will run and create a build artifact using the specified branch. Once the job finishes, it will automatically trigger the deploy job as usual.

For step-by-step instructions, see this blog.

Build and Package Your Extension Manually

The build and package pipelines provide everything you need to package and deploy your extensions. However, you can also use Grunt to build and package an extension directly on your local machine, and then install the extension on the Oracle Cloud Application.

To build an extension locally, you'll need to save the extension's sources to your local system. You can get the sources by:

- Cloning or downloading the Git repository containing the sources, or
- Exporting the extension from Visual Builder and extracting it to your local system.

For instructions on getting the sources, see Download Your Application's Sources in *Building Responsive Applications with Visual Builder Studio*

The root folder of your extension's sources includes two resource files that Grunt uses when building the extension:



File	Description
Gruntfile.js	Contains a basic Grunt script for building the extension that can be modified to add custom build tasks, and to configure built-in tasks. See Customize Your Grunt Build Process in <i>Building Responsive Applications with Visual Builder Studio</i> .
package.json	Declares the dependency and specifies the URL reference to the <code>grunt-vb-build</code> and <code>grunt-vb-audit</code> NPM packages that provides the Grunt tasks used to build and audit extensions. VB Studio automatically updates the package version and URL whenever Oracle publishes a new version of the package.

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

Step	Description
Process the sources	This step consists of several important processes. The most important is "metadata processing", when the extension sources are transformed into a deployable form.
	You run the <code>vb-process-local</code> task to process the sources of the extension that you downloaded to your local system. This task creates an output directory (./build/processed) with built assets that can be consumed by the <code>vb-package</code> and <code>vb-deploy</code> tasks.
Optimize the processed sources	This step consists of a number of parts, including optimizing images, optimizing styles and creating required module bundles.
	To optimize the processed sources, run the <code>vb-package</code> task. The task generates the <code>./build/optimized</code> directory with the optimized sources.
Install the extension artifact	This step consists of installing the extension archive on the Oracle Cloud Application. You can install the archive by either:
	 sending a REST request to the Oracle Cloud Application. See Deploy an Extension From Your Local System.
	 running the Grunt vb-deploy task. See Deploy Your Extension Using Grunt.

Build Your Extension Using Grunt

To build an extension locally, you need to install Node.js and its package manager (npm) on your local system. After Node is installed and you get your sources, you can run Grunt tasks to build the extension.

To build your extension locally:

- 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 extension.

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



4. Enter the task names in the command-line interface to process the sources and package the extension. The following example shows how you execute these tasks along with some supported parameters:

```
# First build extension sources
./node_modules/.bin/grunt vb-process-local
# Package the extension sources. This task in turn executes
# vb-optimize and vb-manifest
./node modules/.bin/grunt vb-package
```

To view the full list of supported parameters for each task, see Grunt Tasks to Build Your Visual Application in *Building Responsive Applications with Visual Builder Studio*.

When these Grunt tasks finish, you can install the resulting packaged extension artifact on your Oracle Cloud Application. To install the extension, you can install the extension archive using REST or deploy the extension archive using the Grunt vb-deploy task.

Deploy Your Extension Using Grunt

The Grunt vb-deploy task deploys a packaged extension archive to an Oracle Cloud Application.

Before you deploy the extension, you must build the extension archive using the <code>vb-process-local</code> and <code>vb-package</code> tasks. The <code>vb-package</code> generates an archive in the <code>build/processed/*</code> and <code>build/optimized/*</code> directories. The <code>vb-deploy</code> task deploys the generated archive.

To run the vb-deploy task:

• Enter the task and task options in the command-line interface:

```
# Deploy extension archive
./node_modules/.bin/grunt vb-deploy
--url=<Application Cloud URL> --username=<username> --password=<password>
```

where:

- --url=<Application Cloud URL> is the URL of the base Oracle Cloud Application.
- --username=<username> is the username to access the Oracle Cloud Application extension manager.
- --password=<password> is the password to access the Oracle Cloud Application extension manager.

A password specified via the <code>--password</code> 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 <code>--password</code> option, especially in VB Studio jobs where the password is provided via a job variable. For example: <code>grunt vb-deploy '--password=password'</code>

After running the task, you can open the Environments page in VB Studio and View Your Deployments.

Work With Translations

As you develop your extension, at some point you'll want to consider whether you need to *internationalize* and/or *localize* the App UIs you plan to deliver. *Internationalizing* refers to translating them into your required languages, while *localizing* refers to customizing them for a specific locale.

Localizing might include, for example, modifying the date format or the currency symbol for the intended location.

To ensure the App UIs you are building are translated into the languages you need, you'll want to save the text associated with any new UI components to a *translation resource file*. See Associate a Translation Key with a UI Component.

When you're finished developing your App UI, you can download the available translation bundles and send them for translation. See Download and Upload Translation Bundles.

You might also want to customize an App UI to suit your business needs. For example, you might find that your company, local business environment, or locale uses different terms and spellings than those provided by the original App UI. For example, the App UI you're configuring may use the term, "customer", but your company prefers the term "client". Again, you can override the text values in the translation resources to reflect these preferences. See Override a Translation Key Value.

What are Translation Resources?

To ease with translation, all strings in an App UI—such as headings, labels, and messages—can be stored in a separate external file, rather than hard-coded in the App UI. This means you can translate the App UI by simply downloading this file, translating it, and uploading a newly translated file.

During runtime, the App UI uses the language file corresponding to the user's browser language setting unless you set a different locale for it programmatically.

Translation Bundles and Files

A *translation bundle* is a set of translation files that has one file for each supported language. Typically, each file contains all the text strings in an App UI in the supported language.

Your extension may include one or more translation bundles. Let's say you are extending an Oracle Cloud App, like Digital Sales. Your extension will contain one translation bundle for Digital Sales, as well as one for the Unified Application, which is the basis for all Oracle Cloud Applications. If you also created your own App UI within this same extension, you'd have yet a third translation bundle, this time for the text strings for your new UI components.

Translatable Strings

A translation file is a JSON file that stores the App UI's *translatable strings* for a given language. These strings are stored in key-value pairs, where the *translation key* is a unique ID in the bundle for the translatable string. It is this bundle/key that you use to bind a UI component's text property value to the string in the translation file.

Let's say you have a number of fields and lists for entering or selecting the city for an employee or site. Rather than setting each component's text property to "City", you can add the string to the JSON file and instead set the component's text property to reference the string's key.

Here's an example of a key-value pair in a JSON file for the string, "City":

```
"siteCity": "City",
"@siteCity": {
   "description": "The name of the city where the office or employee is located",
}
```

In this example, the key is siteCity and the translatable string is City. The entry also includes a description field that helps your translators understand the context for the translatable string. For example, without a description, your translators may have difficulty telling if "Title" refers to personal title, such as "Ms" or "Mr", or to a professional title, such as "Senior Software Developer" or "Director of Marketing". A brief description here can remove any uncertainty.

You can reuse a translation key in multiple places and, if required, modify the key value to update all instances at once. Using our previous example, you might use the translation key siteCity on a Contact page and on the employee Profile page. If you later decide to use "Municipality" instead of "City", you can change the value of the key to update what gets displayed on the Contact and Profile pages.

Let's take a look at how this all fits together in practice. For example, you might decide to include a text field for the employee's city on your Profile page. Initially, you set the "label-hint" property to "City".

If you look at the code for this page, you'll see that the input text field includes a "label-hint" property set to "City":

```
<oj-input-text label-hint="City" class="oj-flex-item oj-sm-12 oj-md-6"></oj-
input-text>
```

If you decide to use the siteCity translation key for the field's "label-hint" property, you'll get this code instead:

```
<oj-input-text label-hint="[[$translations.extensionBundle.siteCity()]]"
class="oj-flex-item oj-sm-12 oj-md-6"></oj-input-text>
```

In this case, \$translations.extensionBundle.siteCity() is a JavaScript function where:

- \$translations indicates that this refers to a translation;
- extensionBundle is the bundle where the key is stored;
- siteCity is the translation key; and
- () is the arguments list you can use to pass parameters to the function. In this case, the arguments list is empty.

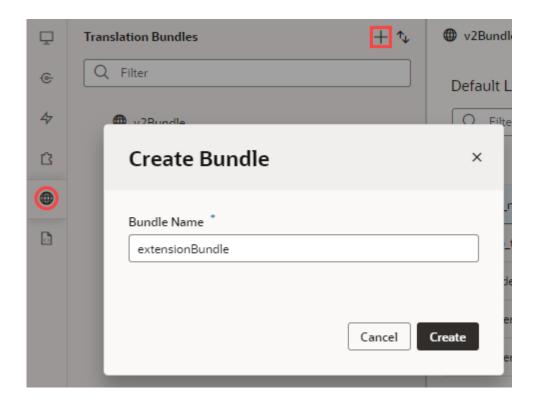


Create a Translation Bundle

If you create an App UI in your extension, you'll likely want to create a translation bundle for it, so that you can store any new translatable strings that aren't already in one of the existing translation bundles.

You create a translation bundle from the Translations pane. This pane displays a list of the bundles for the App UI you are creating as well as any for the App UI you are extending, grouped by pillar. Extensions not yet assigned a pillar are listed under the Other heading. You can open this pane using the Translations icon () in the Navigator.

To create a translation bundle, click the Create Bundle icon (+), type a name for the new bundle, then click **Create**.



Note:

The bundle name can include hyphens and underscores but can't include spaces or special characters such as @ or #.



Create Translation Keys

As you develop your App UI, you'll need to enter text for any new headings, labels, and messages you add. To make sure these text strings are translated, you'll need to create a translation key for each new translatable string.

You can do this in one of two ways. You can open the New String popup from your App UI translation bundle. Or, you can create a new translation key from the Properties pane for a new UI component and bind it to the component. In either case, you'll provide a key value, the translatable string, and a description. The description provides your translators with additional context.

You can also provide custom metadata for a translation key as well as placeholder metadata for strings that include expressions. Access these fields from the Properties pane for a translation key when you select it from your App UI's translation bundle.

Strings can be either static values or expressions written in International Components for Unicode (ICU) format. If you want to use an expression in a translatable string, see Use an Expression in a Translatable String.

Once you've added a string to your translation bundle, you can associate it with a UI component using the Translatable String popup available from the component's Properties pane.

If the string you want to use has already been saved, you don't need to add it here. You can just search for and select it using the Translatable String popup when you add the UI component.

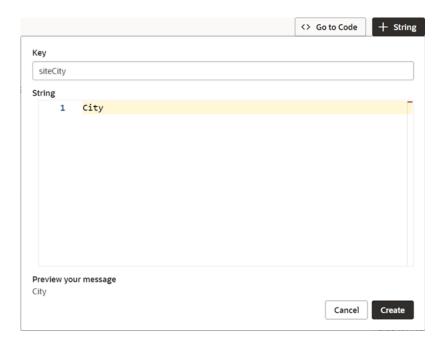


This task describes how to create a translation key from a translation bundle. For the steps to create a translation key from a component's Properties pane, see Associate a Translation Key with a UI Component.

To create a key for a translatable string from your translation bundle:

- Open the Translations pane using the Translations icon (⊕) in the Navigator.
 - The Translation pane displays a list of the bundles for the App UI you are creating as well as any for the App UI you are extending, grouped by pillar. Extensions not yet assigned a pillar are listed under the Other heading.
- Select a translation bundle to open it in the canvas.
- Click + String from the top right of the translation bundle page to open the New String dialog.





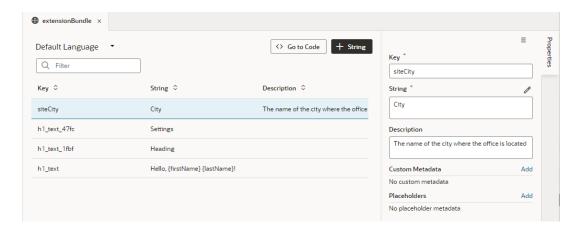
Enter a unique value in the Key field, then type the string you want to save in the String Editor.

If you enter an expression, VB Studio parses the string as you type to check that it is syntactically correct. If there are errors, these are displayed below along with a preview of the string.

5. When you're satisfied, click **Create**.

The translatable string is added to the translation bundle.

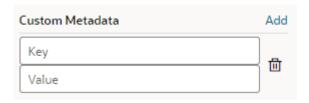
6. To add a description for the string, select the key from the list and type a description in the Properties pane.



The **Description** field can provide additional context for the string that can be useful to translators when translating the string.

To add custom metadata, click Add beside Custom Metadata and enter a key and value in the available fields.





You can add any information here that helps others understand the purpose or context of the translation key.

8. If there are placeholders in the translatable string, you can add placeholder metadata here. Click **Add** beside **Placeholders** and enter a placeholder name and description for each placeholder in your expression.

For more information about placeholder metadata, see Use an Expression in a Translatable String.

Associate a Translation Key with a UI Component

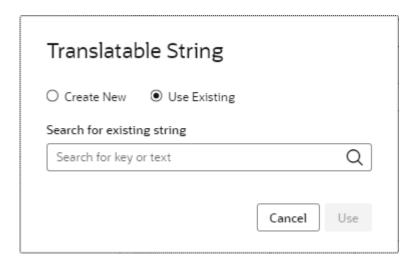
When you add a UI component to your App UI, you'll want to associate a translatable string with it so that the component's text property displays in the correct language once your App UI is deployed.

You can use the Properties pane to bind an existing string to your UI component. Or, if the string you want has not been saved to a translation bundle, you can add it to your extension's translation bundle here. When you add a new string, a key is generated automatically, but you can specify your own key in the dialog box, if desired.

Translatable strings can be static values or expressions in ICU format. If you want to use an expression in your string, refer to Use an Expression in a Translatable String.

To associate a translation key to a UI component's text property:

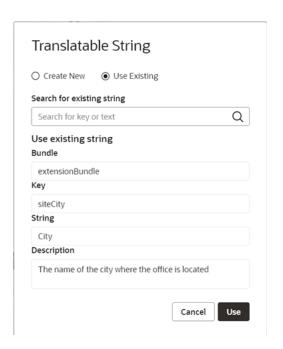
- 1. Select the component on the canvas that you want to associate with the translatable string.
- 2. Hover over a text field in the Properties pane to display the globe icon (above the field. The field could be any text-type field such as text, label, label hint, or placeholder.
- 3. Click the globe icon () for the text field to open the Translatable String popup.





4. To use an existing string, choose **Use Existing**, type the translation key or string in the search box, then select it from the list.

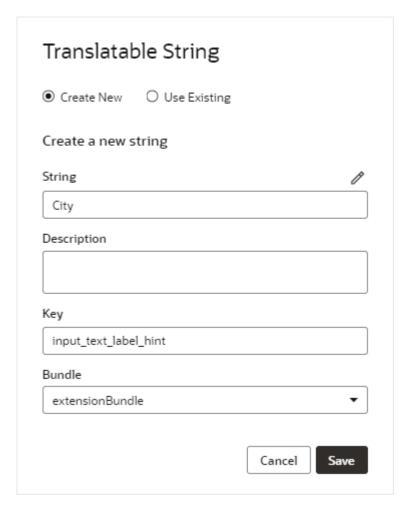
The popup displays the translation bundle, key, and string.



5. Click **Use** to associate the component with this string.

A JavaScript function call, such as [[\$translations.extensionBundle.siteCity()]], replaces the text value in the Properties pane.

- **6.** If you can't find an existing string for your component, create a new one instead.
 - a. Choose Create New.



The **String** field displays the text value for the new translatable string key. By default, this is the text value of the component. The **Key** field displays a unique systemgenerated key for a new translatable string.

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.

- **b.** To create an expression or ICU message, click the String Editor icon () to open the String Editor and create your expression. See Use an Expression in a Translatable String.
- **c.** If desired, replace the system-generated identifier for the key or select a different translation bundle to save it to.
- d. Click Save.

A JavaScript function call, such as

[[\$translations.extensionBundle.input_text_label_hint()]], replaces the text value in the **Text** field.

If you just created a new translation key and want to add metadata, open the translation bundle and set custom or placeholder metadata for the new key. See Create Translation Keys for the steps.

If your string includes an expression, VB Studio uses default values for the expression's parameters. You'll need to use the Expression Editor to replace any variables in the expression with variables defined in the App UI. See Use an Expression in a Translatable String.

Expressions in Translatable Strings

VB Studio supports expressions, including ICU (International Components for Unicode) messages, in translatable strings that you save to a translation bundle.

Suppose you have a page that provides details for a training course and you want to add a message that displays the number of people enrolled. To do this, you would create an expression in the String Editor that outputs a different message depending on a numeric value, such as:

For this to work, you'll need to assign a variable that holds the number of enrollees for the course to the COUNT parameter in your expression. In this example, if the value of the enrollees variable is "4", the output of the expression is "There are 4 people enrolled."

Just like static strings, strings with expressions can be reused in multiple places and translated along with static strings when you translate your App UI.

If you're unfamiliar with ICU, refer to the ICU User Guide...

Use an Expression in a Translatable String

You can create and test an expression or ICU message in VB Studio using the String Editor. This editor parses your expression as you type it to ensure that it is syntactically correct.

If the expression is valid, the editor displays your expression's parameters as well as the resulting message in a preview area. You can use the preview area to type in values for your parameters to see how your message looks.

You can access the String Editor from either your App UI's translation bundle or from the Translatable String popup available from the Properties pane for a UI component. See either Create Translation Keys or Associate a Translation Key with a UI Component for the steps.

After you bind a translatable string with an expression to a UI component, you'll need to assign a variable to each parameter in your expression. You'll do this using the Expression Editor.

Finally, you may want to add metadata for any "placeholders" in your expression so that your translators know how to translate it. You'll need to do this from the Properties pane for the translation key.

A placeholder is text in the translatable string that stands in for the value of a variable and doesn't need to be translated. Placeholders are surrounded by curly brackets ({}) in the string. If you are concerned that the purpose of a placeholder isn't clear and that this could lead to an incorrect translation, you can add a description to provide more context.





This task only covers creating the expression and assigning variables to it. Before you begin, ensure you have a variable defined in the App UI that holds the appropriate value for each parameter in your expression.

To create a translation key that uses an expression or ICU message:

- Open the translation bundle and click + String to create a new string.
 You can also create a new string from the Properties pane for a UI component. See Associate a Translation Key with a UI Component.
- Create your expression in the String Editor.

```
String
      1
           {COUNT, plural,
      2
           =0 {No one is currently enrolled.}
      3
            one {There is one person enrolled.}
      4
            other {There are # people enrolled.}
      5
Preview your message
COUNT
 25
There are 25 people enrolled.
                                                                  Cancel
                                                                           Done
```

In this example, the ICU message outputs a different message based on the number of people enrolled in a course.

If there are any errors, these are displayed in the **Preview your message** area. If the expression is valid, the String Editor displays any parameters you've entered as well as a preview of your string. If desired, you can test your expression by providing values for your parameters to see that the message is properly formatted.

When you have created and tested your expression, click **Done**. Your expression is displayed in the **String** field.

You are now ready to use this translatable string in your App UI.

Bind the translatable string to a UI component.

If you created the translatable string from your translation bundle, you can now associate it with a UI component. Open the Translatable String popup from the component's Properties pane, select **Use Existing**, then search for your new string.

If you created your translatable string from the Translatable String popup using the **Create New** option, you can now click **Save**. See Associate a Translation Key with a UI Component.

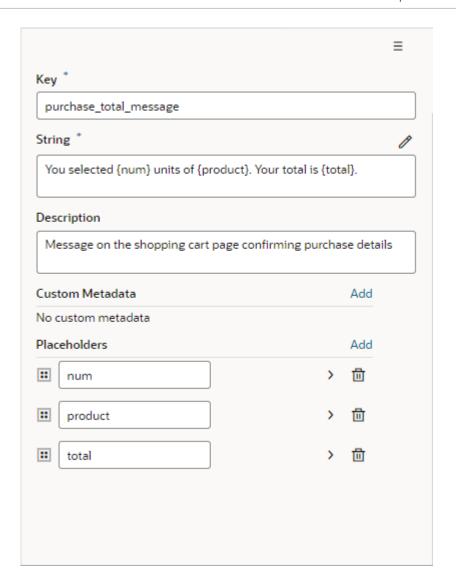
When you bind your translatable string with a UI component, a JavaScript function call, [[\$translations.extensionBundle.bind_text_value_9ef4({ 'count': 2 })]], replaces the value in the text field for your component.

VB Studio assigns a default value to each parameter in your expression. (In this example, the default value for the count parameter is 2.) Next, you'll need to replace the default values with variables that contain appropriate values.

- Assign variables to your expression:
 - a. Click the Expression Editor icon (f^x) above the text field to open the Expression Editor.
 - b. From the Expression Editor, replace the default value for each parameter with an appropriate variable from the Variables pane. In this example, when you assign a variable for enrollees to your count parameter, the updated JavaScript function call might look like this:
 [[\$translations.extensionBundle.bind_text_value_9ef4({ 'count': \$variables .enrollees })]]
 - c. Click Save.
- 5. Add placeholder metadata for the translatable key.
 - a. Open the translation bundle and select the translation key with the expression.
 - b. From the Properties pane, click Add next to the Placeholders heading.
 - c. Type the name of the placeholder in the available field, then click the right arrow.
 - **d.** Type a description for the placeholder in the **Description** field.

In this example, the translatable string includes three placeholders: num, product, and total. Metadata are included for these placeholders under Placeholders.





Note:

You may want to add metadata for a placeholder that is not used in the root (English) string, but which may be used by a translator creating a different localization. If a translation of the string uses this placeholder, it's the developer's responsibility to ensure an appropriate value is assigned to the placeholder at runtime.

To view the translation key in the JSON file, click <> Go to Code. The code should look like this:

```
"description": "The number
    of units selected for purchase"
},
"product": {
      "description": "The name of the product
      selected for purchase"
},
"total": {
      "description": "The total for the items
      selected for purchase"
}
```

Override a Translation Key Value

As you configure your App UI, you may want to use a different string value for a UI component than what the App UI's creator originally specified. You can't modify these translation bundles directly as they are read-only, but you can override the values in these translation bundles.

When you override a value, this value is used everywhere the translation key is referenced. Suppose you want to replace every instance of "customer" with "client" throughout the App UI, but the translation bundle that stores this string is read-only. Simply type a new value for the string to override the value.

Overridden key values are stored in a separate translation bundle. Override bundles are appended with -x; for example, <base-bundle-name>-i18n-x.json.

To override the value of a translation key in a read-only translation bundle:

 From the Translations pane, select the translation bundle with the key you want to override.



2. From the translation bundle page, select an available key.

The value is displayed in the **Key** field in the **Properties** pane.

3. Edit the value.

The overridden key is displayed in bold in the list. An override bundle with a -x designation appears in the source folder.

Download and Upload Translation Bundles

If you plan to translate your App UI into one or more languages, now is the time to send your translation resources for translation. VB Studio can provide your translation files in either the JSON or the industry standard XLIFF (XML Localization Interchange File Format) format.

To translate your App UI, download the translation bundles, translate the strings into the desired languages, and then update the locale for each translation file. You can choose to download the full set of translatable strings, or just download the strings you've added or changed since the last time you downloaded.

When you prepare your translation files for upload, save the translation files for each language in a separate folder using the locale code for the folder name. For example, if you translate the <code>extensionBundle-i18n.json</code> bundle into Icelandic, save it to a folder named is for the Icelandic locale code, like this:

```
is\extensionBundle-i18n.json
```

When you're ready to upload your new files, add them to a zip file, taking care to use the same folder structure as the downloaded zip file. Suppose the downloaded zip included two translation bundles for your root locale, extensionBundle1 and extensionBundle2, with translation files for the Arabic and Korean languages. The structure will look like this:

```
.
.\extensionBundle1-i18n.json
.\extensionBundle2-i18n.json
.\ar
.\ar\extensionBundle1-i18n.json
.\ar\extensionBundle2-i18n.json
.\ko
.\ko\extensionBundle1-i18n.json
.\ko\extensionBundle2-i18n.json
```

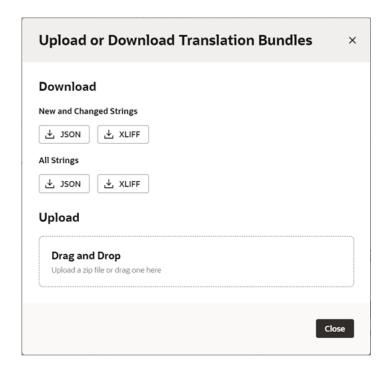
If you translate each root translation file into German, you would create a folder (de) at the top-level of the zip and save the new translation files in this folder, like this:

```
.\de
.\de\extensionBundle1-i18n.json
.\de\extensionBundle2-i18n.json
```

Note:

The XLIFF files you download from VB Studio use the XLIFF file structure but include message strings in the ICU format. This is because XLIFF messages don't support the full feature set provided by ICU.

- Open the Translations tab ().
- 2. Click the Upload or Download Translation Bundles icon (to open the Upload or Download Translation Bundles dialog box.



To download the complete set of translation strings, click the JSON or XLIFF download button under All Strings.

To download only new or changed strings, use the buttons under **New and Changed Strings** instead.

Visual Builder Studio downloads a zip file containing all the translation bundles for your extension, including those with overrides from any read-only bundles.

Initially, your extension translation bundle will include only the default language JSON file,

<

4. Unzip the files to your desktop.

After the files have been unzipped, you'll see the root bundles at the top level as well as separate folders for each language the bundles have been translated into.

- 5. Create a new file for each language you want to support and translate the strings into that language:
 - a. Open each translation file and translate the strings into the desired language.

Here is a sample of a JSON file showing header details, such as bundle name, path, and locale. It also shows the first key-value pair for the Application Navigation Drawer.

```
{
  "@@x-bundleName" : "extensionBundle",
  "@@x-bundlePath" : "extension/sources/translations/self/
extensionBundle-i18n",
  "@@locale" : "en",
  "app_title_navigation_drawer" : "Application Navigation Drawer",
  "@app_title_navigation_drawer" : {
    "description" : "Title Application Navigation Drawer",
    "source_text" : "Application Navigation Drawer",
},
```

b. Modify the locale value of the file to include the two-letter language code for the translation language. For example, use fr for your French language file, de for your German file, and so on.

The locale attribute for the French file should appear like this: "@@locale" : "fr",

- c. Create folders for each language using the locale code for the folder name. In this example, you would need a folder named fr for your French language translation file.
- d. Save your files to the appropriate folder.

Your translation files should use this format for the file name:

```
<bundleName>-i18n.json
```

The relative path for a French language translation file using bundle name extensionBundle should look like this:

```
fr\extensionBundle-i18n.json
```

6. Add all new and updated folders and files to a zip file.

When you create your zip file, it must have the same structure as the downloaded zip file, with the root bundles at the top level and each translation file in a separate language folder underneath.

Open the Upload or Download Translation Bundles popup and drag the updated zip file to the Drag and Drop area.

You'll see a notification about the number of new and updated bundles that you have just uploaded.



Part III

Configure an App UI

Use the chapters in this part to configure an App UI—one created by Oracle, or by someone else at your company—to suit your business needs.

- Customize an App UI
- Customize Dynamic Tables and Forms
- Customize Dynamic Containers
- Customize Variables and Constants
- Trigger Actions in Dynamic Components
- · Work With Fragments From Dependencies

If you can't satisfy your business needs by adding and configuring existing dependencies, you might want to create your own App UI in your extension. By creating your own App UI, you can add the pages and application resources you need to your Oracle Cloud Application. For details about creating an App UI, see Build an App UI or Fragment.

VB Studio provides two different modes for customizing pages in dependencies: Express and Advanced. If Express mode is available, you can use buttons in the Designer header to toggle between the two modes. The editing tools in Express mode are designed to help you quickly customize components, without the distraction of the more complex tools in Advanced mode. This chapter describes how to use Advanced mode to customize pages. If you prefer to use Express mode, you should refer to Extending Oracle Cloud Applications in Visual Builder Studio Express Mode.



9

Customize an App UI

A configuration to an App UI can be as simple as just hiding certain fields in a dynamic form or table for certain audiences, to displaying brand new content through the use of a dynamic container.

Note:

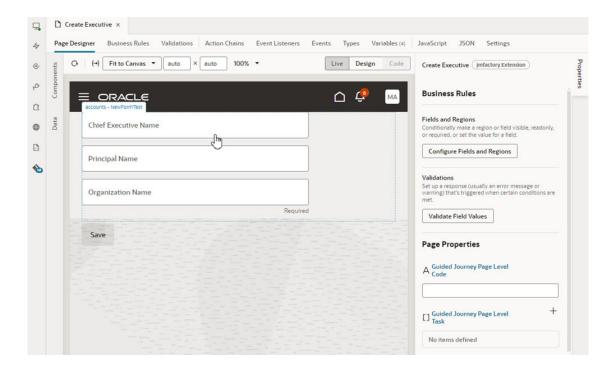
If you're new to App UIs, it's strongly recommended that you acquaint yourself with these topics before you get started:

- The Basics, to understand essential concepts like dynamic components, dependencies, and so on;
- How Are App UIs Structured?, to get an idea of how App UIs are laid out;
- Configure an Oracle Cloud Application, if you're not sure where to start.

Perhaps the two most important questions when it comes to configuring an App UI are:

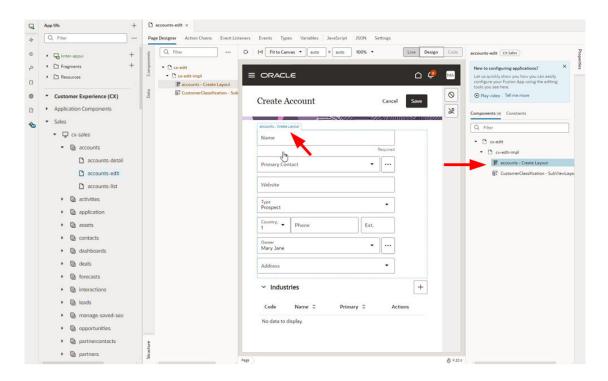
- · What can I configure?
- What if I don't see the App UI I want to configure in the Designer?

When your page is open in the Page Designer, you can use the canvas to select customizable components. As you move your cursor over the page, the customizable components become outlined in blue, and a tag with the component's name appears at the top:



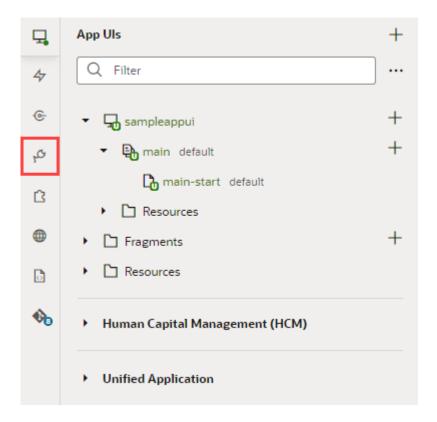


You can also use the Properties pane to select and configure the page. In the image above, the Properties pane contains links to edit the page using business rules. However, the Properties pane might look like the image below, where the Properties pane contains tabs listing the page's editable components and constants. When you hover over a component on the canvas, the component is also highlighted in the Properties pane's Components tab:

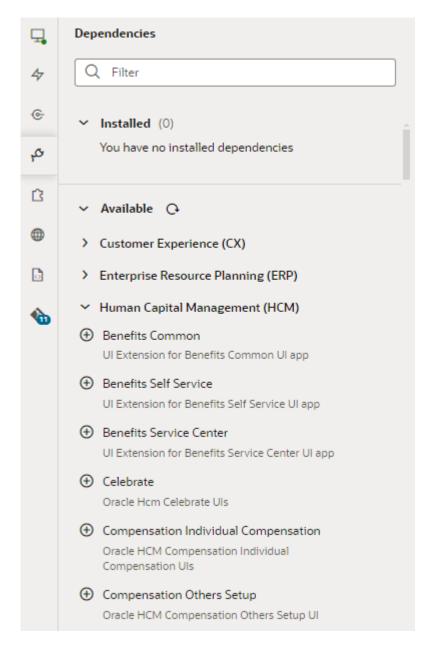


To configure an App UI, the extension containing the App UI you want to configure must be listed as a dependency in the App UIs pane. This means you should see your dependent extension listed by pillar. If you don't see the App UI you want listed by pillar, you can add it by clicking **Dependencies**:





This opens the Dependencies pane, where you can select the extension you want to add as a dependency. When adding a dependency, you add the extension *containing* the App UI you want to configure, not the App UI itself.



What Are Dependencies? explains more about how to work with dependencies in your extension.

How Tos

You may find this curated list of topics helpful as you start to customize your Oracle Cloud Application:

Dynamic Tables and Forms

- · Change the set of fields displayed
- · Define display logic to conditionally change the set of fields displayed
- · Conditionally hide or show a specific field
- Partition form field into subgroups



- Change how a specific field is displayed using a field template
- Configure a field to call an action
- Completely override the way the entire form is laid out using a form template

Dynamic Containers

- Add (or reorder) sections of content on a page
- Add content from a custom child object to a page
- Add content from a custom service connection to a page

Layouts

Create fields to use in a layout (for calculated and virtual fields)

Constants

Change a specific behavior of the application

Fragments

· Create a re-usable module, like a foldout panel or a footer

Events

- Define actions that take place in response to application events
- Use events provided by the Unified Application to start action chains



10

Customize Dynamic Tables and Forms

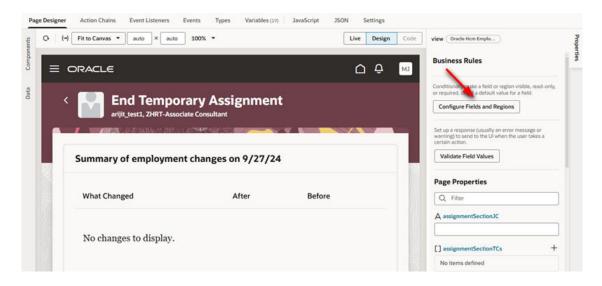
If your Oracle Cloud Application contains extendable dynamic components, you can configure them to customize how they are rendered in pages. You may want to hide a table row, say, or display some extra fields in a dynamic form that Oracle—or perhaps the creator of an extension you've added as a dependency—didn't include.

There are two ways to customize dynamic tables and forms: *business rules* and *rule sets*. They each allow you to do the same thing, that is, they allow you to customize what is displayed in a table or form based on conditions, but the tools used to do this are different. How you choose to customize the tables and forms will depend on your needs. Business rules can be quicker when you are performing simple tasks, such as hiding a field, or making a field read-only, enabling you to quickly configure a page's tables and forms. Rule sets are better suited when you need more control over how a table or form is rendered, allowing you to configure a component's layout at a more granular level, which is not possible using business rules.

For more on the differences between rule sets and business rules, see What Are Dynamic Components?.

When Do You Use Business Rules?

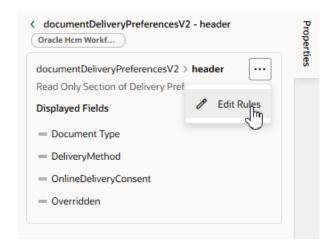
You use business rules when you want to customize the properties of fields in forms and tables, for example, to make a field in a form read-only, or hide a column in a table. You can do this in a specific component or to every component on the page, as long as business rules are enabled for the page. If your page looks something like this, the dependency developer has enabled business rules for the page, so you can open the business rules editor using the Configure Fields and Regions button as shown here:



When you click Configure Fields and Regions, you'll be able to create business rules that can customize all the tables and forms in the page. To see how to customize tables and forms using the business rules editor, see Control Your Display with Business Rules .

If you aren't able to accomplish what you want using business rules, you can try using rule sets. If the page is enabled for business rules, try selecting the component you want to edit in

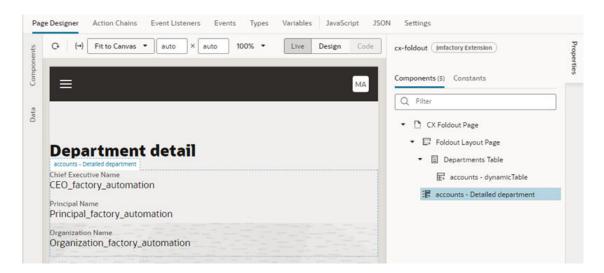
the Structure view. When you select the component, its details pane opens in the Properties pane. The details pane contains a link to open the component's rule set in an editor.



When Do You Use Rule Sets?

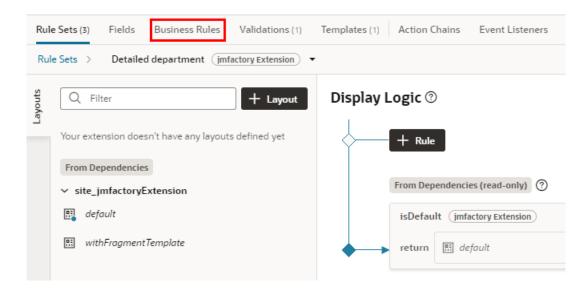
Customizing tables and forms using rule sets is more complex than using business rules, but you get greater control over what is displayed in the component. For example, if you want to customize the order that fields are displayed in a component based on some conditions, you need to use rule sets. However, you do need to configure each component's rule set individually, which can be complicated and time-consuming when you want to customize multiple components, with many different conditions and layouts.

If your page looks something like this, it means the dependency developer has not enabled business rules for the page, and you need to configure the page's components individually:



Notice there is no direct link to the business rules editor in the Properties pane. Instead, the Properties pane lists the page's editable components. When you select a component, the Properties pane displays the component's details pane, which contains a link to open the component's rule set in an editor. To see how to customize a component's rule set, see Control Your Display with Rule Sets.

Once you are in the rule set editor, you can open the Business Rules tab in the Layout if you decide to configure the component using business rules:



You can use rule sets and business rules together. For example, you could use a rule set to customize the order that fields are displayed in a form, and then use business rule to hide fields under some conditions.

Control Your Display with Rule Sets

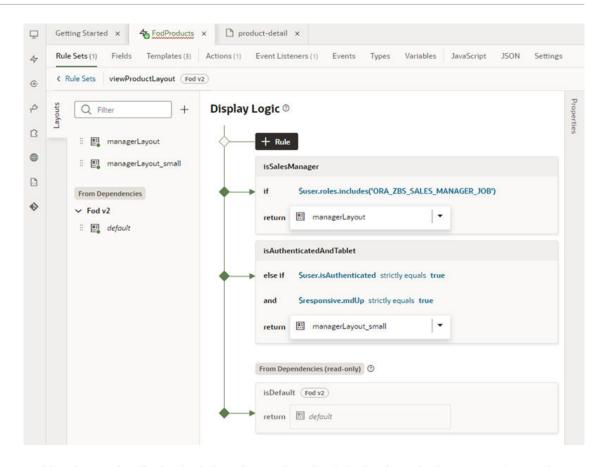
When you use Advanced mode to customize dynamic tables and forms in a page, all the elements that control how they are rendered are configured in a *Layout*. A Layout will contain a *rule set* for each dynamic table and dynamic form component. The rule set determines what data is displayed in the component. The Layout might also define *templates*, *variables*, *action chains* and other artifacts that are used to control how the components look and behave. A Layout is defined at the extension level, which means it is available to all the App UIs in your extension. For details, see Work With Layouts in Your Extension in Part 2 Build an Extension.

Note:

Recall that everything you do to customize an Oracle Cloud Application is done within an extension. In addition, you must add the App UI you want to customize as a dependency, either by clicking **Edit Page in Visual Builder Studio** in your Oracle Cloud Application, or by using the Dependency tab explicitly.

Determine What's Displayed at Runtime With a Rule Set

To configure a dynamic form or table component, you'll usually want to start by opening its rule set. A component's rule set determines what a component displays at runtime through the use of *display logic* and *layouts*. Here's an example of what a rule set for a dynamic form looks like:



In this rule set, the display logic has three rules: the default rule at the bottom, and two rules that have been added above it. The top rule, <code>isSalesManager</code>, determines if the user viewing the page has a specific user role (<code>ORA_ZBS_SALES_MANAGER_JOB</code>). If they do, then the user will see the fields defined in the <code>managerLayout</code> layout displayed in the form. If they don't, the next rule (<code>isAuthenticatedAndTablet</code>) is evaluated.

This rule set also has two layouts, manageLayout and manageLayout_small, in addition to its default layout. Each layout contains a different configuration of fields, columns, and so on, which dictate how the component should look when rendered at runtime.

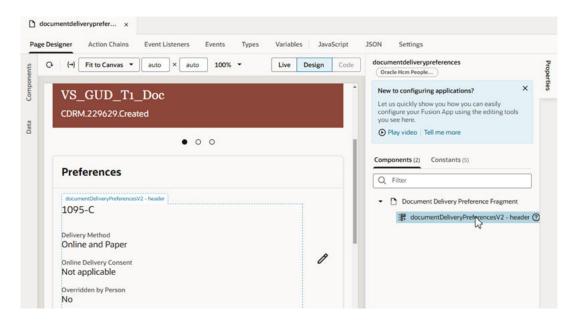
Open a Rule Set From Oracle Cloud Applications

The simplest way to open a component's rule set is from the Oracle Cloud Application page where the component is located.

To open the rule set of a dynamic table or form component from an Oracle Cloud Application page:

- Open the Oracle Cloud Application page containing the component.
- Click Settings in the upper right corner, then click Edit Page in Visual Builder Studio to open the page in the VB Studio Designer.
- 3. In the Designer, select the dynamic form or table you want to configure.

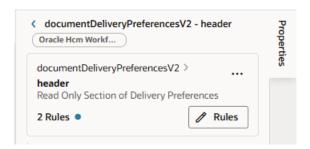
You can select the component on the canvas, in the Structure view, or in the Properties pane. The Component's tab in the Properties pane lists all the editable components on the page. A blue border is displayed around the component on the canvas when your cursor hovers over it:



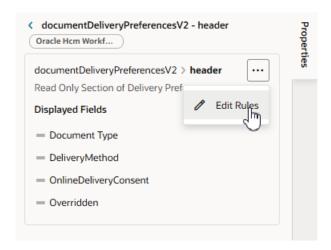
If you don't see the component you're looking for on the canvas or in the Properties pane, you can open the Structure view and select the component there.

4. Open the component's rule set in the editor.

When you select a dynamic table or form component, the component's details pane opens in the Properties pane. If the details pane looks like this, click **Rules** to open the rule set editor:



If the component only has the default rule, the component's details pane looks like the image below. In this case, you can edit the layout directly in the details pane (see below), or open the rule set editor by clicking ..., and then selecting **Edit Rules** in the menu:



After opening the component's rule set, the next step will be to start configuring it by creating rules in the rule set's display logic.

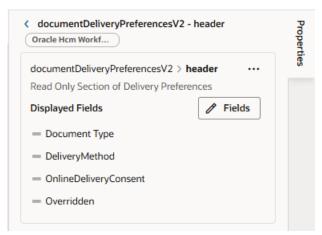
Edit a Component's Layout in the Properties Pane

Many dynamic forms and tables have rule sets with only one default rule. When this is the case, you can change the fields displayed in the component directly in the Properties pane, without opening the rule set editor.

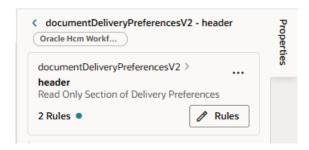
To edit the layout in the Properties pane:

- Open the page you want to modify in the Designer.
- 2. Select the component in the Properties pane or on the page to open the component's details pane.

If the component's details pane looks something like this, the component's rule set has one rule, which is always applied, and you can edit the layout to add, remove, and change the order of the fields directly in the Properties pane:

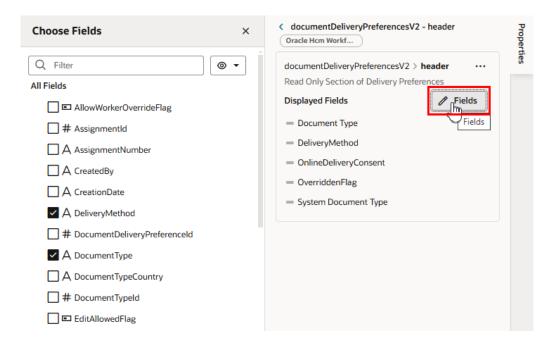


However, if the details pane looks like this, you need to use the rule set editor to configure the fields:



In this case, you click **Rules** to open the rule set editor, and then create your rules. See Create a Rule in a Rule Set.

- 3. Edit the fields displayed in the component.
 - To change the field order, grab the handle next to the field and moving it into position.
 - To delete a field, click the Trash icon next to the field name.
 - To add and remove fields, click Fields, and then select and deselect fields in the Choose Fields popup.



Note:

Editing the fields displayed in the component creates a new layout and a new rule named isDefault1. The new rule has no conditions.

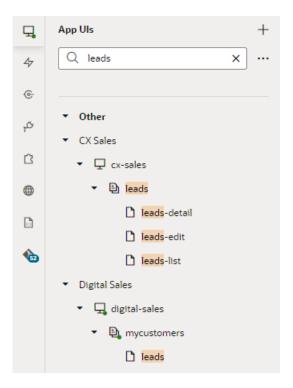
The changes you make to the layout in the Properties pane are always displayed in the component because the rule controlling the displayed fields has no conditions. If you want to use conditions to determine the fields displayed in the component, or you want to group fields, you need to create the rules in the rule set editor. For more, see Create a Rule in a Rule Set.

Open a Rule Set From Visual Builder Studio

If you opened VB Studio by logging in directly, instead of from an Oracle Cloud Application page, you can open a component's rule set by using the Navigator to locate the page containing the component. You can also use the Navigator to open the Layout where the component's rule set is defined. (The Layout is where all the elements for configuring a dynamic table or form are found. For more about Layouts, see Work With Layouts in Your Extension.)

To open a dynamic table or form's rule set using the Navigator:

- In your VB Studio project, open the workspace with the extension you want to configure.
 If you don't have an extension or workspace yet, see Create an Extension.
- 2. Do one of the following:
 - If you know the page containing the dynamic table or form you want to configure:
 - a. Open the App UIs pane.
 - Select the page in the list of dependent extensions. Dependencies are grouped by pillar in the App UIs pane. To locate the page, you can browse the dependencies listed under each pillar, or you can use the pane's Filter field to filter the list by the page name or extension name:



If you don't see the page you want to configure, you need to add the extension in the Dependencies pane.

Note:

Your extension will always contain the Unified Application as a dependency, but the page containing the dynamic component will be in one of the other dependencies.

c. In the Page Designer, select the component you want to configure to open the component's details in the Properties pane.

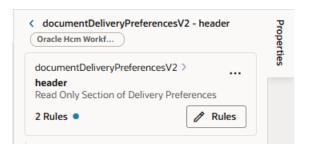
If the page contains any dynamic forms or tables, they will be listed in the Components tab in the page's Properties pane. If the component is visible, they will be outlined in blue when you hover your cursor over them on the canvas or in the Components tab.

If you don't see the component you're looking for on the canvas or in the Properties pane, you can open the Structure view and select the component there.



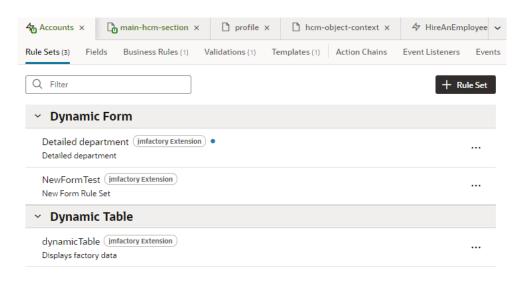
Not all pages have extendable components, so you'll see a note in the page's Properties pane if there aren't any.

d. Click **Rules** in the component's details pane in the Properties pane.



- If you know the Layout containing the dynamic table or form you want to configure:
 - a. Open the Layouts pane in the Navigator.
 - b. Select the Layout in the list. The Layout opens in a new window that contains tabs for each of the editors you'll use to configure the Layout: Rule Sets, Fields, Templates, and so on.
 - **c.** Open the **Rule Sets** tab (if it's not already open), and then click the name of the dynamic form or table you want to configure:





The Rule Sets tab lists the dynamic components defined in the Layout, grouped by type. In the image above, the Accounts Layout has three rule sets. You would see more rule sets if there were more dynamic forms or tables defined in the Layout. From the badge next to the rule set's name, we know the name of the dependency where the rule set is defined (jmfactory Extension).

The next step is to start configuring the rule set by creating rules in the rule set's display logic.

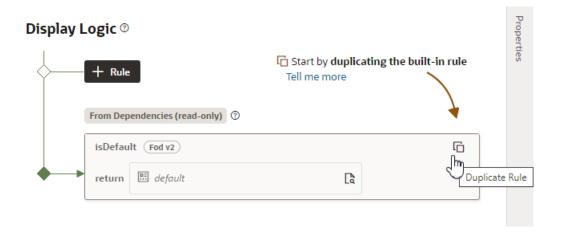
Create a Rule in a Rule Set

Once you've opened the rule set for the component you want to configure, you can start adding rules to the display logic. You'll see that the display logic will already have a default rule, and you'll add your rules above it. 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 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 on the display logic rules.

To configure rules in the display logic:

- 1. Open the rule set you want to configure.
- 2. Under Display Logic, click and give it a name, preferably something meaningful. For example, to create a rule that displays only when the user is a sales manager, you might call the rule isSalesManager.

The rule set for a dynamic component always contains a default rule that is set by the Layout in the dependency. You can't change this rule, but you can copy it to use it as the basis for your own rules, or you can create a rule from scratch.



If you want to also create a copy of the layout to use as a starting point, make sure that check box is selected in the Duplicate Rule dialog box.

3. In your new rule, click **Click to add condition**, then select an Attribute and Operator from the dropdown lists, and select or enter a Value. Click **Done**.

The Attributes dropdown list contains the fields and variables that you can use, and the Operators list contains the operators (for example, '=' and '<=') that are valid for the attribute you select. You can set the Value by typing in the field or selecting a value in the dropdown list. If a list of valid values for the attribute can be determined from the data source, the values will be displayed in the Value dropdown list. For example, if the service definition contains an attribute (Status) that can have three valid values ('Not Started', 'In Progress', and 'Completed'), the dropdown list would display these values.

There are several context objects in the Attributes list that you can use when building rules. In addition to the *\$fields* context, there are variables in other built-in contexts that provide a way to access various pieces of information when building conditions. 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. Context variables include:

\$fields variables determined by the fields available in the Layout. 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 <code>%numberValue</code> (for example, <code>%fields.ConflictId.numberValue()</code>) or <code>%value</code> (<code>%fields.ConflictId.value()</code>). You should use <code>%numberValue</code> 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 <code>%numberValue</code>, the field's value will be converted to a number, if possible. If the value can't be converted, the <code>%numberValue</code> will be <code>NaN</code> (Not a Number). The only limitation is that <code>%numberValue</code> 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:

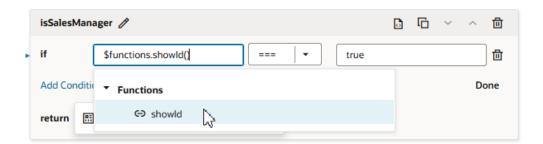
When using user.roles, start entering the role name you need in the Value field. The Value drop-down will display up to 25 matching Oracle Cloud Applications job and abstract roles; keep typing until the desired role displays. (The drop-down will not list any duty roles. If you want to specify a duty role, you can manually type the duty role name in the Value field.)

Note:

A rule condition can also specify a discriminator sub-type value when the attribute is a polymorphic object. When you open the layout associated with a rule which specifies a sub-type, the object's sub-fields in the Fields palette will be filtered using that sub-type. For example, if a condition requires the Meal sub-type, when you open the rule's layout, the object's sub-fields will be filtered for that sub-type.

If you want to edit or remove the filter, click **Modify** under the object in the Fields palette. To re-apply the filter using the same sub-type, click **Reset according to Display Logic rule**. For more details, see Work with Polymorphic Objects in a Layout.

If any JavaScript functions are defined in the Layout's JavaScript tab, you can use them in the condition. For example, you might add a function named <code>showId</code>, and then create a condition that calls the function. The functions that can be used in the condition are listed in the Attributes dropdown list.

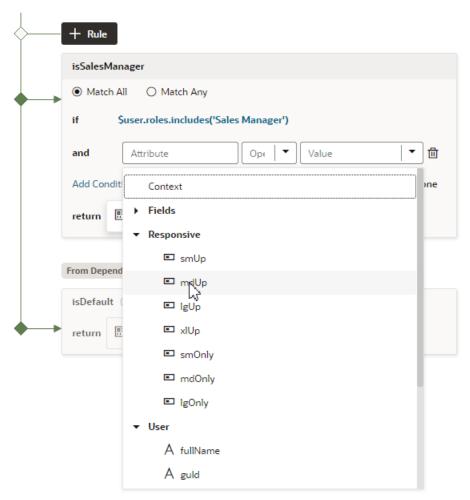


You can also click [□] to open the condition's code view, and use the code completion to help you add the function. For more about using JavaScript in your app, see Add a Custom JavaScript Function.

You can add more conditions and group conditions if you want to use more attributes to make the rule more precise. For example, you may want to use a layout that displays an extra column if the user has the manager role AND is on a device that has a medium-sized screen. You would then create a rule with two conditions, and select **Match All** to require that both conditions are true.



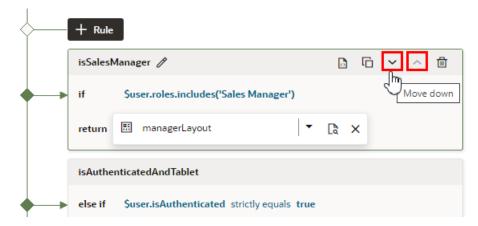
Display Logic ®



If you select **Match Any**, then the rule will be evaluated as true if any of the conditions in the rule are true.

- 4. 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.
- 5. Click + Rule to add another rule, and then build its conditions.
- 6. Use the **Move Up** and **Move Down** buttons to make sure you have the rules in the order you want them evaluated.

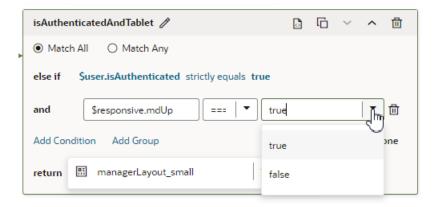
Display Logic ®



Remember, 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.

Create a Rule Based on User and Device

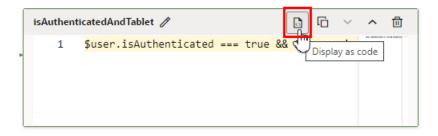
When you're creating rules, you build conditions using the dropdown lists to select an Attribute, Operator and Value. In addition to fields retrieved from the Oracle Cloud Application, the Attributes dropdown list includes some built-in variables storing details about the current user and the device they're using. You can use them, for example, to create a condition that is true only when authenticated users are viewing the page on a tablet-sized screen or larger:



You can select built-in variables that are determined by your extension's context (for example, user.email). Two kinds of built-in extension context (\$context) variables are available:

- user variables are determined by the current user. For example, the user.isAuthenticated variable's value is True if the current user is an authenticated user.
- responsive variables are determined by the screen size of the device your application 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.

If you click oyou can open an editor to see and edit the expression for the rule:



For the rule above, you'd see the following expression:

```
$user.isAuthenticated === true && $responsive.mdUp === true ?
'managerLayout small' : null
```



Depending on your Oracle Cloud Application, when using \$context.user.roles in a condition or display properties, the list of values might contain the app's pre-defined roles (for example, \$context.user.roles.manager). If the attributes list displays roles as a property, in your rule you would specify the value as true or false. If the roles are not listed, then type in the role name manually.

Create a Layout in a Rule Set

A rule set's *layout* defines the fields that are displayed in a dynamic component at runtime. You create and configure the component's layouts in the Rule Sets editor.

Note:

Whereas a *Layout* (with a capital L) is a resource that is available to everything in the extension, a *layout* (lowercase) is an intrinsic part of a particular rule set.

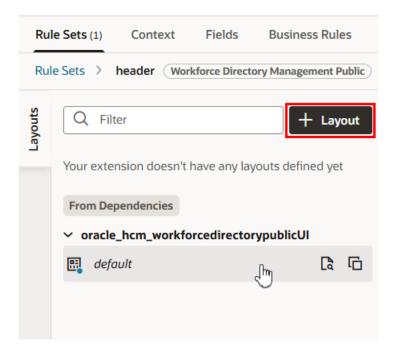
When extending a dynamic form or table, each rule set contains a default layout that is seeded for you. You can't edit the default layout, but you can duplicate it and use it as the basis for a new layout. The default layout is always used in the last rule of the display logic tree.

The fields you can display in a rule set's layout are determined by the fields available from the data resource used by the component. This data resource is defined in the Layout containing the rule set you're configuring. For example, the Layout might define a data resource that has five fields. You can choose which of these five fields (and also any other virtual or calculated fields defined in the same Layout) 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 resources. For details on creating fields in Layouts, see Create Fields For a Layout.

To create a new layout in a rule set:

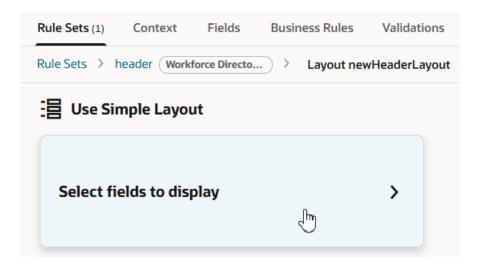
1. In the Rule Sets editor, open the rule set for the component you want to configure.

Click + Layout in the rule set's Layouts pane and type a name for the new layout. Click Create.



You can also click in an existing layout to duplicate it and use it as a starting point.

3. Click the new layout name, then click **Select fields to display** to open it in the layout editor.

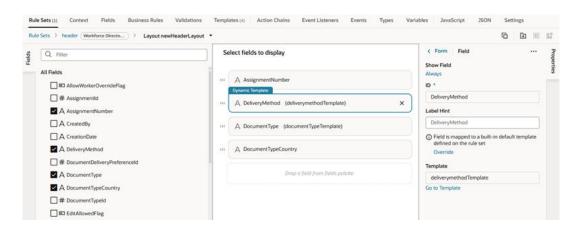


When you create a layout and haven't selected any fields for it yet, you'll see the **Select fields to display** option when you open the layout. (You won't see this option if the layout is a duplicate.) You'll also see the templates that already exist in the Layout listed as options. Click a template name if you want apply the template to the layout, otherwise, click **Select fields to display**.

If you create a new layout rather than duplicating one, the layout might already contain some fields before you've added any. Any fields that are marked as "Required" in the Layout's Fields editor are automatically added when you create a new rule set layout.

Add fields from the Fields palette to the rule set layout.

The Fields palette lists all the fields and objects you can add to your layout. You can add a field or object to a layout by selecting its checkbox in the Fields palette or by dragging it from the palette onto the drop target in the center pane. You can remove a field from the layout by clicking its Delete icon in the center pane.



To help you locate the fields you might want to add, the Fields palette might contain a Suggested Fields section at the top of the palette. This section lists the fields that have been identified as the most relevant or most important when building your layout, which are usually the custom object fields created in App Composer (which you can usually identify because $\, \circ \,$ is appended to the name) and the fields marked as required in the Layout.

You can also filter the list of fields by entering a string in the Filter field at the top of the Fields palette.

Change the order that fields are displayed in the component by dragging fields into position in the center pane.

After a rule set layout is created, you can set it as the layout that a display logic rule should display, as described in Determine What's Displayed at Runtime With a Rule Set. You can use the same layout in multiple rules.

Edit a Field's Properties in a Layout

When you edit a field's properties in a rule set's 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 Required or Read Only. If you want to edit a property so that it's the same in all layouts—for example, if you want it always to be Read Only—you should edit the field's properties in the Fields editor.

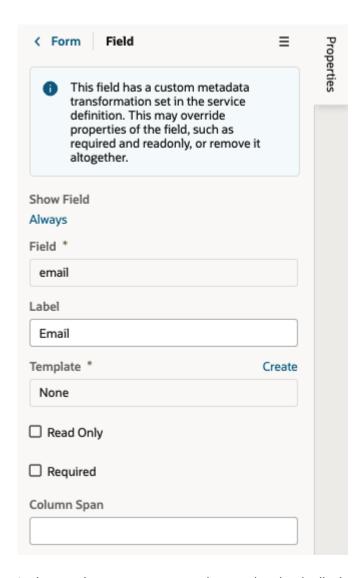
To edit a field's properties:

- 1. In the Rule Sets editor, open the layout and select the field in the center pane.
- 2. Set the Show Field property to control when the field is displayed.

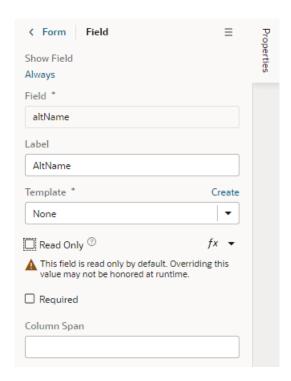
The default setting for this property is Always. For details on how to set the property, see Use Conditions to Hide and Show Fields in a Layout .

3. Edit the field's Read Only and Required properties in the Properties pane.

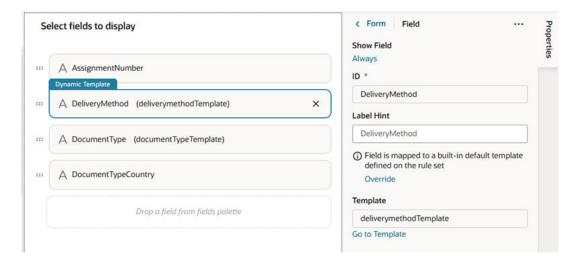
If you are editing the properties of a field defined by an Oracle Cloud Application service, the image below shows the message you'll see in the Properties pane if the field has undergone a custom transformation, which may override any changes you make to the field's properties.



In the next image, you can see the warning that is displayed in the Properties pane when you try to edit a property that you might not be able to override because it was already set in the Fields editor. For example, you'll see this warning if a custom field's property is set to Read Only and you try to override it in your layout.



The Read Only and Required properties might not be editable in the Properties pane if you select a field that has a template applied to it. In this case, you might need to remove the template if you want to edit these properties in the layout. In the next image, you can see that a field template is already applied to the selected field.



For details about using templates with fields, see Apply a Template to a Field.

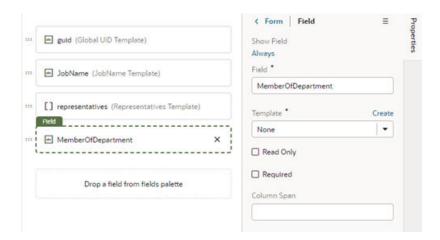
Use Conditions to Hide and Show Fields in a Layout

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 Sets 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.



- 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**.

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.

The Attributes variables are grouped by *context* in the list. In addition to the *\$fields* context, there are variables in other built-in contexts that provide a way to access various pieces of information when building conditions. 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. 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 <code>%numberValue</code> (for example, <code>%fields.ConflictId.numberValue()</code>) or <code>%value</code> (<code>%fields.ConflictId.value()</code>). You should use <code>%numberValue</code> 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 <code>%numberValue</code>, the field's value will be converted to a number, if possible. If the value can't be converted, the <code>%numberValue</code> will be <code>NaN</code> (Not a Number). The only limitation is that <code>%numberValue</code> is limited by the maximum precision allowed by the Number type in Javascript.

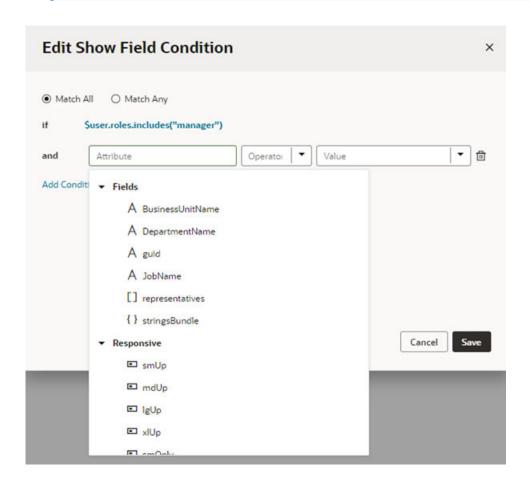


- *\$responsive* variables determined by the screen size of the device the app is currently displayed on. For example, the responsive.mdUp variable's value is True if the current user is using a device where the screen width is 768 pixels or more, such as a tablet. Look for these variables under the **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, start entering the role name you need in the Value field. The Value drop-down will display up to 25 matching Oracle Cloud Applications job and abstract roles; keep typing until the desired role displays. (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 must be granted the

ASE_REST_SERVICE_ACCESS_IDENTITY_INTEGRATION_PRIV privilege to see user roles in the drop-down list. Contact your instance administrator if you require this user privilege.



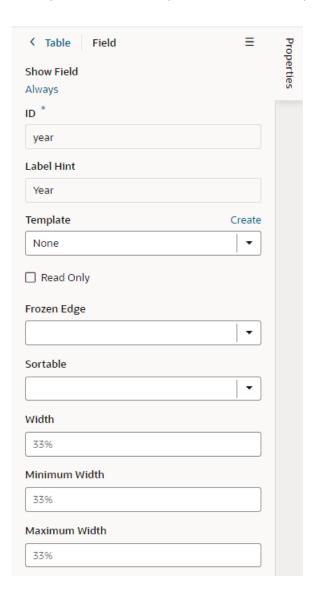
Configure How Columns Are Rendered in a Dynamic Table Layout

When editing the layout for a dynamic table, you can edit the fields in the Properties pane to configure the width of each column in the table, and if the table is sortable by any of the fields. 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.

When you set the properties for a field in a layout, the settings only apply to the current layout. The other layouts are not affected.

To configure the properties for columns in a table layout:

- 1. In the Rule Sets editor, open the dynamic table's layout.
- Select the field in the center pane that you want to edit.When you select the field, you can see the field's properties in the Properties pane.



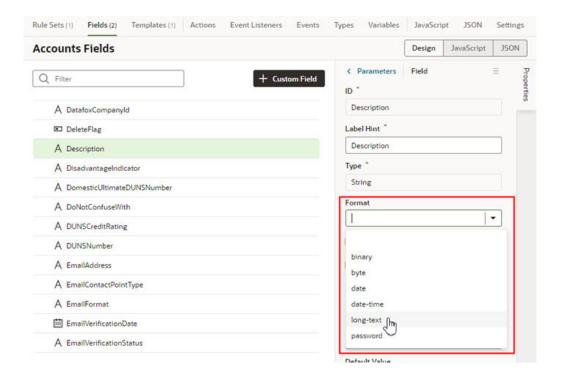
For fields in a dynamic table's layout, the Properties pane contains some display properties to control how the field is rendered in the table:

- Frozen Edge: Set this property to start or end to "freeze" the column of the selected field. Select start to pin the column at the beginning (which means it can't be scrolled past), or end to freeze the column at the end (to lock it in view). For details, see Frozen Columns in the Oracle JET Developer Cookbook.
- Sortable: This property determines if the table is sortable on the selected field.
- Width: This property sets the default width of the selected field column in the table.
- Minimum Width: This property sets the minimum width of the selected field column
 when the table is first rendered on the page. A user can manually resize the column
 width to make it narrower.
- Maximum Width: This property sets the maximum width of the selected field column
 when the table is first rendered on the page. A user can manually resize the column
 width to make it wider.

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.

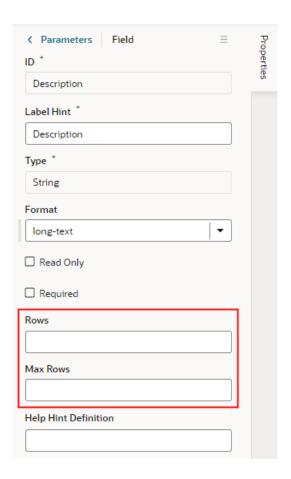


Notice that the Format property now has a blue dot next to it to indicate the property has been modified by the extension.

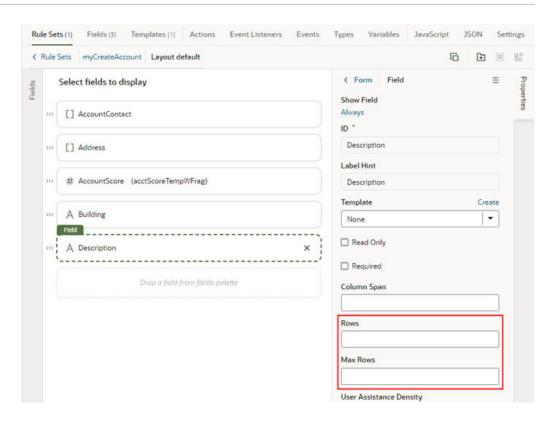
When you switch the format to long-text, two additional properties are displayed in the Properties pane.

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.

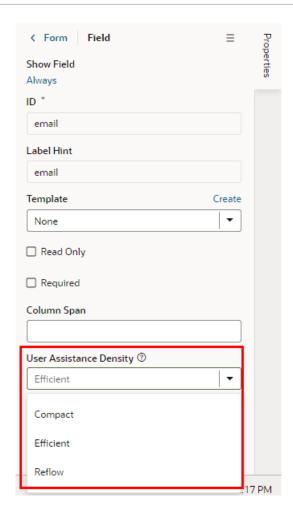


Set How User Assistance is Rendered in a Layout

Use the User Assistance Density property to set how a field's user assistance text, like 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 Sets editor, open the layout and select the field in the center pane.
- Select the field's User Assistance Density property from the dropdown list in the Properties pane.

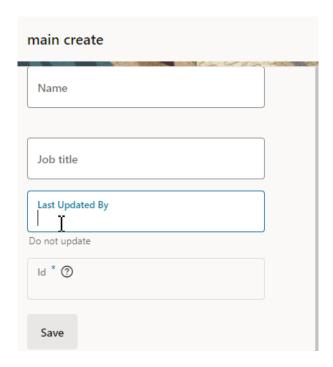


You can choose from three options:

- compact 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 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 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.

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 first field (Name) is set to efficient, the second and third fields (Job Title, Last Updated By) are set to reflow, and the fourth field (Id) is set to compact. You can see that the cursor is in the Last Updated By field, and that the space below the field has expanded so that the field's user assistance text ("Do not update") can be rendered below the field.

Control How a Field is Rendered with Templates

You can customize how fields in dynamic forms and tables are rendered on a page using *field templates*. You can use a field template to apply simple styling details to a field, but a template can also contain UI components, for example, text fields or images, and define their properties. Components in a template can access the variables and constants, even action chains and event listeners, defined in the Layout.

The Layout you're configuring might already define default templates for some fields, which are applied in every rule set layout, but you can override them if you want to apply your own templates. Suppose the Layout defines a field template called BoldType that is applied to the Update field. If you do nothing, the Update field will have the BoldType template applied in every rule set 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.

Create a Field Template

The template editor for creating a field template is similar to the Page Designer, with a Components palette, Structure view, canvas and Properties pane. You can add components to a template by dragging them onto the template editor canvas, and then configuring the component's properties. The components you add to your template can access the variables, constants, action chains and event listeners defined in the same Layout.

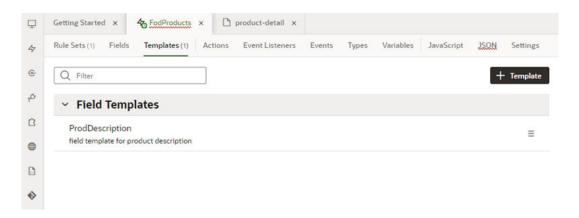


When you create a field template, you can also let other extensions use it when they are extending the same Layout. After they add your extension as a dependency, they can apply your template to fields in the Layout.

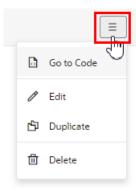
To create a field template for a form field:

1. Open the Layout's Templates tab.

The Templates tab displays a list of field and form templates that are already defined in the Layout. Templates defined in a dependency are labeled with the dependency's name.

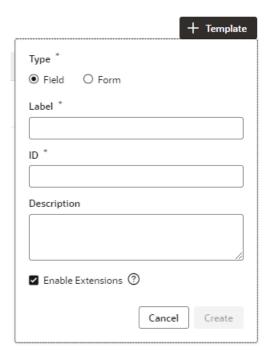


If you want to duplicate a template that you've created, open the template's menu and select **Duplicate**. You can also use the menu to edit, delete, and view the code of templates you've created.



You can also duplicate templates defined in dependencies, but only if the template only contains a fragment (or fragments).

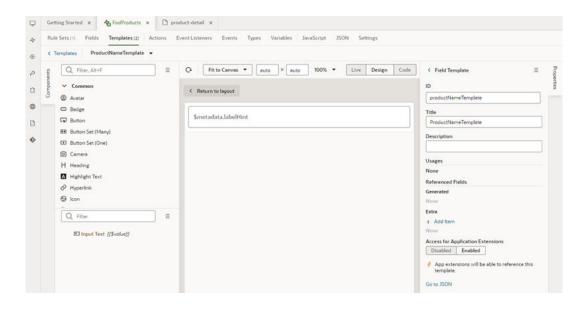
2. Click + Template and select Field in the dialog.



Select **Enable Extensions** if you want to allow other extensions extending the Layout to use this field template in their rule set layouts, though they won't be able to edit the template. Selecting this will automatically make your extension extendable, which means other extensions can add it as a dependency.

3. Specify the Label and ID. Click Create.

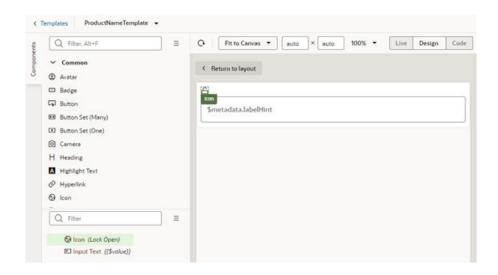
Your new field template has an Input Text component that is generated automatically. This is used to display the data and display name when you apply the template to a field in a layout.



4. In the template 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.

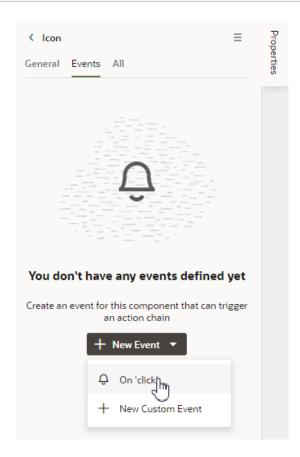
In this image, you can see in the Structure view that the template contains an Icon component and an Input Text component:



5. 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 add 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'd then need to edit the action chain in the Layout's Actions tab to define the behavior.





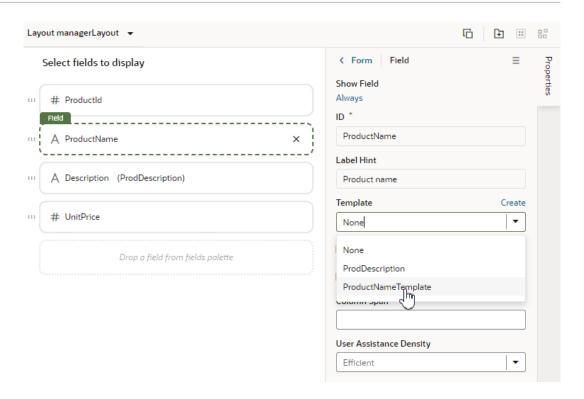
Alternatively, you can edit a field template's code directly in the Code editor, and use the editor's code completion to help you.

After creating the template, it's added to the list of field templates in the Templates tab. In the Templates tab, you can open and duplicate templates you've created.

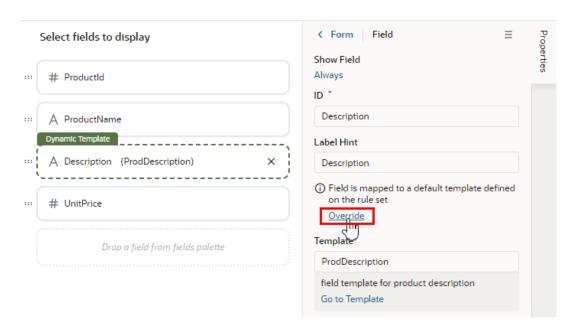
Apply a Template to a Field

To apply a template to a field in a layout:

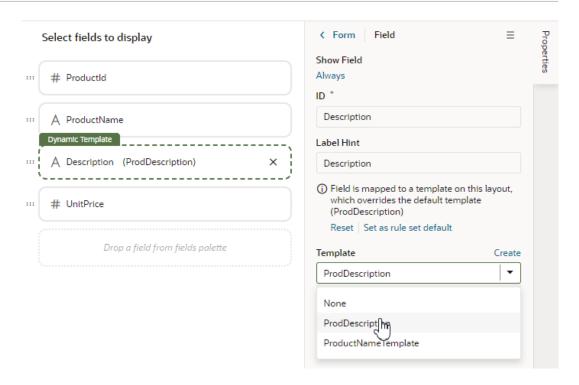
- In the Rule Sets editor, open the layout you want to work on and select the field in the center pane.
 - The center pane of the layout editor lists the fields that will be displayed in the layout and the templates that are applied to them. If you duplicated an existing layout, your new layout might already list some fields, or have templates already applied to fields.
- 2. Select a template from the Template dropdown list in the Properties pane.
 - If no template has been applied to the field (as in this image), you can select a template in the list.



If the field already has a field template applied to it (you'll see the template name next to the field name), you can still select a template in the list unless a default template has been defined for the field. You'll see a notification in the Properties pane and an Override button if a default field template has been defined.



If a default template has been defined for the field, click **Override** in the Properties pane, and then select the template from the dropdown list.



At any time you can click **Reset** in the Properties pane if you want to re-apply the default template.

When you apply a template to a field, it's only applied to the field in the current layout. If you want the template applied to the field in every layout in the rule set, click **Set as rule set default** in the Properties pane, or open the template in the Templates tab and set it in the template's Properties pane.

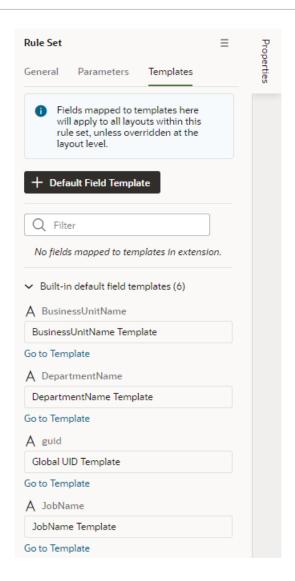
Set the Default Template for a Field in a Layout

In addition to applying templates to fields in individual layouts, you can set the default field template that will apply to the rule set. The default field template will be applied to the field in every layout in the rule set. After setting a default field template for a field, you'll need to override it if you want to change the field's template in an individual layout.

To define a default field template for a field in a rule set:

1. Open a rule set in the editor, then open the Templates tab in its Properties pane.

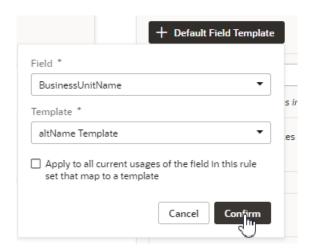
The Templates tab lists all the fields in the layout that have a default template applied to them. A field's default template might be defined in the Layout in the Unified Application (Built-in default field templates) or in the extension.



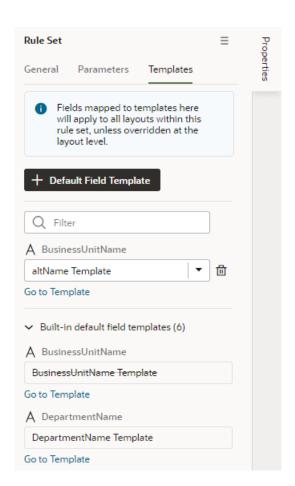
- 2. Click + Default Field Template.
- 3. Select the field that you want to apply the template to.

You can select a field that already has a default field template applied to it if you want to override it.

4. Select the template that you want to apply to the field. Click **Confirm**.



In the Templates tab, the new default field template mapped to the field is added to the list of default field templates defined in the extension. In this image, in the list of Built-in default field templates, the BusinessUnitName template is crossed out to show it's been overridden by a template defined in the extension. You can remove any default field template mappings that you create by clicking in ext to the mapping in the list.



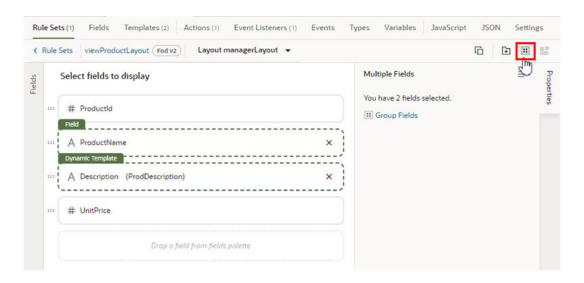
Add and Group Fields in Dynamic Form Layouts

When creating a layout for a dynamic form in the Rule Sets editor, you can group fields so that they are displayed together in a layout, so you can treat them as a single entity. For example, you might create an address group that contains the fields (for example, name, address, city, state, country and post code) that you want to be displayed as a group in your layout. 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:

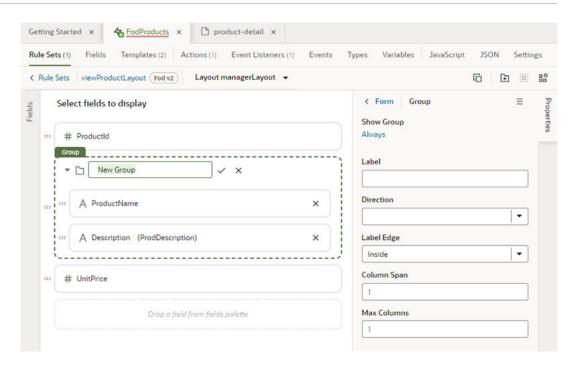
- In the Rule Sets editor, select the rule set of the dynamic form you want to work with, and then open the layout you want to edit.
- Select the fields that you want to group together, either by holding down the CMD key (on macOS) or the Ctrl key (on Windows), and then click **Group Fields** in the Properties pane, or in the toolbar.



The selected fields are grouped under a new folder in the layout diagram.

Type a name for the new group by naming the folder in the layout editor. Click ✓ to save the group name.





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.

Add a Link to a Page in an App UI

When extending a dynamic component such as a dynamic form, you use field templates when you want to add links to pages in your extension or in other App UIs.

For example, let's say a page in one of your extension's dependencies contains a dynamic form, and you want to add a link in the form that navigates to an App UI page in your extension. To do this, you could create a field template that contains the link, and then create an action chain that navigates to your extension's page when that link is clicked.

- 1. Create the field template where you want to add the link.
- In the template editor, drag an icon component from the Components palette onto the canvas. This icon will be the link.
- In the component's Properties pane, open the Events tab, and then click + Event Listener. Click On 'click' in the dropdown list.

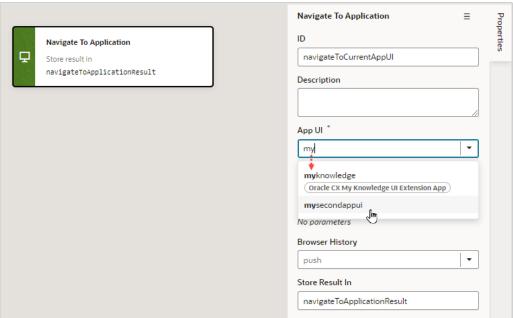
Selecting On 'click' will add a component event to the component, and will create a new action chain that will be triggered by the event.

The new action chain opens in the editor in the Layout's Actions tab.

- 4. In the action chain editor, drag the Navigate action from the palette onto the action chain.
- Select the Navigate action in the action chain.
- 6. In the action's Properties pane, select the page you want to link to:
 - a. Select App UI (if not selected).
 - b. In the App UI dropdown list, select the App UI containing the page you want to link to.



The App UI can be an App UI in your extension, or in one of your extension's dependencies.

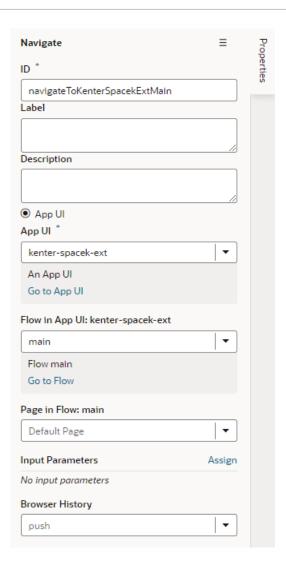


In the dropdown list, the label next to the App UI name displays the name of the extension where the App UI is defined. In the image above, the kenter3_ext App UI has no label because it's defined in your extension, not in one added as a dependency.

Note:

In addition to App UIs in your extension, the dropdown list will contain the App UIs in your dependencies that are navigable by other App UIs, if any. This property is set by the App UI developers in the page- and flow-level Settings editors, so you can't change this setting in your dependencies. For more, see Navigate From Fragments and Layouts to Other App UIs.

- **c.** Select the flow and page in the App UI you want to navigate to, if available.
 - If the App UI's developer has set any pages and flows as navigable, you'll be able to select them in the dropdown lists. If you can't or don't select a flow and page, navigation will be to the default page in the App UI's default flow.



7. Apply the template to the field in your dynamic form.

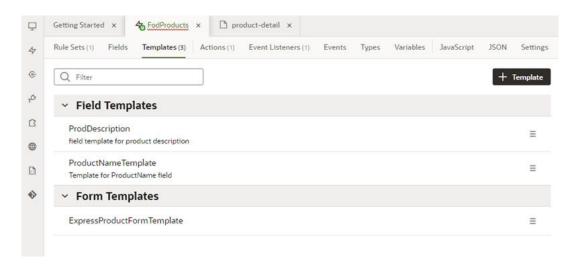
Create Templates for Form Layouts

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. The template can be one of the templates defined in a dependency, or it could be your own form template. For example, you might have a page that uses a dynamic form to display a detail view that includes contact details, 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 'Leads' form template that includes the Rating Gauge component, and then apply the template to the form. You can re-use the template in multiple form layouts in a Layout's rule sets, but templates can't be shared between Layouts.

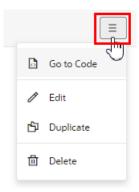
To create a form template for a dynamic form:

Open the Layout's Templates tab.

The Templates tab displays a list of field and form templates that are already defined for the Layout. Templates that are defined in a dependency will have a badge that displays the name of the dependency.



If you want to duplicate a template that you've created, open the template's menu and select Duplicate. You can also use the menu to edit, delete, and view the code of templates you've created.



You can also duplicate templates defined in dependencies, but only if the template only contains a fragment (or fragments).

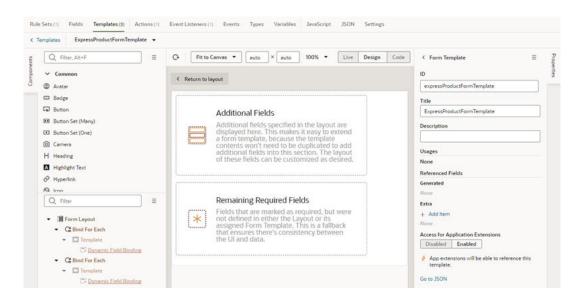
2. Click + Template and select Form in the dialog.



Specify the Label and ID. Click Create.

The new template opens in the form template editor. The form template editor contains a Components palette, Structure view, canvas and Properties pane.

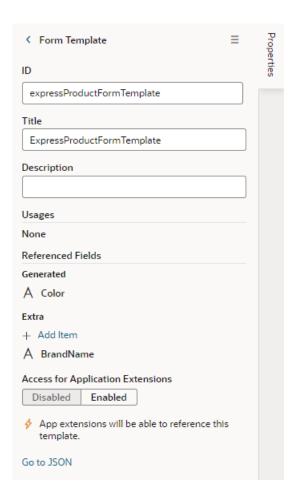
In this image, you can see that the canvas has two read-only template fields that are generated automatically: Additional Fields and Remaining Required Fields. These fields are used to display the data and display names for the fields defined in the layout. These template fields render all the fields in the layout, so you don't need to modify the template each time you change a layout.



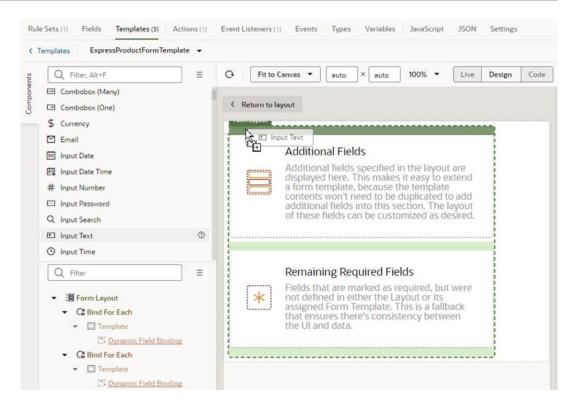
4. While Form Template is selected in the editor, click **Add item** in the Extra section in the Properties pane.

The Extra fields are fields that are defined in the template, not the layout. Each field you add in the Extras section is displayed in the form when the template is applied, and can't be removed in the layout.

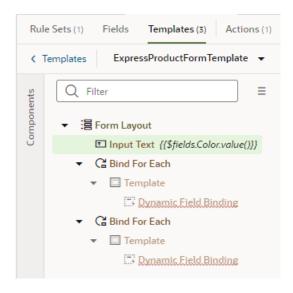
Each Extra field must be mapped to a component if you want it to appear in the form. This image shows the Properties pane after adding the BrandName field to the template as an Extra.



5. Drag the component you want to add from the Components palette and position it in the Structure view or on the canvas.

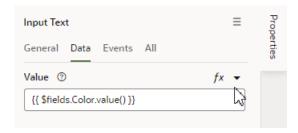


You can add components above and below the read-only template fields, but not within them. In the Structure view of this template, you can see an Input Text component that was positioned above the Additional Fields template in the Form Layout.



6. 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 f^x to open the Expression Editor, or (x) 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 To 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.

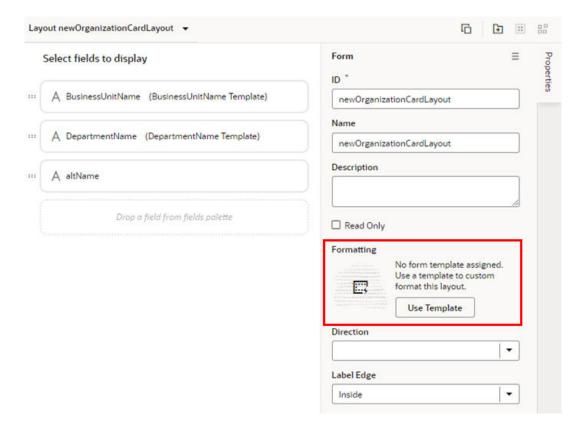
Apply a Template to a Form

To apply a form template to a form:

- 1. In the Rule Sets tab, open the rule set for the form you want to work on.
- 2. Click the name of the layout to open it in the layout editor.

The center pane of the layout editor lists the fields that will be displayed in the layout and the templates that are applied to them.

3. 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. The list of templates will include form templates defined in your dependencies as well as the templates you've created.

You can click **Create** in the Properties pane if you want to create a new form template.

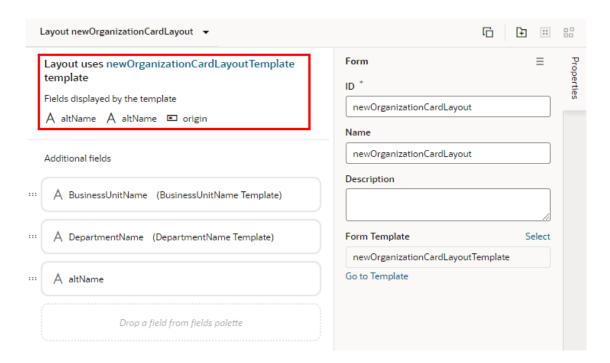
4. Select the template you want to apply in the Use Layout Template window. Click Select.

The Use Layout Template window displays the templates you can apply to your form layout. In addition to templates you've created, the window will also include form templates defined in your extension's dependencies that you can apply

You can also 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

(NewOrganizationCardLayoutTemplate) applied to the form layout, and the fields defined by the template (altName, origin).



You can't edit templates defined in dependencies, but you can edit the fields defined in templates you've created. To open a template you've created, click **Go to Template** in the Properties pane.

If you select a template that contains a field you don't want to appear in your form, you'll need to select a different template, or click **Select** to open the Use Layout Template window, and then select **No Template**.

Create Fields For a Layout

You can create fields in a Layout if you'd like to use a field that isn't defined in your Oracle Cloud Application. You can set the fields to variables, or to expressions that reference other fields available in the Layout.

Note:

Creating a field in VB Studio doesn't create a field in your Oracle Cloud Application. You need to use App Composer to create custom fields in your Oracle Cloud Application. For details, see Add Objects and Fields in Configuring Applications Using Application Composer.

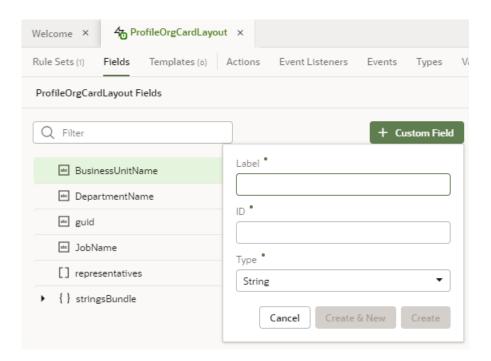
You can't create or modify the fields in your Oracle Cloud Application, or the service definition used by a Layout, but you can override some field properties, such as "Read Only" and "Required". So if a field's Required property is set to False in the service definition, you can override the property to make it more strict and set it to True. This won't change the description in the service definition, where the property will still be set to False. However, you can't override a property to make it less strict, meaning you can't set a Required property to False if it is already set to True in the service definition.

If the fields defined in the service definition don't meet your needs, you can create *calculated fields* and *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. To create a virtual field, see Create a Virtual Field below.

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 calculated field that combines these fields into a single field called <code>fullName</code>, and use that in your layouts instead. The value of this new field is <code>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.</code>

To create a calculated field:

Open the Fields tab in the Layout you want to configure.





- Click Custom Field.
- 3. Type a label for the field (the field's display name). When you type in the label field, 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.

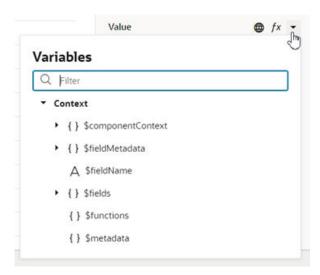
5. If you want to create an expression and use an existing field, click **Referenced Field**, then select a field in the list. Click + to add it.

You can add any field available in the Layout, including the sub-fields of fields with an object type. If you add one or more referenced fields, you won't see the Default Value property in the Properties pane. Instead, you'll see a Value property, which you use to specify the expression.

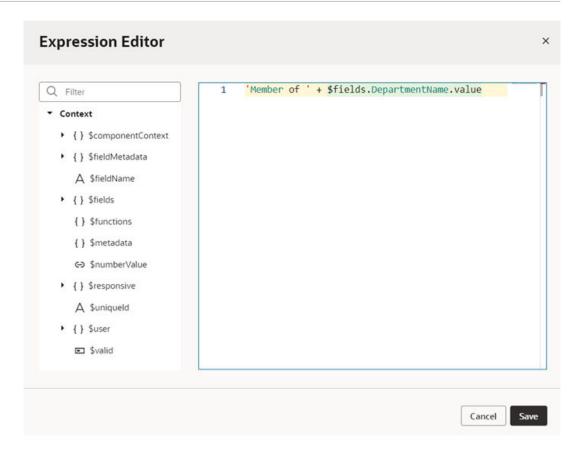
If you set a calculated field to Read Only, you set the field's value or expression in the Value field (there is no Default Value property for read-only fields).

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, you can click (x) to open the Variables picker.

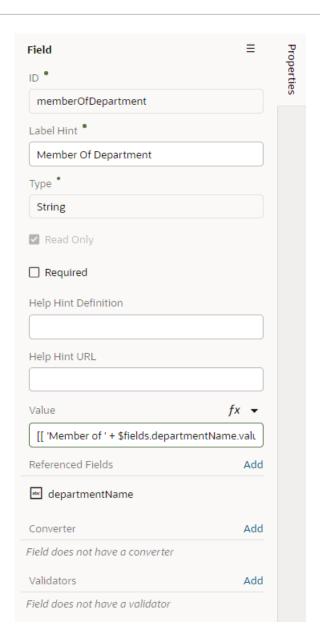


if you want to use an expression, you can click f^x to open the Expression Editor. In the Expression Editor, variables are grouped by context in the Context pane. For example, fields available in the Layout are listed under fields, and user metadata is listed under fields. You can select a variable in the Context pane to add it to your expression, or start typing in the editor to use the editor's code completion. You can also add text strings to your expression by typing in the editor. Click **Save**.



The expression you create in the editor is added to the Value field, for example,

```
[[ 'Member of ' + $fields.departmentName.value() + $fields.JobName.value() ]].
```



7. Optionally, you can click **Add** to add converters and validators to the field.

You can 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 to make sure, for example, a string in a field is not too long, or so that dates are formatted the way you want, as shown in Add Converters and Validators to Fields.

Your custom fields (and any fields that you have modified, for example, in the Properties pane) are indicated by a blue dot to the right of the field name. In this screenshot, you can see the blue dot next to MemberOfDepartment.





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 rule set layouts. For example, you can create a single field that combines several contact details that are stored in different fields in your 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.

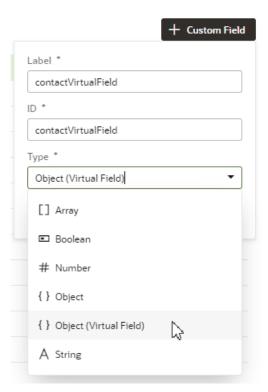
When you add a virtual field to a layout, you'll need to define a field template to display it. You'll need to create the field template if it doesn't exist. The template will contain components for each of the referenced fields that you want to display in the layout.

To create a virtual field:

- Open the Layout's Fields editor.
- Click Custom Field and type a label for the field (the field's display name).

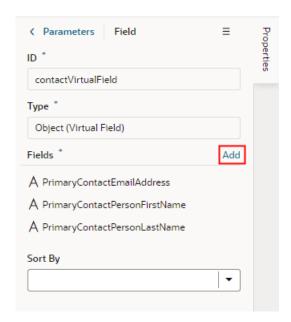
When you type in the label field, a suggested ID is generated for you. The ID can't be changed later.

3. Select the Object (Virtual Field) type. Click Create.



In the Properties pane, click Add and select the fields you want to include as reference fields.

You can add any of the available fields as reference fields, including sub-fields of objects.



Select a field in the Sort By dropdown 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).

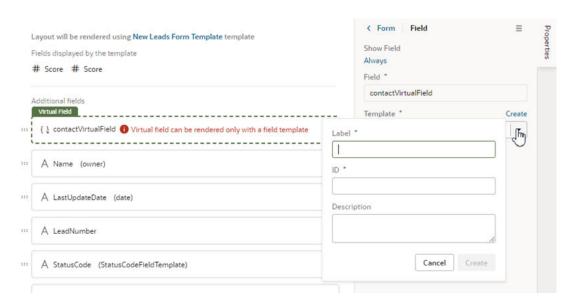
The table won't be sortable by the virtual field if you don't select a Sort By field.

- In the Rule Sets editor, open the rule set layout of the dynamic form 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, click **Create** in the Properties pane and type a name for the template in the Label field. Click **Create** to open the new template in the editor.





You need to 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 dropdown list. You'll need to create one if no template exists.

- 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.
- 10. Click Return to layout when you're finished.

The new template is applied to your virtual field.

You can add the virtual field to other rule set layouts in the Layout. When adding the virtual field, you can apply the same field template, or create additional field templates to apply to the virtual field.

Add Converters and Validators to Fields

You can add converters and validators to fields, including some built-in ones provided by Oracle JET. You might want to add a convertor 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 could also add a validator to a field to check if a value entered in it is valid, for example, to check if a date is not earlier than the current date.

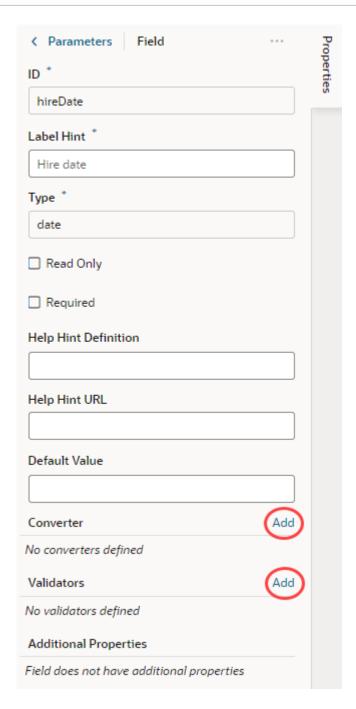
You can find details and examples in the Oracle JET Developer Cookbook:

- Built-in Oracle JET converters
- Built-in Oracle JET validators

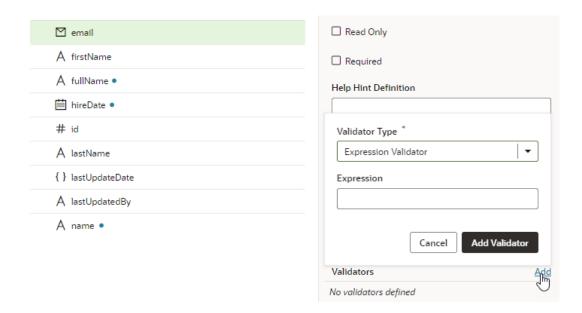
To add a converter or validator to a field:

- 1. Open the Layout's **Fields** tab, and then select the field you want to work with.
- In the field's 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:

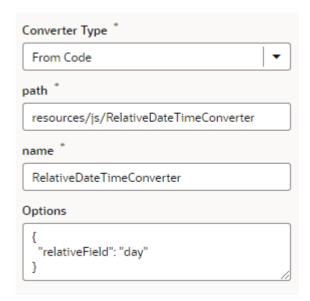


Change the type if needed, enter additional details, then click Add Validator or Add Converter.

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 in the UI. For the Length Validator shown here, the options specify how to count the characters and the minimum and maximum string lengths allowed:

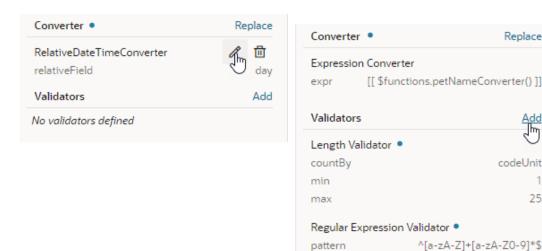


You can also create your own validator or converter by selecting the **From Code** option. With this type, the **path** field specifies the location of a JavaScript file that implements the custom validator or converter; the **name** field specifies the name of the constructor; and the **Option** field specifies the options specific to the custom validator or converter, for example:



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 relative Field.

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 4, 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.



Use Context Parameters in Extensions

In addition to built-in system variables that can be used within a dynamic component (for example, device size or user role), the Unified Application's developer at Oracle or a dependency's developer can define context parameters that might otherwise be inaccessible to a dynamic component.

25

Dependency developers and Unified Application's developers at Oracle may choose to define context parameters for:

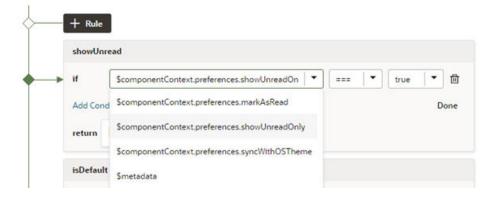
- Values produced by your application that come from outside the component, such as page variables,
- Details from other parts of the application,
- Other values that might be useful when extending the application.

You might see parameters in these three contexts in your extension:

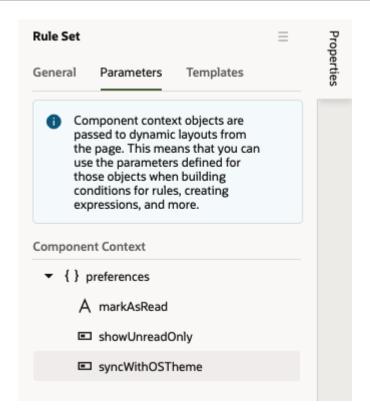
Context	Description
\$componentContext	Parameters in the \$componentContext can be accessed in a dynamic component's Expression Editor and in the rule set condition builder.
	You might also see \$componentContext parameters in business rule conditions when no \$ObjectContext parameters have been defined.
\$baseComponentContext	Parameters in the \$baseComponentContext can be accessed in a dynamic component's Expression Editor and in the rule set condition builder.
	They can also be used to pass information to service definitions, for example, to add parameters to the query used to call your Cloud Application service.
\$objectContext	Parameters in the <code>\$objectContext</code> can be used in the condition builder for business rules and validation messages.
	Parameters can also be used when creating conditions in the advanced expression builder, and in expressions setting the default values in business rules.

For example, there might be some preferences defined by the app that could be useful when creating a condition in a rule set. When you create the condition, you can select the parameters in the \$componentContext in the condition builder's Attributes dropdown list. As you can see in this image of the Attributes dropdown list, \$componentContext is prepended before the parameter names.

Display Logic ®



You can see the list of valid \$componentContext parameters in the Parameters tab in the Properties panes of a rule set:

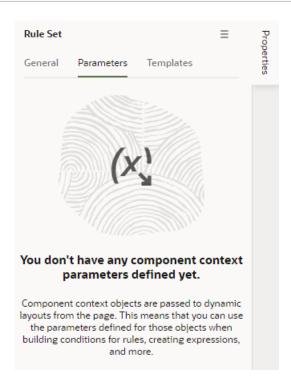


The Parameters tab shows the name and type of each parameter. To see a parameter's description and the valid values, move your cursor over the tooltip icon which is displayed when you hover over the parameter name.

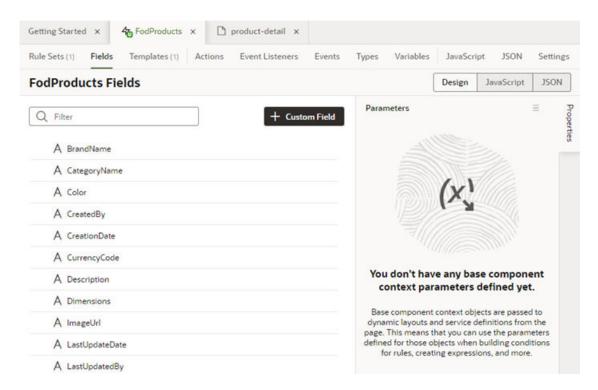


If you enter an invalid parameter value in the Expression editor or condition builder, a warning will be displayed in the layout's JSON file. To write efficient expressions that handle situations where a referenced field might not be available or the field's value could be null, see How To Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be Null.

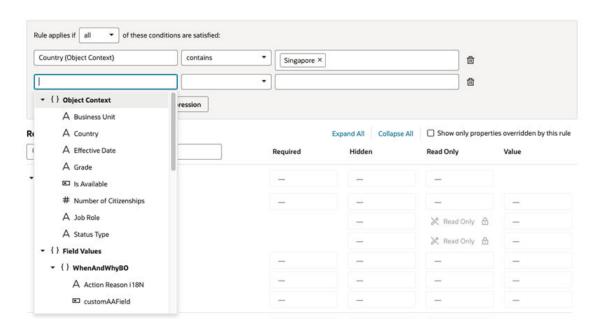
If you don't see any parameters in the Parameters tab, this means that the Unified Application has not defined any \$componentContext parameters for the component:



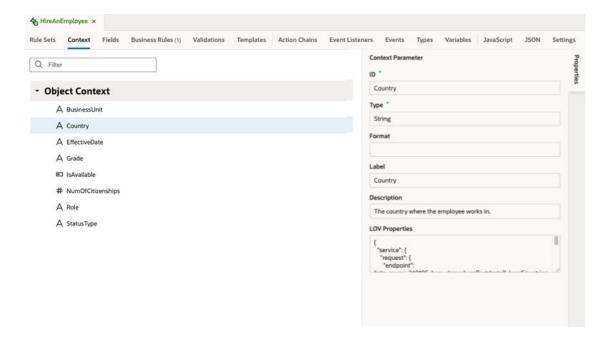
The \$baseComponentContext parameters are listed in the Parameters pane of a Layout's Fields tab when no field is selected:



When creating conditions for business rules, you can select <code>\$objectContext</code> parameters in the condition builder's criteria dropdown list. As you can see in this image, the parameters are grouped under Object Context in the dropdown list:



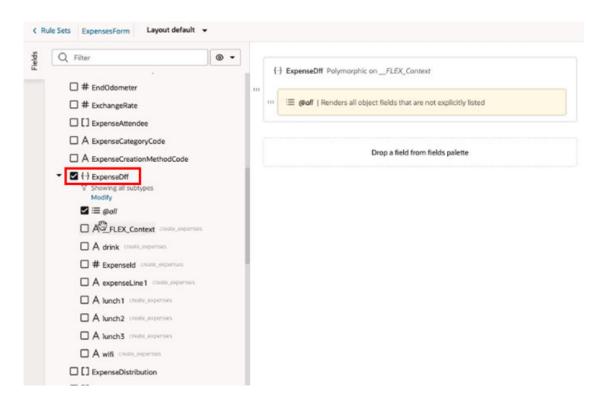
The \$objectContext parameters and their properties are listed in the Context tab:



Work with Polymorphic Objects in a Layout

When you are adding fields to a layout, the Fields palette might include *polymorphic* objects containing fields you can select. Polymorphic objects are defined by the service that your extension uses, and define a set of fields rather than a single field. A polymorphic object can display different field subsets based on a pre-defined *discriminator* sub-type that is evaluated at runtime. The polymorphic object might have several sub-types for the discriminator field, each defining a different set of fields.

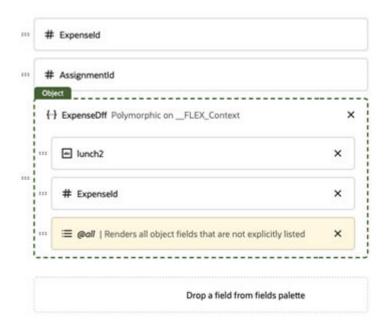
In the image below, ExpenseDff is a polymorphic object, and __FLEX_Context is the discriminator field.



An <code>@all</code> field is automatically added when you add a polymorphic object to a layout, but you can also explicitly add fields to the center pane. When the center pane contains the <code>@all</code> field, all the fields, as determined by the discriminator, will be displayed in the layout. You can control the order that fields are displayed in the polymorphic object by explicitly adding fields to the object in the center pane.

In the example above, the discriminator has a sub-type Lunch that determines the fields that will be displayed (Expenseld, expenseline1, lunch1, lunch2, lunch3). All of those fields will be displayed when the Lunch sub-type is applied at runtime because the polymorphic object in the center pane includes the @all field. If you remove the @all field from the center pane without explicitly adding any fields, no fields will be displayed in the layout.

By explicitly positioning some of the fields (lunch2, Expenseld), as in the image below, you can control the display order of the fields.



The fields will appear in the following order when the Lunch sub-type is applied to the discriminator:

- lunch2 (added and positioned explicitly)
- Expenseld (added and positioned explicitly)
- expenseLine1 (added using @all)
- 4. **lunch1** (added using @all)
- lunch3 (added using @all)

If the @all field is removed, only the fields added explicitly (lunch2, ExpenseId) will be displayed when the Lunch sub-type is applied.

If the wifi field was added explicitly to the object, it wouldn't be displayed when the Lunch subtype is applied to the discriminator.

To add a polymorphic object to your layout:

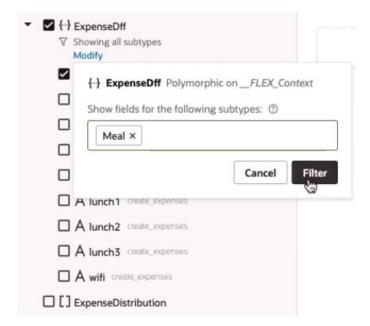
- 1. In the Rule Sets editor, open the layout you want to work on.
- 2. Select the polymorphic object in the Fields palette, or drag it from the palette into the list in the center pane.

The Fields palette lists all the fields and objects you can add to your layout, including polymorphic objects. Polymorphic objects in the palette have a special object icon (\(\frac{1}{2} \)). You can expand the object node in the palette to see the sub-fields that could potentially be displayed if you added the object to the layout.

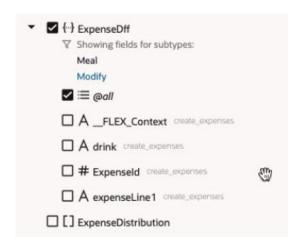
If you want to see which fields you can add to the center pane, for a specific sub-type:

- a. Expand the object node in the palette.
- **b.** Click **Modify** and select a sub-type from the list. You can also broaden the filter by selecting more than one sub-type.
- c. Click Filter to filter the fields displayed under the object node.





In the image below, the sub-fields of the polymorphic object <code>ExpenseDff</code> are filtered by the <code>Meal</code> sub-type. Instead of listing all the object's sub-fields, only the sub-fields defined by the <code>Meal</code> sub-type (<code>drink</code>, <code>ExpenseId</code>, <code>expenseLine1</code>) are listed under the object node. These are the fields that will be displayed when the <code>@all</code> field is added to the center pane and the <code>Meal</code> sub-type is applied.



By seeing the fields defined for each sub-type, you can decide which fields you might want to add to the center pane, if any.

3. Arrange the order that fields are displayed in the component by dragging fields into position in the center pane.

You can control the order that a polymorphic object's fields are displayed by adding them explicitly and positioning them where you want.

Control Your Display with Business Rules

Whether you're a functional administrator tailoring your end users' interactions or an end user yourself, *business rules* enable you to create rules to control how regions and fields in

components are displayed at runtime, based on conditions such as the user's role and the employee's business unit or legal employer.

When the dependency developer has enabled business rules for a page, you can easily control what's displayed on the page as a whole, instead of configuring page components such as tables and forms individually. For example, you could create a single rule to conditionally hide every occurrence of a field on the page, in every form where it appears. If you were to do this using rule sets, you would need to configure the rule set for each form to hide the field.

If you don't see the Business Rules pane in your view of the Properties pane, that means your page is not enabled for business rules, but you can still use business rules to override field properties in components. In addition to a rule sets editor, each Layout has a business rules editor where you can configure rules for fields in the rule sets. You can use the rule sets editor as described in Control Your Display with Rule Sets to configure your components, and then use business rules to override field properties.

Before we dive into how to use business rules, it's important to realize that you work with business rules in the context of an *extension*. If you're not familiar with extensions, you should probably read both What Is an Extension? and What's the Extension Lifecycle?, at a minimum.

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.



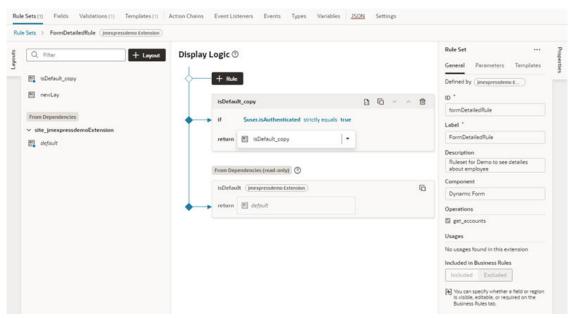
If you're looking for information on business rules as they pertain to business objects, see Create Rules for Business Objects.

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.

In Visual Builder Studio (VB Studio), all the elements that control how dynamic tables and forms are rendered on a page are configured in a *Layout*. Some Layouts are governed by rule sets (note the Rule Sets editor tab at the far left) which influence the appearance or behavior of dynamic tables or forms through the use of display logic. Within the rule set, each rule optionally defines a condition (such as, "Is the current user a manager?"), plus a *layout* to apply to the page when that condition is true. A layout might hide a field in a dynamic form, show a field, or otherwise impact the page's appearance or behavior.

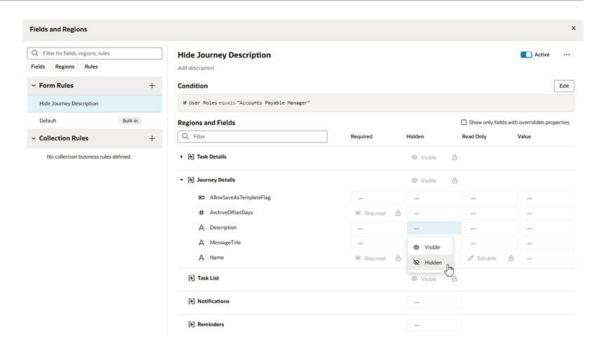




For certain Oracle Cloud Applications, however, Oracle chooses to also employ Layouts defined by *business rules*, which can be used with, and even instead of, rule sets. Conceptually similar to rule sets, business rules can handle simple and complex display needs that might otherwise require tens, if not hundreds, of individual layouts if the rule set approach was used instead.

For example, suppose you had a data object called Person that was used in two dynamic forms. Each of these correspond to a different rule set on the Rule Sets tab. 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 rule sets, you'd have to add a new rule to **each** layout with the condition "if user=manager", plus a new layout that hides the **Organization Name** field when the condition is true. With business rules, on the other hand, 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 region that includes that field.

In other words, business rules let you define a rule at the *object* level. Let's look at a page with several dynamic forms. These are listed in the Regions and Fields section in the image below. Some components use the same data object, and you can quickly see every field used in each component. When you set a property in a business rule you are setting it at the object level, so it can be applied to every occurrence of that field, in each form or table 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. Form and collection rules defined in a dependency are read-only, and you can identify them by their *Built-in* label visible in the list of rules. For more, see Create a Rule For Forms and Create a Rule For Tables. In this example, the Default form rule is a built-in rule defined in an extension dependency.

When you select a built-in rule in the list of rules, you can see the name of the dependency where the rule is defined to the right of the rule name in the editor.

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.

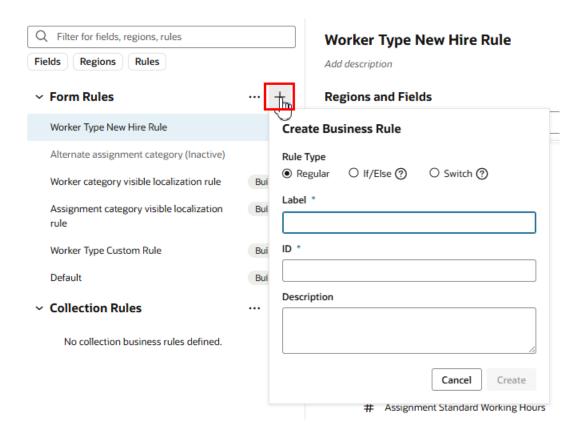
Create a Rule For Forms

A form rule allows you to override certain properties for a dynamic form, assuming that certain conditions are met at runtime.

To create a form rule:

- Open the page you want to configure, and then click Configure Fields and Regions in the Business Rules pane to open the business rules editor.
- In the business rules editor, click to Porm Rules to open the Create Business Rule dialog box.

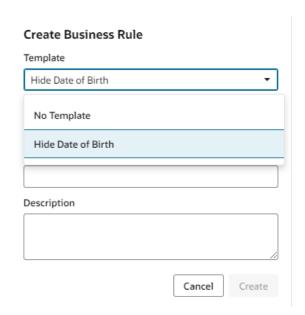




3. Do one of the following.

- Select the Rule Type in the dialog box. Select Regular to create a standard business rule. The other rule types (If/else and Switch) are special types of nested rules. For more about these nested rules, see Add an If/Else Rule and Add a Switch Rule.
- Select a rule template.

 If there are any business rule templates defined in the app, you can select one using the Template dropdown list in the Create Business Rule dialog box:





A template can provide examples of how to configure business or validation rules, for example, how to specify rule conditions, how to write advanced expressions, or examples of how to override properties.

When you create a rule from a template, the new rule contains all the examples in the template. You can edit the examples in the new rule as needed.

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 extension rules under Form Rules.

Rather than creating a rule from scratch or from a business rule template, you can duplicate an existing rule and use it as the basis for a new rule. To duplicate a rule, right-click the rule in the list, and then click **Duplicate**.

Note:

When duplicating rules defined in extension dependencies (the rules labeled "Built-in"):

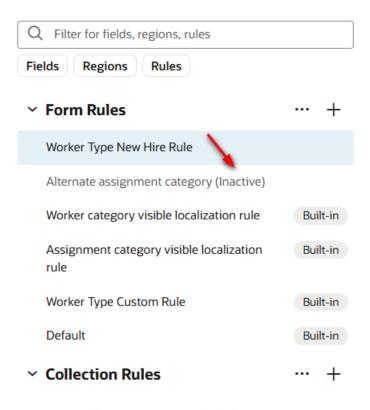
- You cannot duplicate a rule if it uses advanced expressions, for example, in the rule's condition or to override a field's value.
- You cannot duplicate a rule if it uses any global functions that cannot be referenced in the extension. If the global functions are referenceable, when the rule is duplicated they are automatically imported, and the references to the function are updated in the rule.
- When duplicating nested rules, you can only duplicate the top-level rule. When you duplicate the rule, the child rules are also duplicated.

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, so all extension rules are evaluated after the built-in rules. As you create more rules, make sure you position each one in the order you want them evaluated. To change a rule's position, right-click the rule and use the Move up and Move down items in the popup menu. To 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 "(Inactive)" is appended to the name, like this:





No collection business rules defined.

Inactive rules are not included in the rule evaluation process.



You can deactivate all the extension rules at once by clicking the three dots next to the Form Rules and Collection Rules headings, then clicking **Deactivate All**. This can be useful when debugging a page, allowing you to see the page with only the built-in 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.

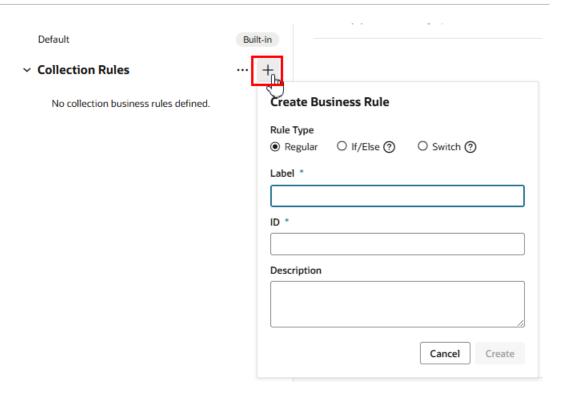
You cannot deactivate built-in rules, as those have been locked into place by Oracle. In other words, a page's out of the box behavior is determined by the default rule, **plus** any built-in rules that may exist.

Create a Rule For Tables

You use a collection rule when you want to hide or display columns in a dynamic table, assuming that certain conditions are met at runtime.

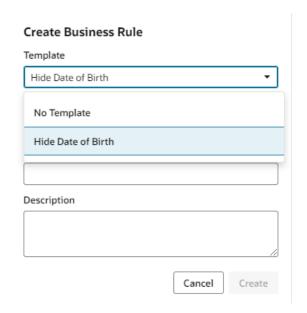
To create a collection rule:

- 1. Open the page you want to configure, and then click **Configure Fields and Regions** in the Business Rules pane to open the business rules editor..
- 2. Click + next to Collection Rules to open the Create Business Rule dialog box.



- 3. In the Create Business Rule dialog box, do one of the following.
 - Select the Rule Type in the dialog box.
 Select Regular to create a standard business rule. The other rule types (If/else and Switch) are special types of nested rules. For more about these nested rules, see Add an If/Else Rule and Add a Switch Rule.
 - Select a rule template.

 If there are any business rule templates defined in the app, you can select one using the Template dropdown list in the Create Business Rule dialog box:



A template can provide examples of how to configure business or validation rules, for example, how to specify rule conditions, how to write advanced expressions, or examples of how to override properties.

When you create a rule from a template, the new rule contains all the examples in the template. You can edit the examples in the new rule as needed.

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.

Click Create

Your new rule is added to the top of the list of extension rules under Collection Rules.

6. Click **Edit**, and then create a condition for the rule.

See Set Conditions for a Rule.

7. 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.



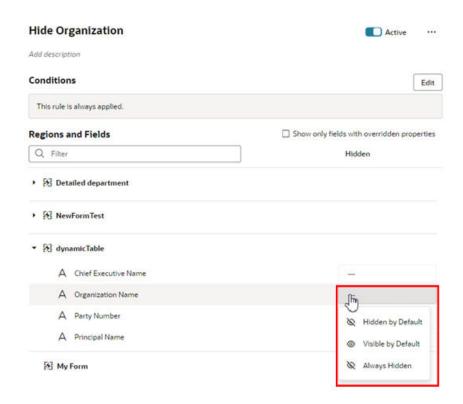
If you have trouble locating the field you want to modify, confirm that there were no errors loading services in the editor. The editor only lists fields from services that have loaded successfully. You'll see "Loading Completed with Errors" in the editor's status bar if any errors occurred when loading the editor.

If you see the error message in the status bar and you want to try to locate the cause, see Resolve the "Loading Completed with Errors" Message in the Troubleshooting chapter.

8. 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:





The Hidden property has three options:

- Hidden by Default hides the table column when the page is rendered. A user can still
 use the table's popup menu to hide and show the column.
- **Visible by Default** displays the table column when the page is rendered. A user can still use the table's popup menu to hide and show the column.
- **Always Hidden** hides the table column. When this option is selected, a user cannot use the table's popup menu to display the column.

In a collection rule, you can set the Hidden property for a field even if it's not included in a table. The Hidden property in a collection rule is only applied to table columns. (The property does not affect fields in forms.)

Depending on how the app was configured, you might have options for filtering the list of fields displayed in the editor. The **Show only fields used by layouts** option filters out the fields not included in layouts. If you don't select this, the list contains all the object's fields. The **Show only fields with overridden properties** option limits the list of fields to those that have been modified by a rule.

Rather than creating a rule from scratch or from a business rule template, you can duplicate an existing rule and use it as the basis for a new rule. To duplicate a rule, right-click the rule in the list, and then click **Duplicate**.



Note:

When duplicating rules defined in extension dependencies (the rules labeled "Built-in"):

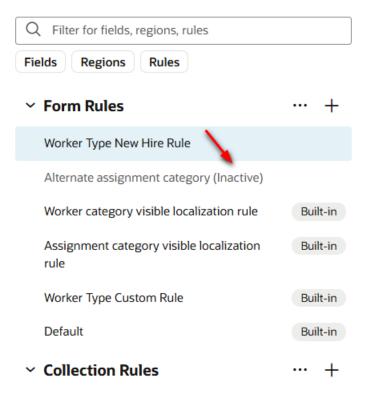
- You cannot duplicate a rule if it uses advanced expressions, for example, in the rule's condition or to override a field's value.
- You cannot duplicate a rule if it uses any global functions that cannot be referenced in the extension. If the global functions are referenceable, when the rule is duplicated they are automatically imported, and the references to the function are updated in the rule.
- When duplicating nested rules, you can only duplicate the top-level rule. When you duplicate the rule, the child rules are also duplicated.

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, so all extension rules are evaluated after the built-in rules. As you create more rules, make sure you position each one in the order you want them evaluated. To change a rule's position, right-click the rule and use the Move up and Move down items in the popup menu. To 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 "(Inactive)" is appended to the name, like this:





No collection business rules defined.

Inactive rules are not included in the rule evaluation process.



You can deactivate all the extension rules at once by clicking the three dots next to the Form Rules and Collection Rules headings, then clicking **Deactivate All**. This can be useful when debugging a page, allowing you to see the page with only the built-in 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.

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:
- 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.
- 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.

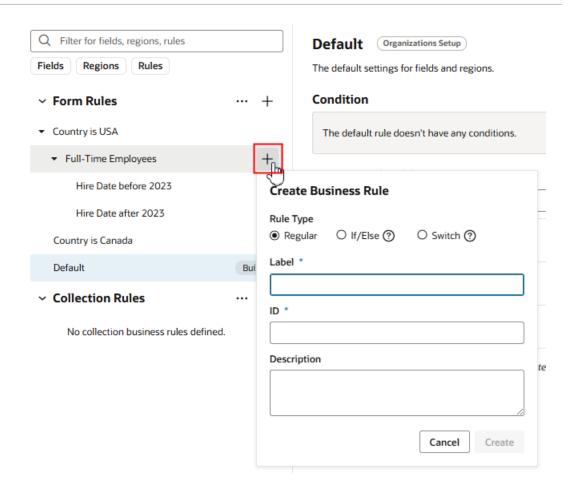
Although you could write a more complex set of conditions to achieve the same outcome, nested rules make it much simpler to apply multiple property overrides at runtime.

Add a child rule

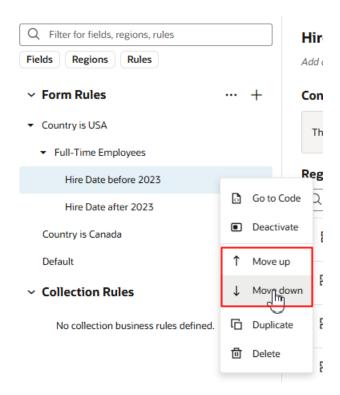
You can add a child rule to any of the rules you've created:

Click 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**.
- 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.

Note:

You cannot duplicate rules defined in extension dependencies (the rules labeled Built-in), which includes the Default rule.

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.

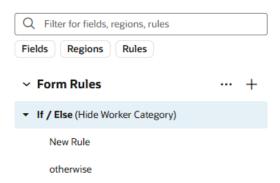
You use an If/Else nested rule when only one of the rules in a list of rules needs to be applied. Unlike regular business rules, where every rule is evaluated, the child rules in an If/Else rule are evaluated in order until one is applied; the unevaluated child rules are skipped. This means that rules won't be evaluated if they don't need to be.

For example, you might have a rule that should be applied when the worker category is Pending Worker, and some rules that should only be applied when the category is New Hire. 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.

- 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.
- 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.



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 when the worker category is Pending Worker.
- **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.

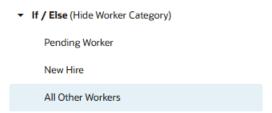
If you don't set any conditions, the rule is always applied if it's evaluated. If a rule has no conditions you should make sure it's the last rule in the list.

Let's define this rule so that it's applied when the worker category is New Hire.

- **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 all other worker categories, and move the rule below the New Hire 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 worker category is not Pending Worker or New Hire.

Add a Switch Rule

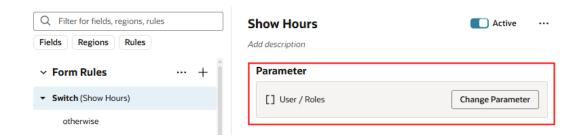
A Switch rule is a special type of nested rule where instead of evaluating a list of rules, a switch parameter (switchOn) is evaluated, and the parameter's value determines which case is applied.

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 Payroll Manager, you want the Payroll 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 Payroll 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).

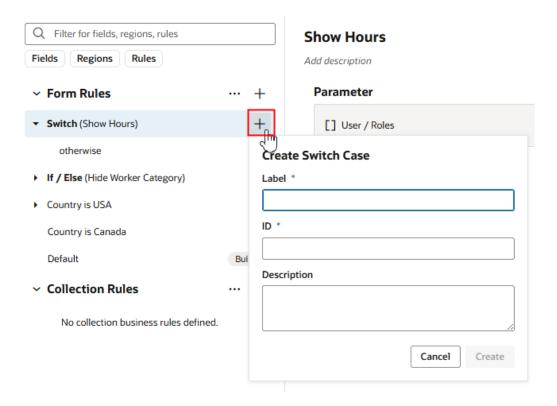


Create cases.

Let's create a case that is applied when the user role is Payroll Manager.

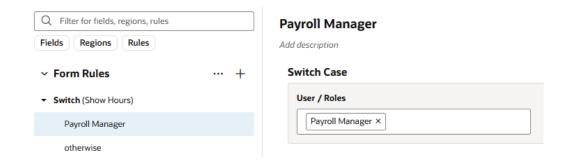
a. Click + next to Switch to open the Create Switch Case dialog box.





- **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 Payroll Manager, so that the case is applied when the user role is Payroll Manager.



- 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.

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.

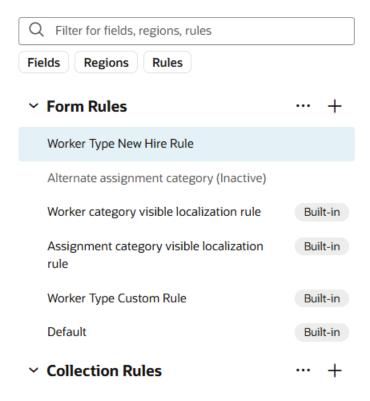
The last rule is applied if none of the cases have been applied. You cannot create conditions for the otherwise rule.

Filter Your Rules

To help you manage your display and find things quickly, VB Studio offers two Filter fields: one at the rules level and one for regions and fields.

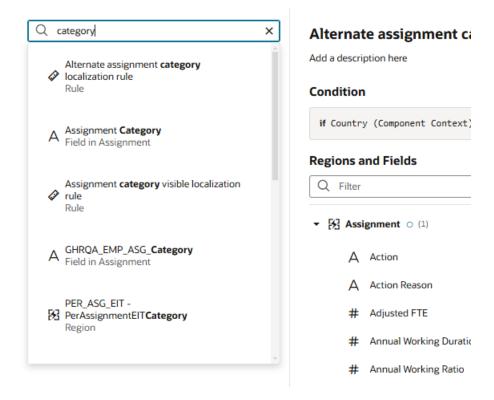
If you have hundreds of rules, it can be painful to scroll through laboriously to find the one you're interested in. Instead, use the Filter field to zero in on the rule you want.

Suppose you have a display that looks something like this, only with many more rules:

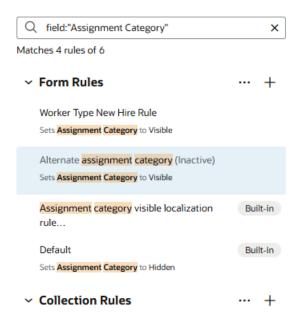


No collection business rules defined.

Simply by typing "category" in the Filter field, you can see the rules, regions, and fields containing that word, thanks to the Filter field's auto-complete feature:



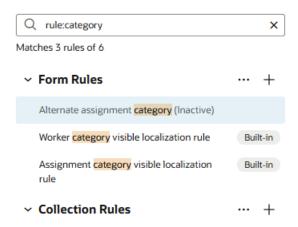
As you can see in the above image, an icon appears next to each search hit to indicate the type of entity containing the word: region, rule, or field. The list also contains a magnifying glass icon you can select to list all rules where the word occurs. In the image above, when you select the Assignment Category field, the list of rules is filtered to only include those that override a property of the Assignment Category field:



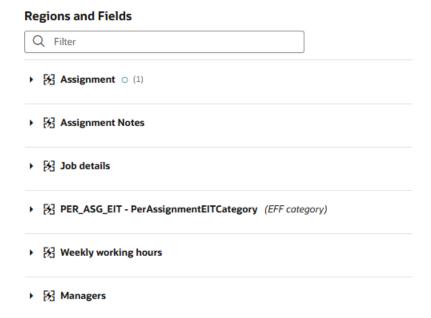
This makes it easy to tell at a glance where the Assignment Category field has been overridden, and to what property value.

The buttons beneath the Filter field (**Fields**, **Regions**, **Rules**) let you limit the scope of your search, making your filter operation that much more efficient. For example, when you click

Rules, and then enter a term like "category" in the Filter field, the filtered list shows the rules that contain the term in the name:

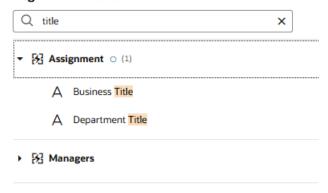


The Filter field in the Regions and Fields section works similarly, but with a slight difference. If you don't have a region expanded in the display, but the filter criteria matches a field within that region, you'll see the region, but you won't see the field until you expand the region. For example, suppose you don't have any regions expanded, like this:



You're looking for fields with the term "title" so you enter that in the Filter field. The Assignment and Managers regions are displayed, but the other regions are hidden. It's not until you expand the Assignment region that you see the fields containing the term "title":

Regions and Fields





When using the region and field Filter field, it doesn't matter which rule is currently selected.

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, or a rule that is applied when a form field is set to a specific value. In some pages, for example, when a page has a form for updating a record, you might need to add conditions to ensure that all the required details are entered in the form.

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.



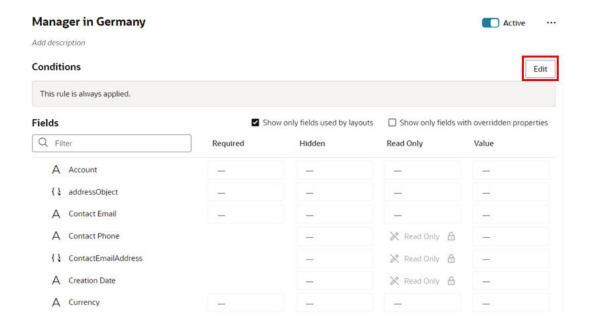
Note:

When using the Roles criterion (\$user.role) in a condition, start entering the role name you need in the Value field. The Value drop-down will display up to 25 matching Oracle Cloud Applications job and abstract roles; keep typing until the desired role displays. (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.)

If the Roles drop-down list is empty, it might be because you haven't been assigned the correct role or privileges to see the roles. When extending a Human Capital Management (HCM) application, to see the roles in the list, you need to be granted the PER_REST_SERVICE_ACCESS_USERS_AND_ROLES_LOVS_PRIV privilege, or assigned a role granted that privilege (for example, Employee (ORA_PER_EMPLOYEE_ABSTRACT). Contact your instance administrator if you require this user privilege.

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:

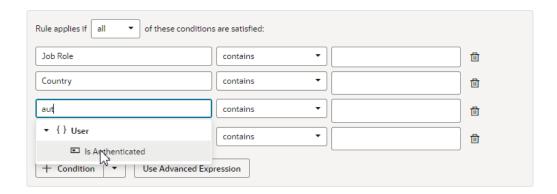
a. Select a criterion from the dropdown menu.

You can choose any of the listed criterion in your conditions, but you cannot change the list of available criteria to add your own.

Some fields in the list of criteria might be grouped under Object Context or Component Context. These are fields the extension developer has explicitly selected as useful

when creating conditions. You can view the list of the Object Context fields and their definitions in the Context tab. See Use Context Parameters in Extensions.

If you know what criteria and values you're looking for, you can try typing in the field to filter the list:



Select an operator.

The options available in the operator dropdown menu are pre-defined based upon the criterion's type. For some criteria, there might be several available operators, including operators to "match" or "not match" a JavaScript regular expression set as a value. For other criteria, such as a boolean type 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.

Type a value or simple expression in the field, or select a value from the dropdown list. Depending on the criterion, you might be able to choose multiple values.

To compare the criterion's value to the value of another field, you can select another field with the same type in the dropdown list.

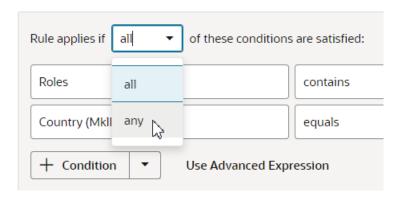
If you want to view or edit the condition as an expression, click **Code** to switch to a code editor. This code editor doesn't have all the features of the advanced expression builder, but it's useful for creating simpler expressions that you can't create with the basic condition builder:





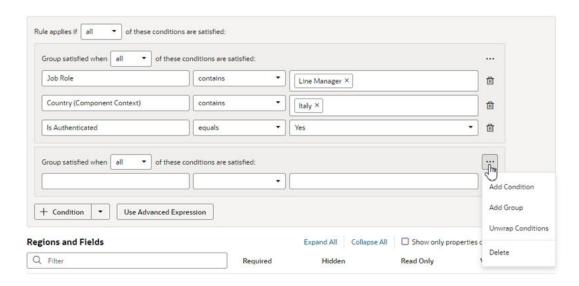
3. To add another condition, click **+ Condition**, and then define the new condition. To remove a condition, click ...

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.



Keep in mind that different Oracle Cloud Applications actions (or pages) can interpret conditions differently, which is outside of the realm of VB Studio's knowledge or control. For example, suppose the Country condition is used for a given page in Human Capital Management (HCM). The Hire an Employee action may interpret this as the person who is being hired, while another action might interpret this as the person viewing the page. To understand how conditions are interpreted for a given Oracle Cloud App, consult that App's documentation.

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**.

The Initial Field Value is set in the page's metadata. The value might be set by the page developer, but you can also edit the page's metadata in the JSON editor to set a value when extending a page. The initial value set in the metadata isn't affected by changes made by rules or users. If the field is a number and no initial value is set, the value is 0; if the field is a string, the value is an empty string.

The Field Value, on the other hand, is the current value displayed in a page. This value may already have been modified by a rule or user. For example, say the value for the Head Count field is set to 50 in the page's JSON (the Initial Field Value). 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 still 50.

Let's look at how to set the Head Count field's initial value in a page, and then add a rule that makes the Head Count field read only when the field's initial value is greater than 100.

- 1. Open the page you want to configure in the Designer, and then open the page's JSON editor.
- 2. Locate the "metadata" element in the JSON.

The metadata element for a form might look something like this.

```
"metadata": {
    "hCMContactsCreateHcmContactsNamesMetadata": {
        "type": "vb/DynamicLayoutMetadataProviderDescriptor",
```

```
"defaultValue": {
     "endpoint": "site_kenter:HCMContacts/create_hcmContacts-names"
     }
}
```

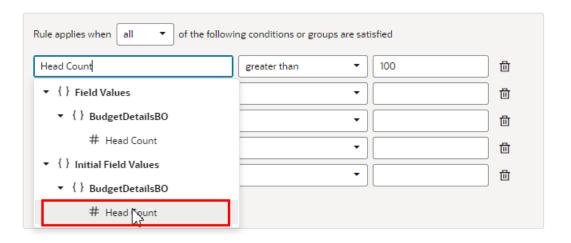
3. Add an initial value for the field.

To add an initial value of 101 for the Head Count field, add something like this (in bold) to the JSON:

```
"metadata": {
    "hCMContactsCreateHcmContactsNamesMetadata": {
        "type": "vb/DynamicLayoutMetadataProviderDescriptor",
        "defaultValue": {
            "endpoint": "site_kenter:HCMContacts/create_hcmContacts-names",
            "initialValue": {
                "HeadCount": 101
            }
        }
    }
}
```

- 4. In the business rules editor, create a new form rule.
- 5. 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**):



Set the Read Only property of the Head Count field to Read Only.

In this example, if the rule is applied in the page, the Head Count field is read only.

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.



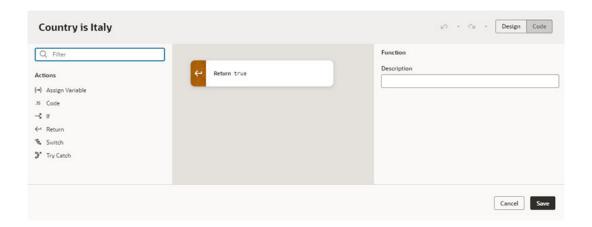
If you use advanced expressions in the condition builder, you will no longer be able to use the basic condition builder. If you have already defined some conditions in the basic condition builder, they will be displayed as actions in the Advanced Expression editor.

To create an advanced expression:

Click Use Advanced Expression in the condition builder.

The Advanced Expression editor has a Design view, which you use to visually create an expression, and a Code view, where you can type the expression. You toggle between the two views using the **Design** and **Code** buttons in the header.

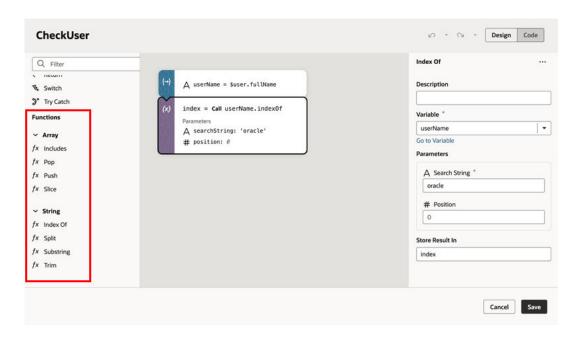
This image below shows Design view, which has an Actions palette on the left, a canvas in the middle, and a Properties pane on the right:



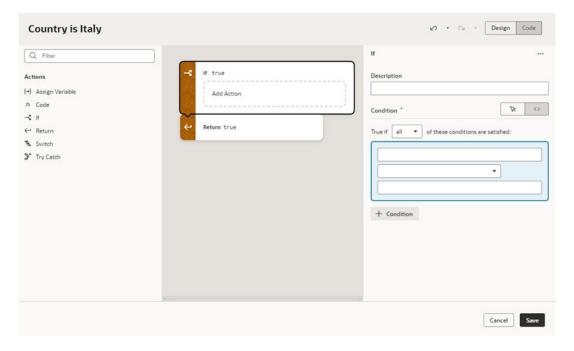
2. Drag an action from the palette and position it on the canvas.

VB Studio includes some basic actions in the palette to help you build expressions. The palette might also contain additional custom functions that have been added to the extension. For example, if Oracle developers have added any global functions to the extension, like Add Numbers or Multiply Numbers, they are listed under Functions in the palette Like the basic actions, you can drag the functions from the palette into the expression, and configure them in the Properties pane. (In Advanced mode, you can add your own global functions. See Add JavaScript Modules As Global Functions.)





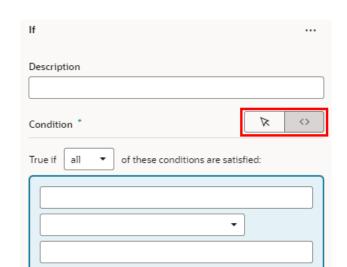
Let's take a look at what happens when we drag an If action onto the canvas:



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.

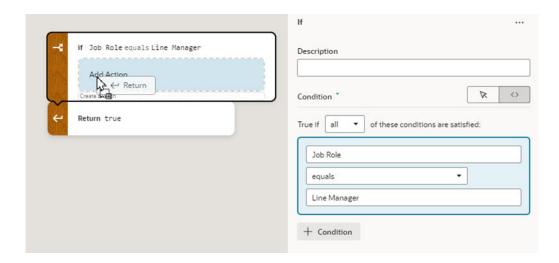


+ Condition

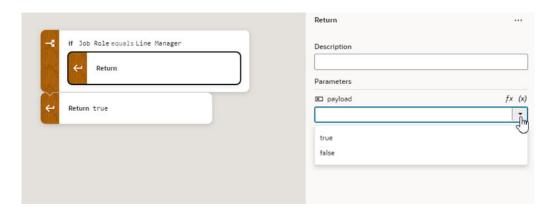
You can use the condition builder's toggle buttons to switch between the Design and Code views:

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:

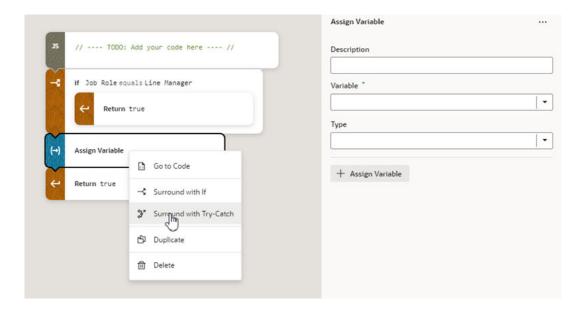


4. Repeat steps 2 and 3 to add more actions to the expression.

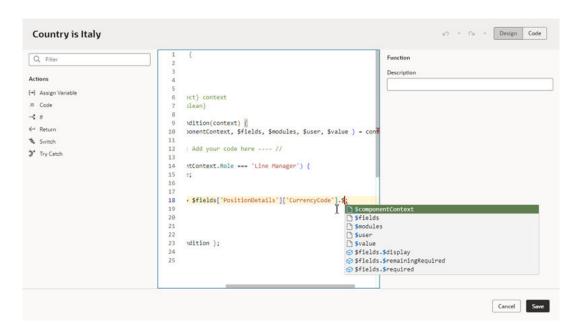
As you add more actions, you can reorganize the order by grabbing the front of the action element on the canvas, and then moving it into a new position:



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 view. In Code view, you can type your expression directly and take advantage of 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, including the built-in rules provided by Oracle (as long as they are not locked).

These properties include:

- Required Make required or optional
- Hidden Visible or hidden
- Read Only Editable or read only
- Value Static or expression

In this context, a region is simply a dynamic form.

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.

Remember that the Default rule is always active, which establishes the out-of-the-box behavior. All other built-in and extension rules 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.

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.

You can use the Filter field to help locate fields in the editor.



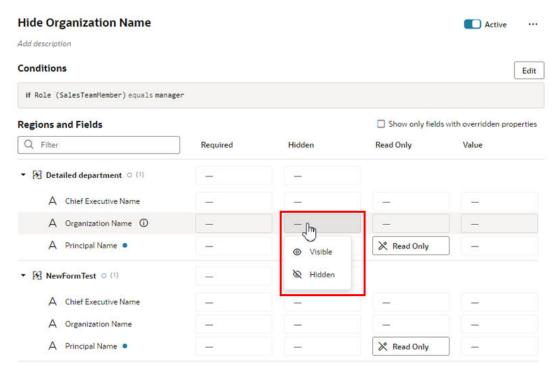
If you have trouble locating the field you want to modify, or the editor is loading slowly, confirm that there were no errors loading services in the editor. The editor only lists fields from services that have loaded successfully. You'll see "Loading Completed with Errors" in the editor's status bar if any errors occurred when loading the editor.

If you see the error message in the status bar and you want to try to locate the cause, see Resolve the "Loading Completed with Errors" Message in the Troubleshooting chapter.

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:



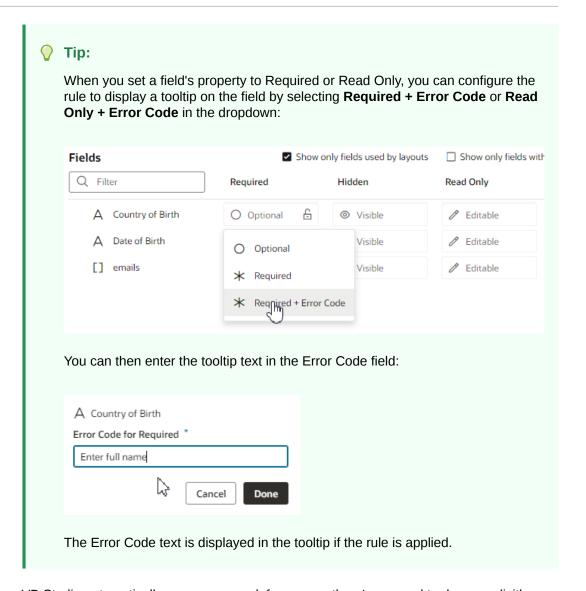


If you change your mind after setting a property, use the **Remove Override** option to restore the property to its original value.

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.

If you see child fields when you expand a region or an object field, you can use the Set All Children popup menu to quickly override the properties of all the child fields. For example, you can use the popup menu to set the Hidden property for all the child fields in a region to Hidden instead of setting the property for each field individually. To override a property, click ... next to the region to open the menu, and then select the property (Required, Hidden, or Read Only) and the override you want to apply:

Regions and Fields Q Filter Hidden Required Summary of Changes Optional Basic Details (i) O Visible ★ Required ActionReason Set All Children... ActionReason Required ActiveStatus Visible ★ Required a Hidden AnnualWorkin Read Only BenchmarkJobFlag



VB Studio automatically saves your work for you, so there's no need to do so explicitly.



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 Studio 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.

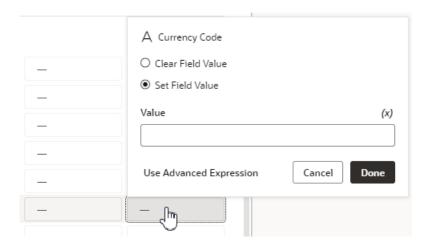
Suppose you have a form in which the user can enter a currency in the Currency Code field. If the user doesn't enter a value—that is, if the field is empty at runtime—you can populate the

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.

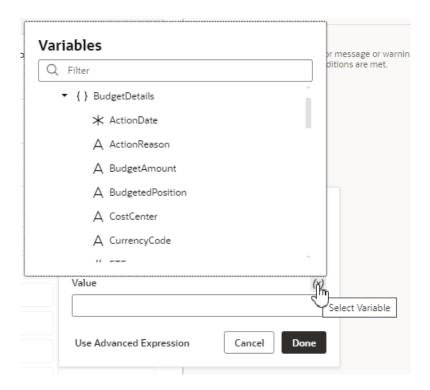
- 3. In the popup dialog, do one of the following:
 - Select **Clear Field Value** to remove any default value already set for the field by other rules. For example, you can use this option to remove the value set by the Default built-in rule. When the rule is applied, this option makes the field's value 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

list:

An expression which typically uses one or more different variables (like <code>\$fields.BudgetDetails.CurrencyCode.\$value()</code>) 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





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.



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:



 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.
- 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 Studio 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.

Set Properties at the Region Level

When setting properties at the region level, there are a few things to note:

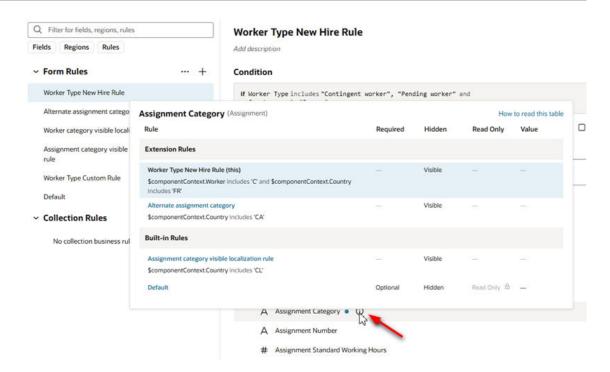
- When you set the Required property at the region level, the region will be required for the
 page, but the fields within the region can be set individually (that is, the setting is not
 inherited). So, even if you hide all the fields within a region, the region heading will still
 appear on the page, albeit with no fields shown beneath it.
- If a field is marked Required by Oracle and then locked, the field's containing region will be considered mandatory at runtime, even though the region itself is not officially marked required. This is because a field cannot be shown on a page by itself, without its parent region.
- When you set the Hidden property at the region level, all of the fields within that region inherit that setting. So even if you make an explicit change for a field within the region, the region-level setting takes precedence.
- You cannot set the Read Only or Value properties at the region level.

Understand What Will Be Shown at Runtime

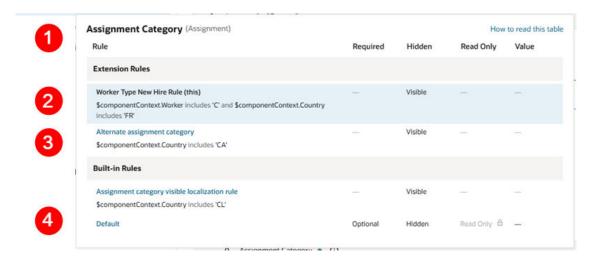
At runtime, VB Studio evaluates all the rules from bottom to top, starting with the Default rule, then moving through the built-in rules, and ending with the extension rules (if any).

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:



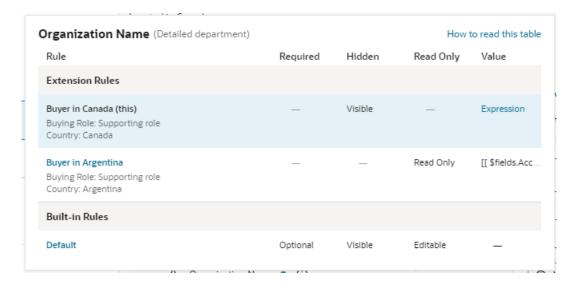
Let's take a closer look at the pop-up viewer:

- Name of the field (Assignment Category) and the region (Assignment) that these rules pertain to.
- Currently viewed rule, which is shown with a blue background. The conditions for each rule appears beneath the rule's name.
- 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 or region).
- 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** field in the **Detailed department** region:



Let's suppose the current user satisfies the conditions for the **Buyer in Canada** rule, that is, they have the Buying Role "Supporting role" 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 a built-in rule that 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 buyers with a "supporting 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 with the "supporting role" 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

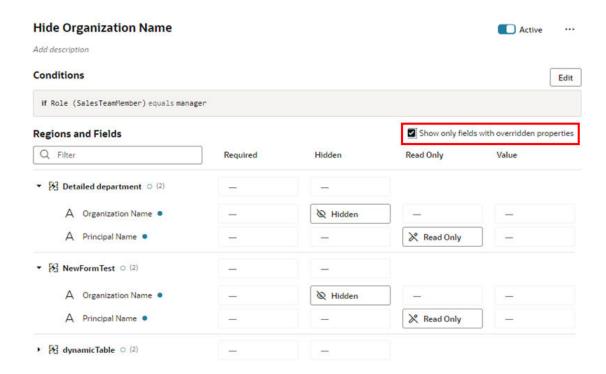
Use **Show only fields with overridden properties** to understand which regions and fields are impacted by a rule, and how are they impacted.

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.



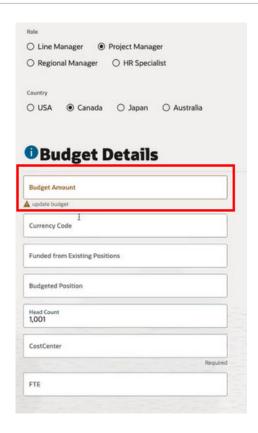


Display Messages in a Form

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, rather than 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:



Note:

Validation rules only display messages in forms rendered using the oj-dyn-form form component. If your application has any forms rendered using the older version of the form component (oj-dynamic-form), validation rules won't work correctly in those forms.

To check which version of the form component is used in a page, :

- 1. Select the form component to open the component's details pane in the Properties pane.
- 2. Hover your cusor over the component's name to view the Info icon (), then move your cursor over the icon to open the component's Info popup.



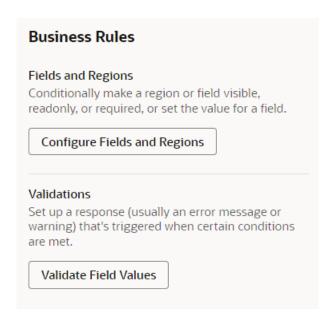
In the example above, the popup shows this form is rendered using the oj-dyn-form form component.

Create a Rule to Validate a Field

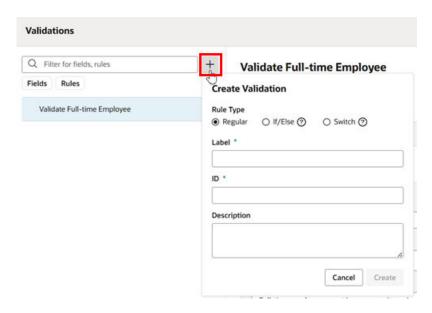
Let's take a look at how to create a rule that works like the example above, displaying a message in a form when the value for the Head Count field is over 1000.

To create the validation rule:

 Open the page containing the form, and then click Validate Field Values under Business Rules in the Properties pane.



- 2. Create a new rule.
 - If some rules are already defined, click \pm to open the Create Validation dialog box.

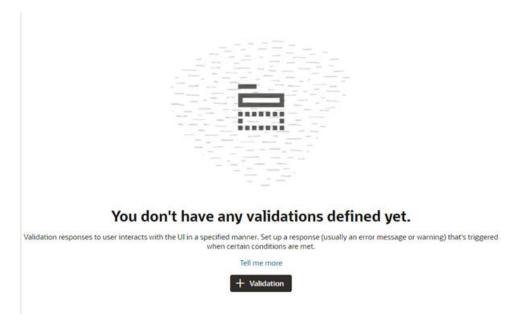


Rather than creating a rule from scratch, you can duplicate an existing rule and use it as the basis for a new rule. To duplicate a rule, right-click the rule in the list, and then click **Duplicate**.



When duplicating rules defined in extension dependencies (the rules with the Built-in label):

- You cannot duplicate a rule if it uses advanced expressions, for example, in the rule's condition or to override a field's value.
- You cannot duplicate a rule if it uses any global functions that cannot be referenced in the extension. If the global functions are referenceable, when the rule is duplicated they are automatically imported, and the references to the function are updated in the rule.
- When duplicating nested rules, you can only duplicate the top-level rule.
 When you duplicate the rule, the child rules are also duplicated.
- If there are no rules yet, click + Validation.



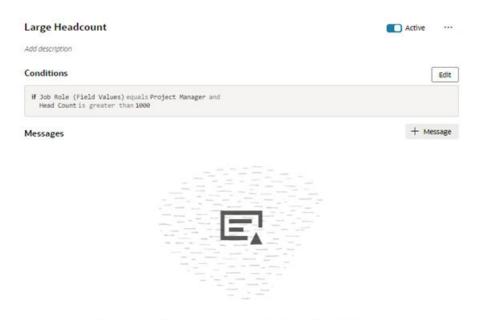
Select the Regular Rule Type in the dialog box, and then enter a label, id, and description. Click Create.

In the Create Validation dialog box, you can select a validation rule using the Template dropdown list (if any are defined in the app), or choose a rule type (Regular, If/else, or Switch). For more, see Add an If/Else Rule and Add a Switch Rule.

4. 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:



There aren't any messages defined for this rule.



5. 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.

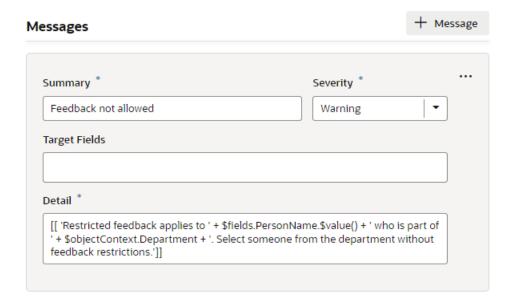
- **Summary**. The Summary text is displayed in the title of a message dialog. The text in the Summary field is not displayed on the page if you are using an inline warning, but it's still a required field.
- Detail. The Detail text is the actual text message. This text is displayed inline in the form 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:

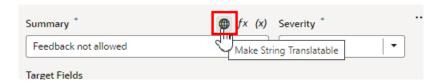
```
[[ 'Restricted feedback applies to ' + $fields.PersonName.$value() + ' who is part of ' + $objectContext.Department + '. Select someone from the department without feedback restrictions.']]
```

To create a message that includes different types, such as variables or objects, you need to surround the message with <code>[[]]</code>, so that it's treated as an Object and evaluated, and not treated as plain text. To construct the message, any static text has

to be enclosed in quotes, and you need to use a + operator to concatenate the static text, variables, and other entities:



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:



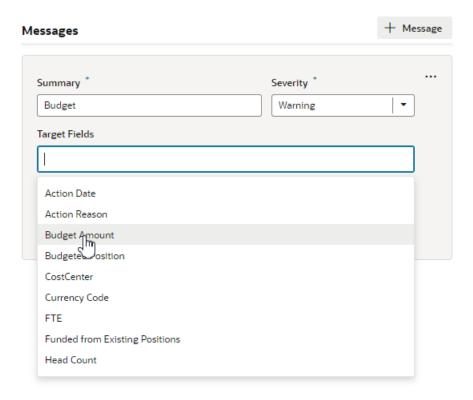
It's not possible to add parameters in the Translatable String popup, but you can click

fx to open the Expression Editor, and then manually edit the translation object if you want to use parameters. For more, see Associate a Translation Key with a UI Component.

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:

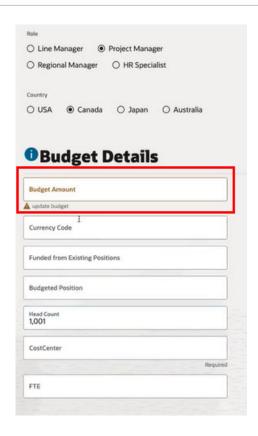
- You do not select a target field, and the form has multiple fields.
- You select more than one target field.

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 view and test the form by entering values to trigger the rule.





Use Templates with Validation Rules

You can use validation rules to display messages only in dynamic forms rendered using the oj-dyn-form component. However, if the form uses field templates, you'll need to manually configure the mechanism for displaying messages.

When using a field template in oj-dyn-form, you'll need to create a mechanism to listen for validation messages in the form component and display the message in the form. In field templates, messages that can be displayed are exposed via the \$messages context.



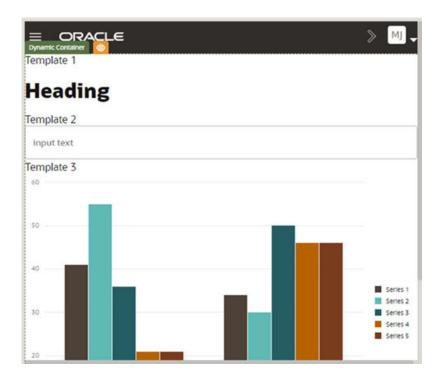
When configuring a validation rule in the editor, you can only select one target field for the message. If you want to display a message on more than one field, you'll need to manually create a mechanism for that.

11

Customize Dynamic Containers

If an App UI developer inserted a dynamic container on a page and made it extendable, you can use it to a) re-arrange the content already there, or b) add your own components to display in the container.

A dynamic container displays content in individual *sections*, or logical regions of the page. You define the UI elements or components displayed in each section, and then set the display logic conditions to determine which sections are displayed. Like the rules in a rule set, the conditions are evaluated in order at runtime. In this example, the dynamic container has three sections, each displaying different content:



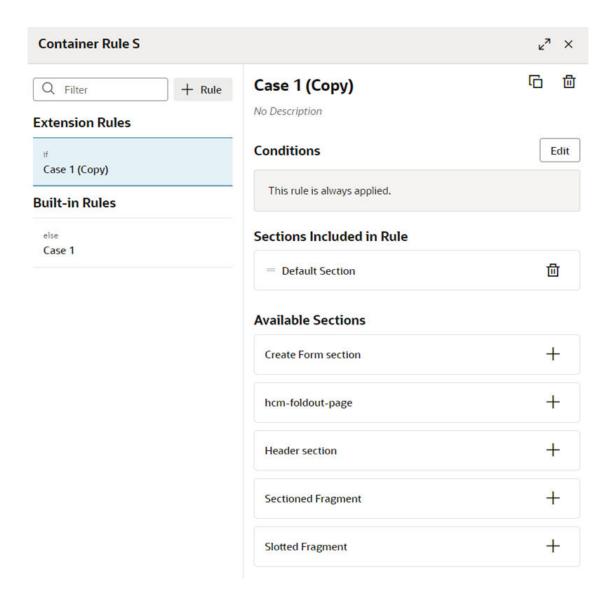
The App UI developer usually defines one or more built-in sections for the container, but you can create your own sections if you want to add components from the Components palette to the page.



Unlike dynamic tables and forms, a dynamic container in a page is scoped to the page on which it appears, which means it can't be used again in other pages. However, if a container is in a fragment, the fragment can be re-used. For more, see Work With Fragments.

How Do Cases Work?

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 for dynamic forms and tables, but instead of selecting which fields to display, you select a section (a template of components) 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. The template defines the content you want to display. Each template in a case is rendered as a section in the container, so if a case defined three templates, you'd see three sections in the container. In the following image, there are several available sections, but only one section is displayed in the container when Case 1 (Copy) is applied:



Configure a Dynamic Container

You can change the sections displayed in a container directly in the Properties pane, or use the container's rules editor if you want to use conditions to define which sections are displayed.

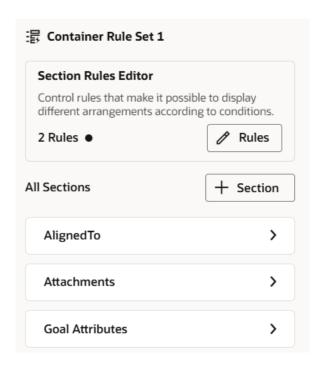
The easiest way to learn how to configure a dynamic container is to examine a real use case. Suppose the App UI developer of one of your dependencies has added a dynamic container and already defined a section. At runtime, the container will display the sections the developer has configured in the default built-in rule. Now let's say that you want to also display a header that is always displayed in the container, so no conditions are needed.

If you don't need to define any rules or conditions in your container, you can modify the container's built-in rule directly in the Properties pane, to add, remove, and change the order of sections. This means that the sections listed under Displayed Sections in the Properties pane are always displayed. You can modify the container in the Properties pane if you see sections with handles listed under Displayed Sections in the Properties pane:



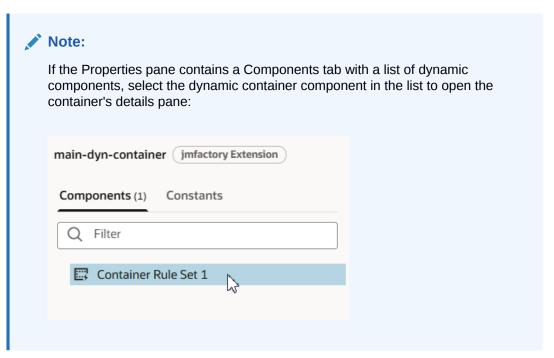
If you want the header displayed only when certain conditions are met, for example, when the current user is a manager, you'll need to use the rules editor to create a rule and define a condition.

Some containers might already have rules and conditions determining what is displayed. If you see All Sections instead of Displayed Sections in the Properties pane, this means the container already uses some rules in addition to the built-in rule, and you need to define the container's sections in the rules editor. The container in this image already uses two rules:

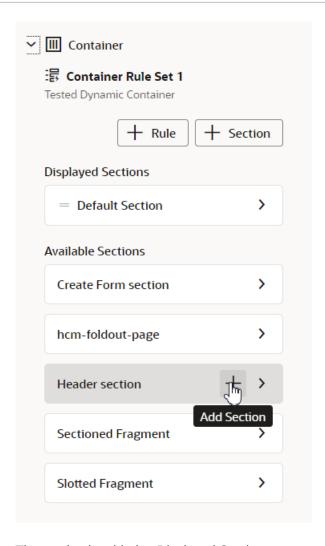


To customize the sections displayed in a container:

Locate the container in the Properties pane.
 The Properties pane lists all the sections that you can display in the container under Available Sections. These sections are usually defined in dependencies, but also includes sections defined in your extension.

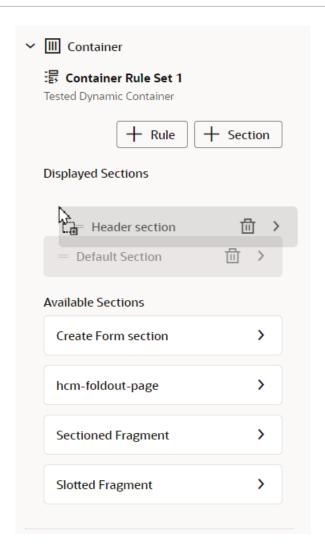


- Customize the container's sections without creating rules and conditions.If the container already uses rules, skip to step 3 below to define the sections using the rules editor.
 - a. Add a section by hovering over the section in the Available Sections list and clicking
 +.



The section is added to Displayed Sections.

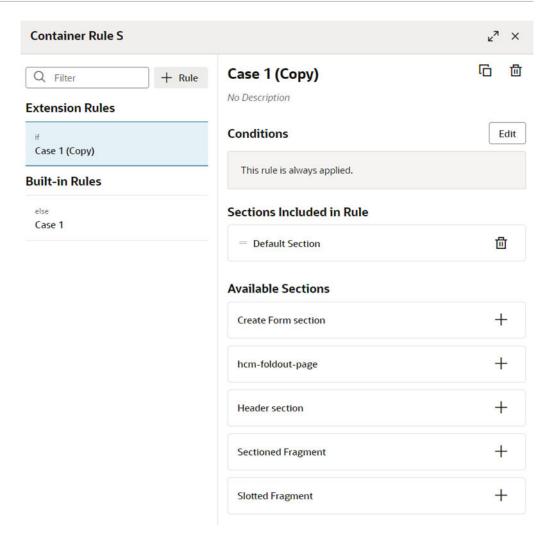
b. Re-order the sections under Displayed Sections by grabbing a section's handle and moving it into the position you want.



- c. Remove a section from Displayed Sections by hovering over the section and clicking $\stackrel{\hbox{$\mbox{\tiny d}}}{}$
- 3. Define a container's sections using rules and conditions:
 - a. In the Properties pane, click + Rule to create a rule, or Rules to open the editor.

The rules editor displays a list of rules for determining the sections displayed in the container. Rules listed under Built-in Rules are defined in the extension dependency, and you cannot edit them.

When you create a rule, it is added to the list under Extension Rules:

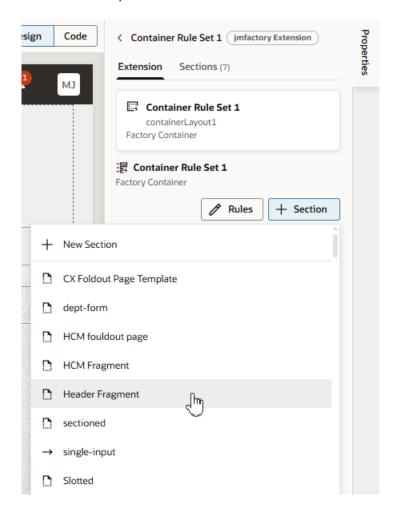


To duplicate or delete a rule, right-click the rule in the list and select **Duplicate** or **Delete** in the popup menu.

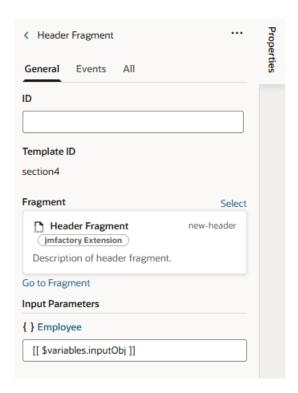
When a rule is selected, you can also use the Duplicate and Delete buttons at the top of the editor.

- **b.** Change the order your extension rules are listed by selecting the rule, and then dragging it into the position you want.
 - Rules are evaluated in order, from the top down. If a rule's conditions are met, the rule is applied, and none of the rules below it are evaluated. When you create a rule, it is added to the top of the Extension Rules list, so you need to make sure your rules are in the correct position.
- c. Define the rule's conditions in the condition builder by selecting the rule, and then clicking **Edit**.
 - The condition builder is the same one used to create conditions in business rules. If you're not familiar with using the condition builder, see Set Conditions for a Rule.
- Choose the sections you want displayed in the container when the rule's conditions are met.
 - To include a section in the list of displayed sections, click the section's + button in the Available Sections list.

- To remove a section, click $^{\scriptsize \square}$. To change the order the sections appear in the container, grab the section's handle next to the section name and move it into position.
- e. Click **X** in the top corner to close the editor.
- Create new sections in the container.
 You can create empty sections, or create sections from one of the templates available in the dependency.
 - To create a section based on a template:
 - **a.** In the Properties pane, click **+ Section**, and then select the template you want to use from the dropdown list.



When you select the template, a new section is added to the container, and the details pane for the template opens in the Properties pane. Depending on the template, the new section might have editable properties, components, or input parameters that you may need to configure:



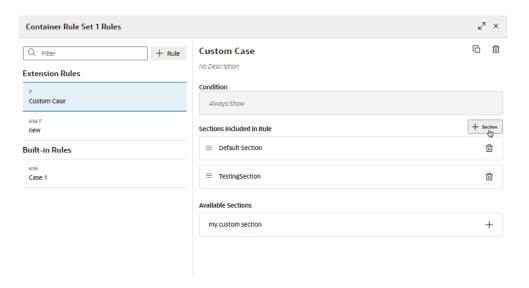
b. Rename the section.

The new section is given the name of the template by default. To rename the section, click ... in the Properties pane, and then select **Rename Section** in the options menu, or hover over the section name and click ?:



- To create a new empty section:
 - a. In the Properties pane, click + Section, and then select + New Section in the dropdown list.

You can also create a new section in the rules editor by clicking + **Section**:



The new section is added to the container, and the section's details pane opens in the Properties pane.

- b. Rename the section. Click ... in the Properties pane, and then select Rename Section in the options menu, or hover over the section name and click ... You can then edit the name in the Properties pane.
- c. Add your content to the new section by dragging components from the Components palette into the section on the canvas.

Guidelines for Working with Container Sections

Here are some things to keep in mind while working with container sections:

- When configuring a container, any section you create extends the full width of the
 container, although an App UI developer has more freedom in this respect. 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.
- 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, dynamic forms, or a button that starts an action chain in your extension.
- 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.
- Unlike dynamic tables and forms, a dynamic container is not bound to a specific data resource, so a section can display data from any service connection available in the extension (and its dependencies).

Add Content From a Custom Object to Your Page

You can add components to a dynamic container section to show data from any data resource available to your extension. The data resource can be one defined in a dependency, or in a

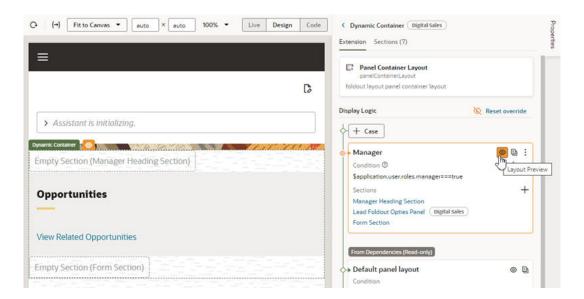
service connection you've added to your extension. For more on using service connections, see Work With Services.

The component can be one of the standard components available in the Components palette, but it could also be a component provided in your extension's dependencies. For example, if you've created a custom child object in your Oracle Cloud Application, you can display the object's data by adding a component, such as a form or table component, to a container section, and then binding the component to the data source.

To add a component that displays data from an Oracle Cloud Application endpoint:

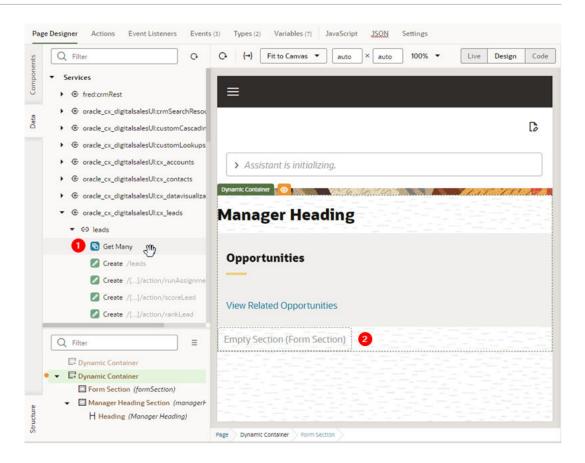
- In the Page Designer, select the dynamic container where you want to display the data.
- Click Layout Preview in the case where you want to add the component.

Activating a case's Layout Preview forces the Page Designer to display that case on the canvas, so you can see the components as they will look in the page. When a case is visible in the Page Designer, you can drag components directly into its sections from the Components palette:



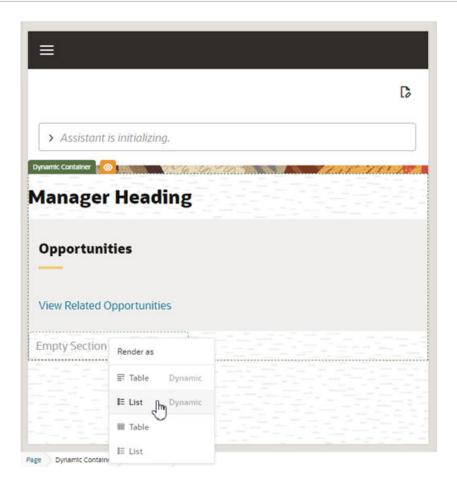
In the image above, the case already has a section named Form Section that we can use for the new component.

- 3. In the Page Designer, open the Data palette and locate the endpoint for the data you want to display in the component.
 - The Data palette lists the accessible endpoints from your extension's dependencies and service connections.
- 4. Drag the endpoint from the Data palette and drop it onto the section on the canvas.
 - You can also drag components and endpoints into the Structure view.
 - In the image below, you can see an object's endpoints (1) in the Data palette, and an empty section (2) in the container.



After dropping the endpoint on the section, select the component you want to add from the menu.

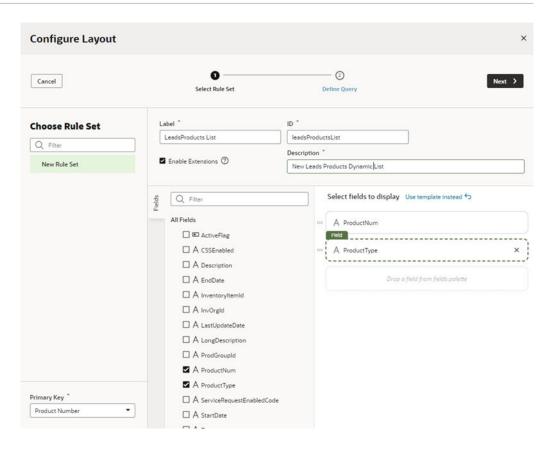
The component options in the menu will depend on the type of endpoint that you dropped in the section. In this example using a Get Many endpoint, you could use either a list or table to render the data. You can choose if the component is dynamic or not, depending on your needs:



In this example, we'll add a dynamic form component. The Configure Layout wizard opens automatically when you select a dynamic component.

- 6. In the Configure Layout wizard:
 - Click Select fields to display, then select the fields you want to display in the component.

If you're binding a dynamic component to an endpoint which doesn't have a Layout, you'll need to enter the details to create a new rule set for the component. The wizard will create a Layout for the endpoint. Select **Enable Extensions** in the wizard if you want to make the new rule set extendable:



If a Layout for the endpoint already exists, you can create a new rule set, but there might already be some rule sets you could use instead of creating a new one. The Layout might also have templates that you can select in the wizard instead of selecting the fields.

- b. Click Next.
- c. Define the parameters for querying the endpoint. Click **Finish**.

After you finish the wizard, you'll see the component rendered in the container in the Page Designer.

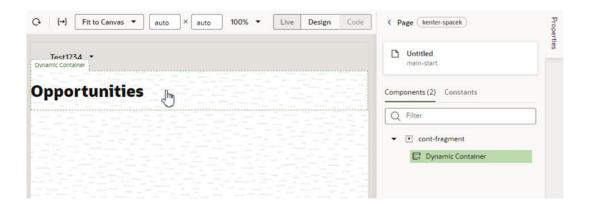
Extend a Container in a Fragment

When a page in a dependency contains a fragment with a dynamic container, you need to open the fragment in the Fragment Designer and customize the container. You won't be able to edit the container's cases and sections defined in the dependency, but you can extend the container to add new cases and sections to add components and functionality to the container, just as you would when extending a container in a page.

To extend a container in a fragment:

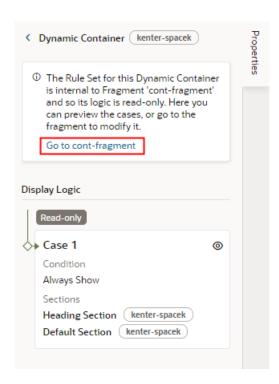
 Open the page containing the container you want to customize, then select the dynamic container.

If the fragment developer has made the container extendable, it will be listed in the Structure view and in the Components tab in the Properties pane.



2. In the Properties pane, click the link to open the fragment in the Fragment Designer.

The Properties pane contains a pane with information about the container, and a link to open the fragment in the Fragment Designer. In this example, the fragment is named <code>cont-fragment</code>:

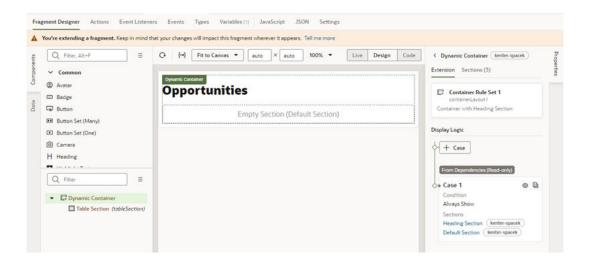


The container's Properties pane also displays the container's Display Logic, but it's readonly in the Page Designer so you can't customize it. You *can* use the Layout Preview in the Display Logic to preview cases, so you don't need to open the fragment in the Fragment Designer just to select a case to see how it looks in the page.

3. Select the dynamic container in the Fragment Designer.

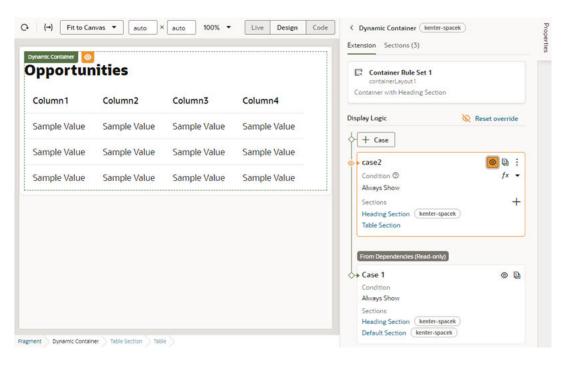
You can select the container on the canvas, in the Structure view, or in the Components tab in the Properties pane.

The Fragment Designer, and all the fragment editor tabs, displays a warning at the top to remind you that you're extending a fragment, and that any changes you make to the fragment will impact the fragment wherever it appears:



4. Customize the container in the Fragment Designer.

You can customize the dynamic container in the fragment like you would customize it in a page. In this example, a new case and section has been added to the container, and Layout Preview has been enabled for the new case:



You can add other components and elements as needed. For example, you might want to create new variables and add events and action chains to add functionality.

Customize Variables and Constants

Variables and constants store static values and expressions that are used in components and pages. For example, they might be used to store data that will be used in an action chain, to store input parameters for a form, and to store strings and values that will be used to define how pages and components in your app are rendered and behave.

To give a simple example, a dependency might use a constant to store the color or the text of a heading in a page. If the dependency developer has made the constant editable by extensions, you can configure the constant to change its value (and thus the heading's color or text). In addition to configuring variables and constants defined in dependencies, you can, of course, create additional variables and constants you may need to configure the pages and Layouts.

Both variables and constants are used to store values. Specifically:

- Variables are typically used to store values used within a page's UI or an action chain. For
 example, you would use a variable for an input text or a total when performing an action in
 the page like adding items to an order, but variables can also be used for property values
 like a text color or font size.
- Constants are typically defined in a dependency, and their values often don't change after they are set by the extension. For example, a dependency might define a constant like contactPageHeading that determines the text in a heading in a page. Your extension can write a value to the constant, and that value will be used every time contactPageHeading is used in your app. The default value of a constant can be static, but it could also refer to expressions that contain variables. If the expression contains variables, the constant's value would change when the variables change, and you can configure the constant to emit an onValueChanged event.

Variables and constants have a *scope* that defines its lifecycle. The scope is defined by where the variable or constant is defined, so if you create a variable in a page, it would have a *page* scope. When configuring dependencies, most variables and constants will have one of the following scopes.

Scope	Description
Page	Variables and constants with a page scope can be used by components and actions within the page, including dynamic containers, but are not accessible from within dynamic forms and tables. These are typically defined in the dependency and are listed in a page's Variables editor if they are editable or readable. For example, the dependency might define an editable constant that you can use to toggle the display of a component. You create and edit page-scoped variables in the page's Variables tab.

Scope	Description
Layout (dynamic forms and tables)	Variables and constants with a Layout scope can be used by components and action chains within dynamic forms and tables, for example, within field templates. You create and edit them in the Layout's Variables tab. For example, you might use a variable in an expression in a field template.
	Dynamic containers use page-scoped variables and constants.
Action Chain	Variables that are created in an action chain can only be used in the action chain where it is created. You create and edit variables with this scope in the Variables tab of the action chain, not in the Variables tab for Layouts or pages.
Fragment	Variables defined at the fragment level can only be used within that fragment. You create and edit them in the fragment's Variables tab.
	Additionally a fragment, though referenced by an outer page, cannot directly reference variables defined in the page. However, a fragment developer can bind a fragment variable enabled as an input parameter to a page-level constant. For details, see Automatically Create and Wire a Fragment Variable on Its Container and Bind Fragment Input Parameters to Page Constants.

When creating field names for variables, if you want to use any special characters (period, colon, dash and space), you'll need to enclose it in bracket notation. For example, an object variable that contains a field called 'child.field' will need to be referenced as \$variables.objVar['child.field'].

For each variable you must specify a Type property to define the type of data that is stored in the variable. Types are also either page-scoped or Layout-scoped. When defining the type for a variable, you'll usually want to choose one of the standard built-in types used to specify data that are a primitive type, a structure or a simple array or collection. If the built-in types don't meet your needs, you can create custom types and types based on an endpoint in the Types tab of your page or Layout.



If a dependency defines types that can be used in extensions, they will also be displayed in the Types editor tab, and you can select them in the Type dropdown list when you create a variable.

For more about using variables, constants and types in your extension, see Work With Variables, Types, and Constants in Part 4 Build an App UI or Fragment.



Change the Value of an Extendable Constant

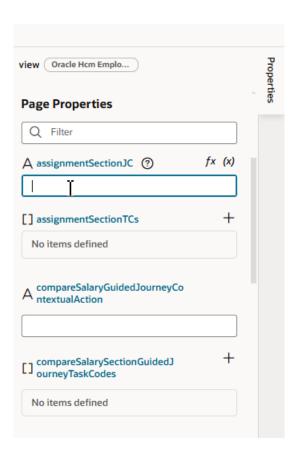
Constants may be used in dependencies to allow you to change the appearance of components and determine app behavior, such as changing component properties or input parameters, or enabling a feature in the page.

To change the value of an extendable constant or property:

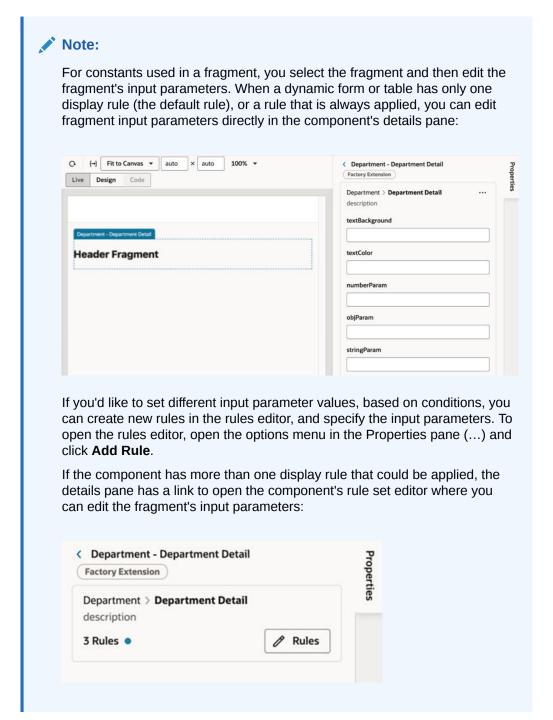
- Open the page in the Page Designer.
- Do either of the following:
 - In the Properties pane, locate the constant you want to modify in the list under Page Properties, and set a value.

If you see a **Configure Page Properties** link in the Properties pane instead of a list of constants, click the link to open an editor where you can use rules to set property values. See Set Property Values Using Rules.

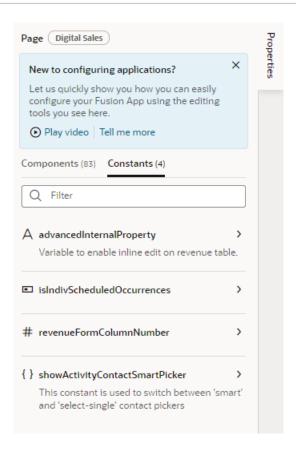
For values that are strings, you typically have the option to enter the value in a text field, or use the Expression Editor or a variables picker to set the value:





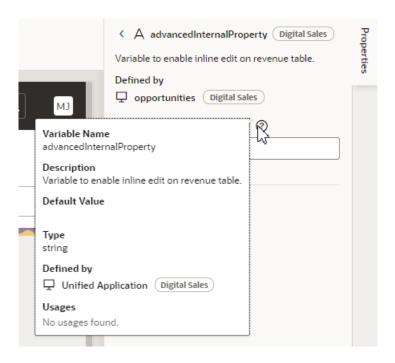


- Open the Constants tab in the Properties pane. In this example, the Constants tab lists
 constants used in the page, but there is no component visible in the Page Designer.
 The constants are used to control elements in the page, for example, to control how
 components are rendered, or to enable and disable some function in the page.
 - a. Click the constant to see its details in the Properties pane.



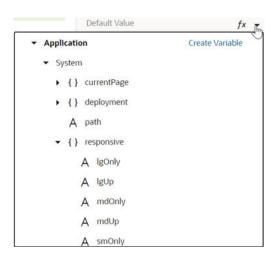
3. Change the constant's value in the Properties pane by selecting a value in the dropdown menu or entering a value in the Values field.

You can move your cursor over the tooltip icon to see details about the constant. In this example, the description states that the constant is used to enable inline editing of a table.



To change the value of the constant, do one of the following:

- Set a static value for the constant by typing a new value in the **Value** field.
- Select a value using the UI component. For some constants, the Properties pane
 might display a component for selecting the value, for example, a date picker
 component for selecting a date. When the constant is a boolean, you might see a
 menu to select "true" or "false".
- Set the value to a constant or variable. You can click the arrow above the field to open the Variables picker and select a variable:

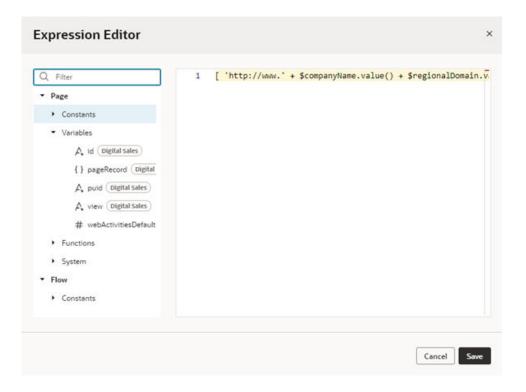


You can click **Create Variable** in the Variables picker to create a variable, if needed.

Set the value to an expression. You can click fx above the field to open the
Expression Editor. 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 To
Write Expressions If a Referenced Field Might Not Be Available Or Its Value Could Be
Null.

In the Expression Editor, you can select variables in the left pane to add them to your expression. You can also add text strings to your expression by typing in the editor.





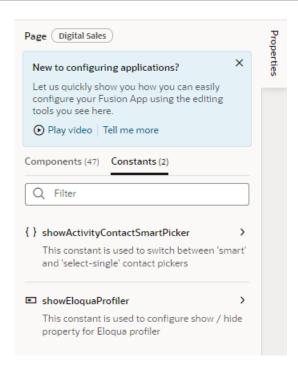
Hide and Show an Element Using a Constant

A dpendency developer may include an element in an app and allow you to hide or show the element using an extendable constant. You can overwrite the default values to change the behavior as needed, to set its value to a static value, another constant, a variable, or an expression.

To overwrite the default value of a constant to hide or show an element:

 Open the page in the Page Designer, and then open the Constants tab in the page's Properties pane.

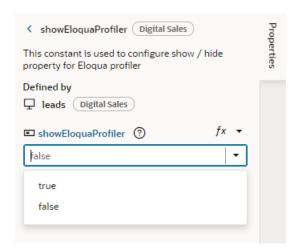
Editable constants used in the page are listed in the Constants tab:



- 2. In the Constants tab, click the constant you want to edit to see its details in the Properties pane.
- 3. Set the value of the constant in the Properties pane.

In this example, the dependency developer provided a description telling you that you can use the constant to show or hide the Eloqua profiler element. The Properties pane also shows where the constant is defined (in the Digital Sales dependency) and a Value field.

The constant showEloquaProfiler is a Boolean constant and takes either "true" or "false" as a value. If you don't want to display the Eloqua profiler in your app, you can hide it by selecting "false" in the menu.



Rather than setting the constant to a static value of "true" or "false", you may instead choose to use another constant or variable in the app to set this value. For example, suppose you want to show this card on mobile devices only. In that case, you can use the presponsive.smonly system variable to set the value of your constant. On mobile devices where the presponsive.smonly value is true, the showEloquaProfiler value is set

to "true" and the Eloqua profiler is displayed. On desktops where the variable is false, it is hidden.

You also have the option to set a constant based on the value of an expression that you create. The expression can include constants and variables as well. Using the previous example, if you instead want to hide the Eloqua profiler on mobile devices, you can create a simple expression using the \$responsive.smOnly variable with the Not operator (!\$responsive.smOnly).

Note:

Make sure the value you enter matches the type of the constant: text for a string constant, "true" or "false" for a Boolean constant, and so on. The Page Designer does not validate the value or expression you enter and you may experience problems during runtime if the type doesn't match.

You can also change the value of page-scoped constants and variables in the page's Variables tab. The Constants tab in the Page Designer lists all editable constants used in the page; the page's Variables tab lists all constants defined in the page, including those that are not currently in use.

Create and Edit Variables and Constants

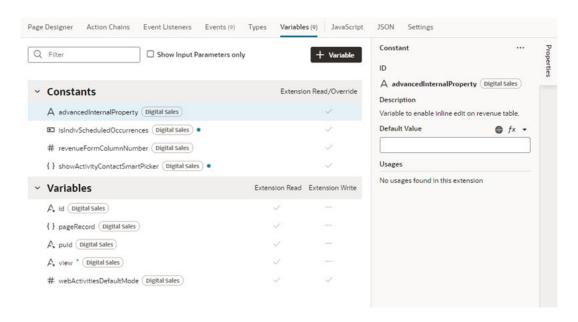
Each page and Layout has a Variables tab with an editor where you can edit variables and constants you've created as well as those defined in dependencies. Variables and constants that are defined in a dependency are labeled with the name of the dependency, and will have fewer editable attributes than those you create yourself.

To edit a variable or constant in the Variables tab:

- 1. Open the page or Layout you want to configure.
 - If you open a page in the Page Designer, the editable constants used in the page are listed in the Constants tab in the page's Properties pane.
- 2. Open the Variables tab of the page or Layout, and then select the constant or variable.
 - The editor lists all the variables and constants that can be used, even the ones that are not currently used.

In this example, a constant defined in a dependency is selected in the Variables tab:





In the image above, the constant listed in the Variables tab is defined in a dependency, and extensions can read and override the constant's value. The check mark next to the constant tells you that the dependency developer has made it readable and writable by extensions.

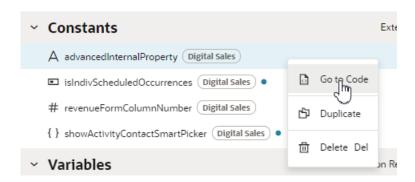
If the dependency developer has marked a variable as one that extensions can use, there will be a check mark in the "Extension Read" column. If the variable can be configured by an extension, there will be a check mark in the "Extension Write" column.

3. Edit the properties of the constants and variables in the Properties pane.

When editing variables and constants defined in a dependency, the dependency developer sets the properties that you can change.

For variables and constants you've created, you can edit the properties in the General tab in the Properties pane, and add event listeners in the Events tab if you want to trigger an action chain. You can also use the Design Time tab to customize how variables and constants are displayed in the Properties pane. See Create Variables.

To edit the JSON file where a variable is defined, right-click the variable name in the tab and choose Go to Code in the menu:



Set Property Values Using Rules

In some pages, you can use rules to set values for extendable page properties. In the rules editor for page properties, you create rules and conditions for setting values, much like you do when using rules to override property values for fields in forms and tables.

For example, your page might use a Guided Journey Page Level Code property, and the dependency developer has made the property extendable. You can create rules to set the property's value based on conditions, such as the user's role. The value is applied only when the rule's conditions are met.

The following steps illustrate how to use the page property editor to set a property's value.

- 1. Open the page you want to modify.
- In the Properties pane, under Page Properties, click Configure Page Properties to open the Page Properties editor.
 - When you open the Page Properties editor, click **Start Preparing** if the editor displays a message "We need to prepare your page properties".
- In the editor, click + Rule, and then enter a name and description in the Create Page Properties Rule dialog box. Click Create.
 Your new rule is added at the top of the list of rules in the editor.
- 4. Move the rule into the position you want it evaluated.

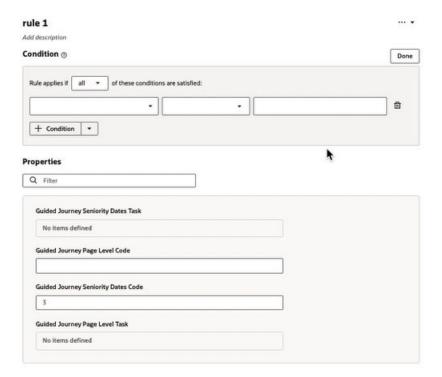
 To change a rule's position, right-click the rule and use the Move up and Move down items in the popup menu.
 - Like business rules, the order in which rules appear in the list is important. Rules in the Page Properties editor are evaluated at runtime from top to bottom. So if a property value is applied by the topmost rule in the list, that value might be replaced by a value applied by a rule below it. However, the rules you create always have precedence over the rules written by the dependency developer, so your rules are evaluated after the built-in rules.
- Select the new rule in the list of rules, then click Edit to open the condition builder.
- Create the rule's condition. You create conditions in the page property editors just like you set rule conditions for fields in forms and tables.



You cannot use the visual expression builder to create conditions for page property rules.

7. In the editor's list of Properties, set values for the page properties that you want to be applied when the condition is met.





Trigger Actions in Dynamic Components

To create a causal relationship in your dynamic components—for example, to add a button that opens a link—you define 1) what you want to happen, and 2) under what circumstances. In other words, you must define three things:

- an *action chain*, composed of a sequence of actions used to define behavior. In the button example, an action chain opens the link.
- events, which define when an action chain starts. In the button example, the button-click is the event.
- event listeners, which start an action chain when the event occurs.

When working with dynamic components, where and how you create the action chains, events, and event listeners depends upon the type of component:

- For dynamic tables and forms, you define the functionality in the Layout.
- For dynamic containers, you define the functionality in the page containing the container.

You can't edit action chains, events, or event listeners defined in a dependency, but you can use them in your extension if the App UI developer has made them extendable. For example, if a dependent App UI adds a "click" component event to a button and makes the event "listenable" for extensions, you could create a listener in your extension that listens for it. This would allow you to use the event to trigger your own action chain, even though you can't configure the component itself.

VB Studio provides tabs with editors for creating the various elements you'll need to add action chains and events to your dynamic component. Here's what each of the editors do:

Editor	Description
Actions	Create and edit action chains describing the behavior of components.
Event Listeners	Create and edit event listeners that start your action chains.
Events	Create and edit custom events used in action chains.
Types	Create and edit variable types.
Variables	Create and edit variables and constants.
JavaScript	A JavaScript editor.
JSON	Edit the JSON metadata files.

An editor only has access to elements in its own scope. So if you're configuring a page, you'd use the page's Variables editor to edit extendable variables used in the page, but if you're configuring a dynamic form you'd need to use the Layout's Variables editor to edit the extendable variables used in the form.

In the Action Chains editor, each action chain also contains a Variables tab for adding and editing the action chain's variables. For details on the Action Chains editor and how to create action chains, see Create an Action Chain and Test Action Chains.

Trigger Actions in Layouts

To trigger actions in dynamic tables or forms, you use the Layout's editors to define action chains, events and event listeners:



When adding functionality to a Layout:

- The action chains, events, and event listeners defined in a Layout can only be used in the Layout, but they can be used by any of the dynamic components belonging to the Layout. So an event listener in a Layout can only listen for events defined in the Layout, and can only trigger action chains defined in the Layout. The exception to this is that an event listener in a Layout can listen for the vbEnter lifecycle event.
- Custom events defined in a Layout can be made listenable by the page containing the
 component by choosing Emit event to page when you create the event. For details on
 how this works, see Raise Fragment or Layout Events that Emit to the Parent Container.
- You can create event listeners for events in the Layout that you define, and for events in the Layout that have been designated as "listenable" in a dependency.
- Your extension can trigger events defined in dependencies if they are designated as "triggerable".

Define Behavior for Components in Layouts

You can configure components in field and form templates to start action chains. For example, if you've created a field template that contains an icon component, you could add a component event to it to trigger some behavior, say, open a URL.

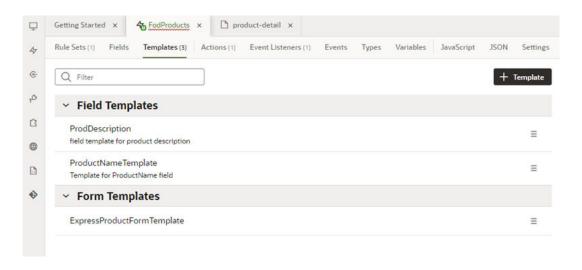


You can only create a **component event** from a component's Properties pane. You can't create one in the Layout's Events editor.

To define the behavior of a component in a field or form template:

 Open the Layout's **Templates** tab, and then select the template that contains the component you want to configure.

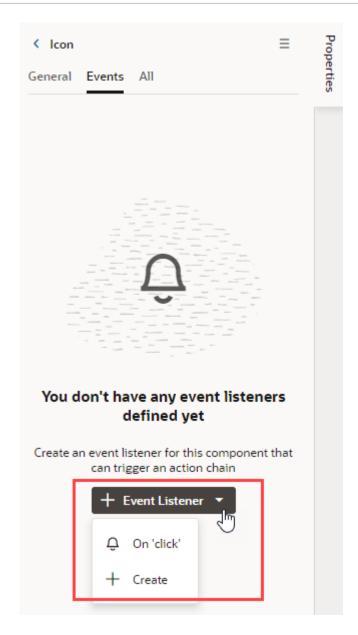




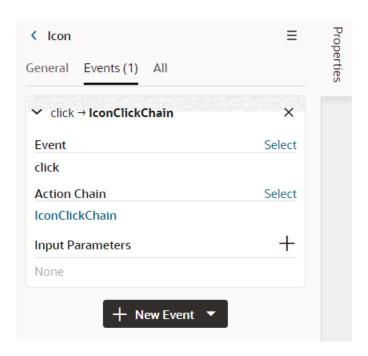
2. Define the component event:

- a. In the template editor, select the component on the canvas or in the Structure view.
- **b.** In the Events tab of the component's Properties pane, click **+ Event Listener** and select the component event option in the list.

The drop-down list shows the component event type that's most typically added to your component (for example, an "onclick" event), but you can also select + Create in the list to select a different type (for example, "onfocus"). If you select + Create, you'll need to select the component event type and specify the action chain it should trigger.

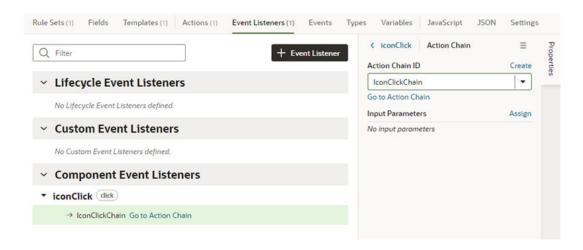


The Events tab now shows the properties of the component event, and you can edit the properties in the tab, for example, to add and assign input parameters that you might want to use in the action chain. Input parameters can provide values from the component and its page to the action chain, which the action chain can then use to determine its behavior. If you create an input parameter in the Events tab, you can edit it in the action chain's Variables editor.



When you add the event, an action chain is created for you, and the Action Chain editor opens automatically. A component event listener for the new event is created for you, and the listener is mapped to the action chain.

In the Event Listeners editor, you can see the new listener listed under Component Event Listeners, as well as the name of the action chain that it will trigger. This image shows the Event Listeners editor with the list of all the event listeners defined in the Layout.



- Define the component's behavior with an action chain.
 - a. Open the Layout's Actions tab.
 - b. Create the variables you might want to use in your action chain.

You can't use a page's variables directly in your action chain, but you can create your own in the action chain's Variables editor.

c. Create the action chain in the editor.

For more on creating the action chain, see Work With JSON Action Chains.



Start an Action Chain From an Action Chain

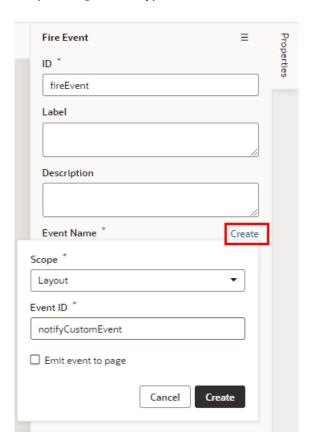
You add the Fire Event action to an action chain when you want to start another action chain from within it. Typically you would do this when you want to trigger a notification, like displaying a popup window with a message, or perhaps when you want to transform some data. For example, there might already be a showMessage action chain that displays a notification. You can add a Fire Event action to your action chain to start the showMessage action chain.

To start an action chain from an action chain:

1. Open the Layout's Actions editor, then select the action chain which you'll edit to add the Fire Event action.

The editor only lists the action chains in the Layout that you can edit.

- 2. In the Diagram editor, drag the Fire Event action from the palette and drop it in the action chain where you want the event to occur.
- Select the Fire Event action in the action chain, then specify the custom event it should trigger in the Properties pane.
 - If the action should trigger an existing custom event, select it in the Event Name dropdown list.
 - If the action should trigger a new custom event, click **Create**, then set the event's Scope to Layout and type an Event ID. Click **Create**.





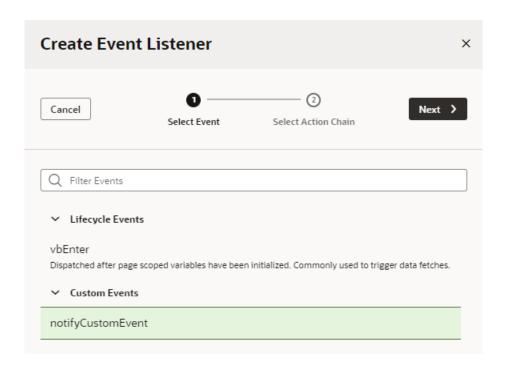


You can choose **Emit event to page** (or **Emit event to container** for fragments) if you want to allow the event to be listenable in the parent container. For details on how this works, see Raise Fragment or Layout Events that Emit to the Parent Container.

4. Click **Go to Custom Event** in the Properties pane to open the Events editor.

You use the Events editor to edit the event's Behavior and Payload properties. For details on these properties, see Choose How Custom Events Call Event Listeners.

- Create an event listener for the custom event.
 - a. Open the Layout's Event Listeners editor, then click **Add Event Listener**.
 - **b.** In the Create Event Listener dialog box, select the custom event for the Fire Event action in your action chain. Click **Next**.



c. Select the action chain you want the event listener to start. Click Finish.

Start an Action Chain from a Field or Variable

You can start an action chain by adding an event to a field in field templates or to variables in the Layout. 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 <code>onValueChanged</code> event that's triggered when the value changes, and the event can start an action chain that retrieves the data and displays it in your page. The event can start action chains you create or action chains already defined in the Layout.

If you are using a variable in your Layout, say in a template or action chain to store an amount, you can add an event so an action chain is triggered when the amount changes. You can add the event to variables you've defined in your Layout, as well as to extendable variables in

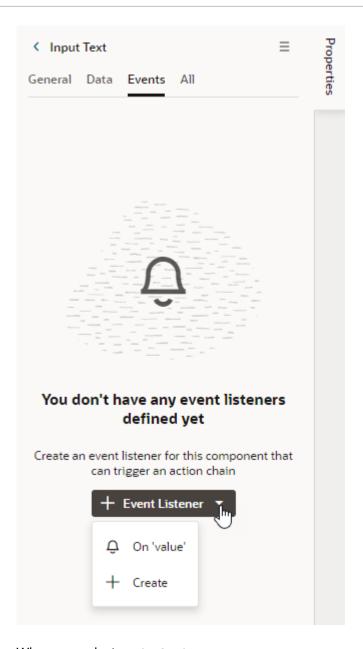
dependencies. You can also add an onValueChanged event to constants if its default value is an expression containing a variable.

To start an action chain when a field's value changes:

- **1.** Do one of the following:
 - Open the Layout's **Templates** tab, and then open the field template you want to edit in the Template editor.
 - Open the Layout's Variables tab.
- 2. Select the text field, then open the **Events** tab in the Properties pane.
- 3. Click + Event Listener in the Events tab and select the On 'value' event in the drop-down list.

The suggested event for a field is <code>On 'value'</code>, which is triggered when the 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. You can also add more events to the field.





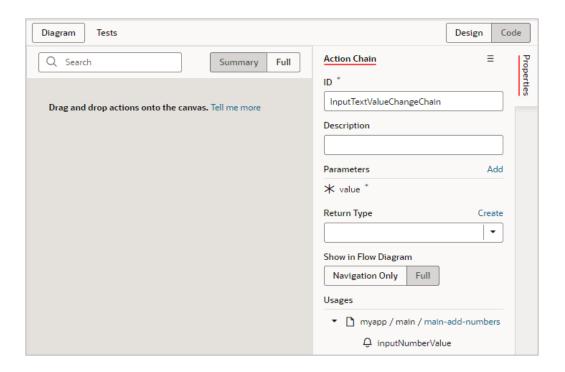
When you select On 'value':

- an onValueChanged event is defined for the field;
- a new action chain is created;
- an event listener for the event is created that will trigger the new action chain when the event occurs; and
- you are navigated to the new action chain in the Action Chain editor.

The variable's Events tab displays the action chain the event listener will trigger. In the Events tab you can change or remove the action chain, add and assign input parameters, and add more action chains.

- 4. Define the action chain.
 - a. Define the action chain's properties in the Properties pane.

In the Properties pane, you can edit the default ID, add a description, and configure the action chain's input parameters and return type.



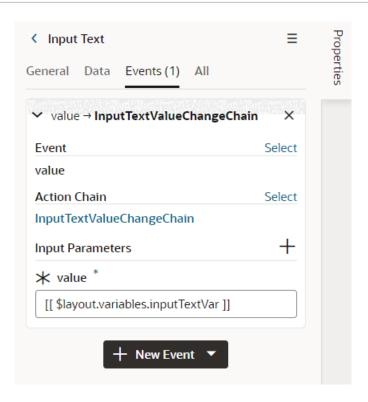
create the variables you might want to use in your action chain.

Depending on the actions you add, you might also need to create variables used in the action chain, for example, variables for the action chain's input parameters. You can't use a page's variables directly in your action chain, but you can create your own in the action chain's Variables editor.

c. Create the action chain in the editor by adding actions from the palette.

For more on creating the action chain, see Work With JavaScript Action Chains.

If you navigate back to the Events tab in the Properties pane for the field or variable, you can see the event details, including the name of the action chain and input parameters. You can add more action chains that will be triggered by the same event, or you can add different events.

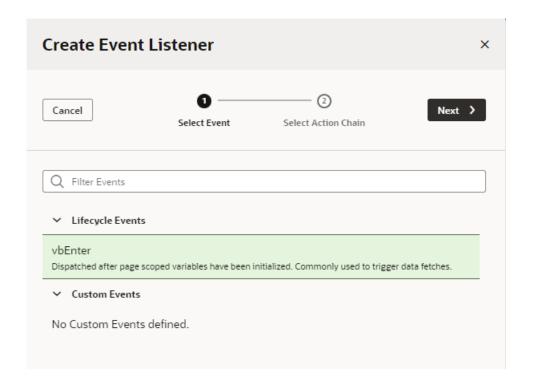


Use Lifecycle Events to Start Action Chains in a Layout

Lifecycle events are pre-defined events that occur during a page's lifecycle. You can start action chains when these events occur by creating event listeners for them. The only lifecyle event that you can listen to in Layouts is the <code>vbEnter</code> lifecycle event, which is triggered after page-scoped variables have been initialized. For example, you can use an event listener for the <code>vbEnter</code> event to trigger an action chain that will assign values to some of your dynamic component's variables when the page opens.

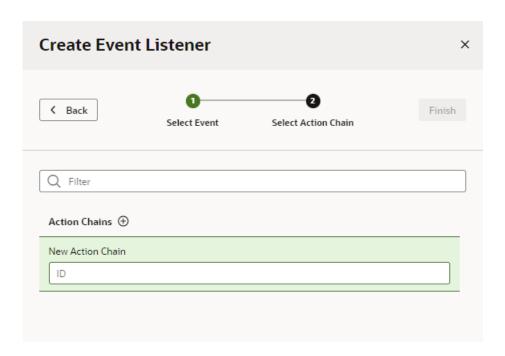
To start an action from the vbEnter lifecycle event:

- 1. Open the Layout's Event Listener tab, then click **Add Event Listener**.
- In the Create Event Listener wizard, expand the Lifecycle Events category and select the vbEnter event. Click Next.



3. Select the action chain you want to trigger. Click Finish.

If the action chain you want to trigger is not listed, you can click **Add** and type an ID for a new action chain, which will then be created for you.



The new listener is grouped by the type of lifecycle event in the Event Listeners editor. The editor also displays the action chains each event listener will trigger. When you select an event listener, you can use the link in the Properties pane to open the action chain in the editor.



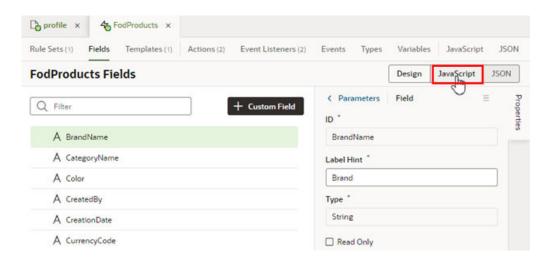
Call Custom JavaScript Functions

You use a JavaScript editor to add custom JavaScript functions that can be called from within pages, components, action chains and fields in your extension. Any JavaScript code that you add will have a defined scope based on the editor where you write the code. If you have some JavaScript code that will only be called from within a specific Layout, you can write your code in that Layout's JavaScript editor.

Each page and Layout has a separate JavaScript file that you open in their respective JavaScript editors. For example, you open the JavaScript file for the FoDProducts Layout by clicking the Layout's **JavaScript** tab, or by locating the Layout's JavaScript file (layout-x.js) in the Source view in the Navigator. After adding the code, you can then call the functions from the Layout's action chains and components.

```
profile x
              FodProducts x
                                            Event Listeners (2)
                                                                                       JavaScript
      define([], function() {
 2
         'use strict';
 4
        var LayoutModule = function LayoutModule() {};
 6
         * @param {String} arg1
 8
 9
         * @return {String}
 10
        LayoutModule.prototype.getSubtotal = function (arg1) {
 11
 12
        };
 13
        return LayoutModule;
14
15
 16
```

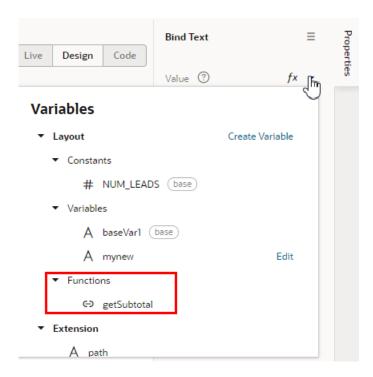
Layouts also have a JavaScript file for custom functions used in properties for the Layout's fields. You can edit this file by opening a Layout's Fields tab and then selecting the JavaScript editor in the header, or by locating the JavaScript file for the Layout's fields (data-description-x.js) in the Source View in the Navigator.



The editor provides code validation and identifies the lines that contain syntax warnings and errors in the right margin of the editor. A lightbulb in the left margin indicates a hint for correcting invalid JavaScript code.

You can use custom JavaScript functions:

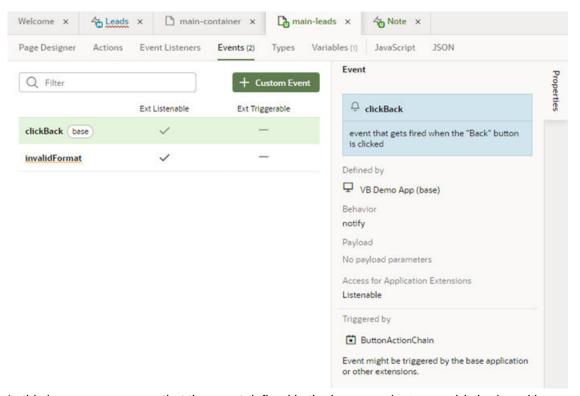
- In an action chain, using the Call Function action. For details, see Add a Call Function Action.
- In a component or field property, by selecting the function in the Expression editor or Variables picker in the Properties pane.



Use Events Defined in a Dependency

A dependency might define events that you can use in your extension to start action chains. In the Events tab of your page or Layout, you can see which events are *Triggerable* and/or

Listenable by your extension. When you select an event in the Events tab, you can see details about the event in the Properties pane, such as the where it is defined, what triggers the event and the event's behavior type.



In this image you can see that the event defined in the base app (clickBack) is badge with base, and that it's designated as listenable by the app extension, meaning you could add an event listener to your app extension that listens for the clickBack event. You can't change the designation of custom events defined in the base app, but you can use them to start action chains defined in the base app or in your app extension.

Event Designation	Description			
Triggerable	Triggerable events are used to start action chains defined in the base app. If an event is designated as Triggerable, you can call the event from an action chain in your app extension using the Fire Event action.			
	For example, the base app might define an action chain that opens a popup window, and define a custom event (for example, openPopup) and event listener to trigger the action chain. If the custom event is Triggerable, you can call the event from your app extension. So if you want to use that action chain when a user selects a specific option, you can add the Fire Event action to the action chain in your app extension that is triggered when the option is selected. In the Actions editor, you would choose the openPopup event when you configure the Fire Event action.			
Listenable	Listenable events are used to start action chains defined in your app extension. If an event in the base app is designated as Listenable, you can add an event listener to your app extension and configure it to trigger your action chain when the listenable event occurs.			
	For example, you can use listenable events to:			
	 execute an app extension action chain asynchronously or synchronously cancel an event in the app extension action chain transform a return value that is passed between the layers of event listeners 			

Trigger Actions in Dynamic Containers

When configuring actions in a container, the events, event listeners and action chains you can use will be defined in the page where the container is located. In addition to those you create, you can also use those defined in the dependency if the dependency developer has made them accessible.

When adding functionality to a dynamic container:

- The action chains in your container defined in the dependency, and any action chains you
 create, will be visible in the page's Actions tab. This means they can also be used in other
 containers in the page.
- You can create event listeners for events that you define in your container, and for events defined in a dependency which the despendency developer has designated as "listenable".

The steps for adding functionality in a container are similar to those for adding it in a page. For details on action chains, see Work With JSON Action Chains.

То	Do	this
Start an action chain from a component		Select the component in the container section. If you're adding functionality to a component and it isn't visible by default in the Page Designer, you can select the dynamic container and then use the case's Layout Preview to display it.
	2.	Add a component event in the Properties pane and define the action chain. See To start an action from a component.
Start an action chain when a variable changes		Select the variable in the page's Variable editor.
		Open the variable's Events tab in its Properties pane.
	3.	Click + Event Listener and select the on 'Value' event, then define the action chain. See Start and action chain when a variable changes.
Start an action chain from an action chain		Select the action chain in the page's Actions tab.
		Add the Fire Event action to the action chain. See Start an action from an action chain.



Work With Fragments From Dependencies

Fragments are reusable pieces of UI that can be used in multiple pages, in other fragments, and also within dynamic components, such as a dynamic form or table. If a fragment developer has made a fragment in a dependency extendable, you can configure it in your extension.

Each fragment is defined once, either in your extension or in a dependency. Each page or component containing the fragment is referencing the same fragment. This allows for consistency across pages and components, so you know the fragment will be rendered and behave the same way in every instance where it appears. For example, if a fragment contains a dynamic container, the dynamic container will be rendered and behave the same way in every page where that fragment appears. The container will include the same templates, cases, sections and UI components in each instance of the fragment. This means that when you customize the container in the fragment, your changes will affect the dynamic container in every place the fragment appears in your extension. For more on what you can do with fragments, see What Are Fragments? and Work With Fragments.

A fragment developer can make the contents of a fragment extendable, which means you can use your extension to configure the fragment, for example, to:

- Change the fragment's input parameters;
- Override a fragment's constants;
- Use events to configure fragment variables; and
- Configure rule sets in dynamic forms in fragments;.

If you've added any dependencies that contain fragments, the fragments are listed in the App UIs pane in the Navigator. (The Navigator's Dependencies pane lists the dependencies added to your extension. For more on how to do this, see Add a Dependency.) When you look at a dependency in the App UIs pane, it will contain a Fragments node that you can expand to see all the fragments it contains:



Note:

The Unified Application, which underlies all App UIs in your extension, also contains fragments, specifically <code>ask-oracle-bo-metadata</code> and <code>ask-oracle-chat</code>. These fragments, while globally available to all App UIs in your extension, are internal to the Unified Application and cannot be customized or referenced in your extension.

What are Referenceable and Extendable Fragments?

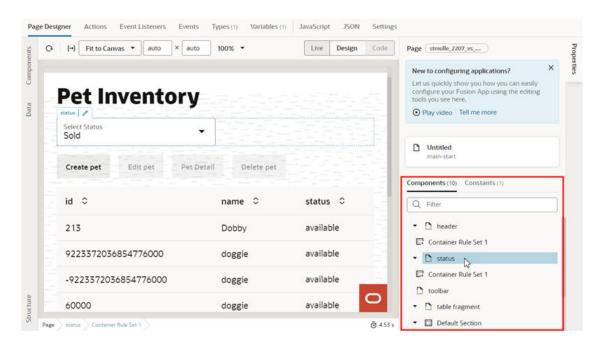
Just because a fragment is listed in the App UIs pane, or is already used in some of your dependency's pages, doesn't necessarily mean you can customize it or add it to pages in your extension:

- To add a fragment from a dependency, it must be *referenceable*, which means the
 developer has made it accessible to extensions. A fragment could be referenceable, and
 not be extendable. For example, you might be able to add a fragment to a page, but there
 is nothing in the fragment that you can customize. You can find referenceable fragments in
 the Page Designer's Components palette.
 When creating a fragment in your extension, if you don't make the fragment referenceable,
 - When creating a fragment in your extension, if you don't make the fragment referenceable, it can only be used in the pages where you've added it—others won't be able to use it in other pages. For more about making your fragment referenceable, see Make a Fragment Available to Other Extensions.
- To customize a fragment from a dependency, the fragment must be extendable, meaning it contains some artifacts—for example, variables or components—which the fragment developer has made extendable. A fragment could be extendable, and still not be referenceable. For example, you might be able to customize some components and variables in a fragment from a dependency, but if the fragment is not referenceable you can't add the fragment to a page. You can find accessible fragments in a page's Properties pane.

When creating a fragment in your extension, if you make artifacts in your fragment extendable, others will be able to customize them.

Looking at the Properties pane in the page below, you can see that the page contains some fragments that their developers have made accessible to extensions:





You can tell you can configure the fragments because they're listed in the Properties pane. This page might also contain other fragments that you can't configure, but they wouldn't be listed in the Properties pane.

When you select a fragment in the Page Designer (you can select it on the canvas, in the Structure view, or in the Properties pane), the Properties pane will display a list of the fragment's extendable items. To open the fragment in the Fragment Designer, on the canvas

you can click the icon () that appears next to the fragment name, or, in the Properties pane, click the Configure button or any of the fragment's extendable items:



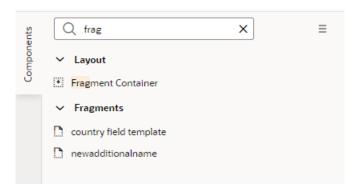
Add a Fragment From a Dependency

When extending a dependency, you can add fragments you've created in your extension, and referenceable fragments defined in your extension's dependencies. For example, let's say that one of your dependencies contains a fragment containing some UI components that have already been formatted, and you would like to include the fragment in a dynamic form. If the fragment is referenceable, and has the correct type, you can add it to a dynamic form template (or field template) in a Layout.

To add a fragment from a dependency to a Layout's template:

- 1. Locate where you want to add the fragment.
 - In this example, we're adding a fragment to a field template in a Layout, but fragments can be added to form templates and dynamic containers in the same way.
- In the Components palette, locate the Fragment Container or the fragment you want to add.

In the Components palette, you can use the palette's filter field to locate fragments. In this example, the palette lists the Fragment Container component and two referenceable fragments that can be added here:



Though there might be many referenceable fragments defined in the dependencies, the Components palette only lists the fragments that can be used in this context. In this example, the context is a field template in a Layout, so only those fragments that the developer has tagged fieldTemplate are listed. For more about how developers tag fragments, see Manage Fragment Settings.

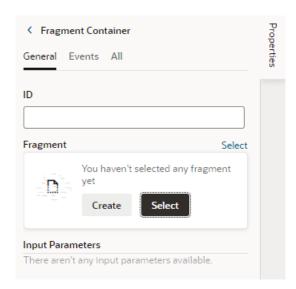
3. Drag the fragment (or Fragment Container) from the palette and position it on the canvas or in the Structure view.

If you drag a fragment onto the template, the Fragment Container component required by the fragment is added automatically.

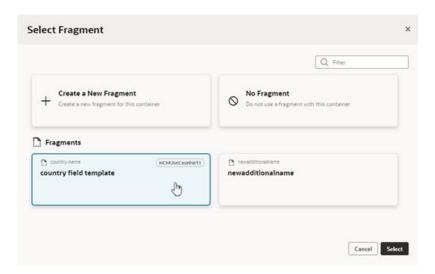
If you drag a Fragment Container instead of the actual fragment, you'll need to select the fragment in the Properties pane:

 After adding the Fragment Container component, click Select in the component's General tab.



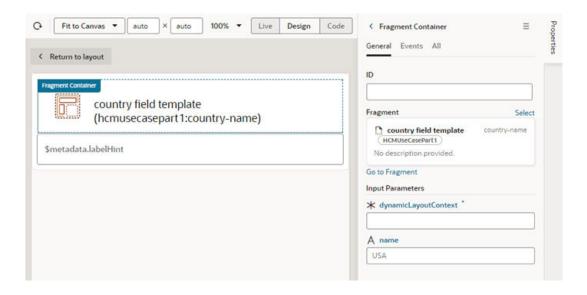


b. Select the fragment in the dialog box. Click **Select**.



Once added, the fragment is rendered with a placeholder on the canvas. When the fragment is selected, you'll see the fragment name and properties in the Properties pane:

- A Select link to select a different fragment.
- A **Go to Fragment** link to open the fragment in the Fragment Designer where you can customize it.
- The fragment's input parameters.



Note:

In this example, we've added a fragment to a field template in a Layout. The fragment's input parameters need to be specified in the Fragment Container so that data can be passed between the fragment and the Layout. In this case, you'll need to provide the context for the Layout and field in the **dynamicLayoutContext** field so that the field in the fragment can be bound to the data. For example, if the template contains a component that renders data from the countryName field, you can use the Variables picker or Expression editor to bind the fragment's Input Parameter variable to the field in the Layout. See Enable Fragment Variables as Input Parameters.

Extend a Fragment in a Dynamic Form or Table

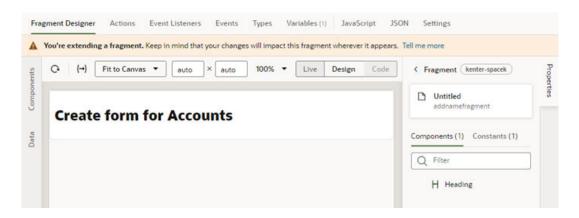
When extending a dynamic form or table that uses a fragment in a form or field template, you can customize the fragment if the developer has made any of its components or variables extendable. For example, a fragment might contain several components that are used in a form template, but only some of the components might be extendable.

To extend a fragment in a dynamic table or form:

1. Open the Navigator's App UIs pane, and then select the fragment in the dependency.

When you select the fragment, the fragment will usually open in the fragment's Fragment Designer tab, but it might open in one of the other editor tabs used for configuring the fragment. For more about working with fragments and the different editor tabs, see Develop an App UI or Fragment.

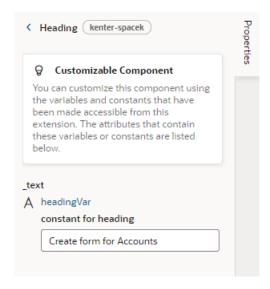
When the fragment is open, you'll see a warning to remind you that you're extending a fragment and that changes to the fragment will impact everywhere the fragment is used:



The Properties pane lists the extendable components and constants in the fragment.

- 2. In the Fragment Designer, select the component on the canvas or in the Components tab in the Properties pane.
- 3. Customize the component's attributes in the Properties pane.

In this example, the fragment developer has made the Heading component extendable, and the component's customizable attributes are listed in the Properties pane when you select it:



For this component, the Heading component uses a constant to set the component's text. You can edit the constant in the Properties pane, or you can open the fragment's Variables editor tab to customize the constant (and any other extendable variables in the fragment).

You can use the other fragment editors to extend the fragment, for example, to configure extendable events or add new events in the Events editor tab.

Extend a Fragment in a Dynamic Container

If your dependency contains a dynamic container that uses fragments, for example to format fields, add UI components in a section, or to add functionality, you can customize the fragment's contents that the developer has made accessible to extensions.

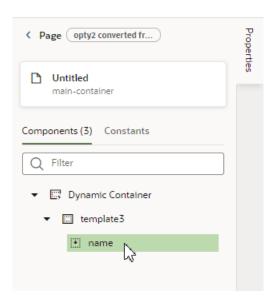


When you extend a fragment in a dependency, the changes you make will affect every place that fragment is used in your extension.

To extend a fragment in a container:

1. With your page open in the Page Designer, select the fragment on the canvas or in the page's Properties pane.

If the page has any accessible fragments, they're listed in the Components tab in the page's Properties pane. In the following image, the fragment name is the only extendable fragment in the dynamic container:



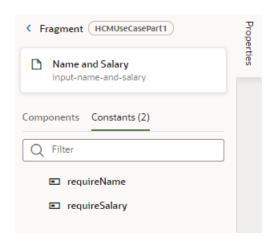
2. Click **Configure** in the Properties pane to open the fragment in the Fragment Designer.

The Properties pane displays a list of the extendable items in the fragment. You can click individual items to open them in the fragment's editor.

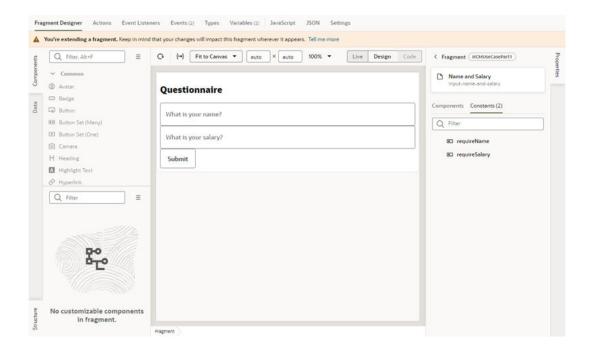


3. After the fragment is open in the Designer, you can customize the elements that the fragment's developer has made accessible.

In the Fragment Designer, the extendable components and constants are listed in the fragment's Properties pane. In this example, though the fragment contains components, the developer hasn't made any of them accessible, so they aren't listed in the Properties pane. The only constants that the fragment developer has made accessible (requireName and requireSalary) are listed in the Constants tab in the Properties pane:



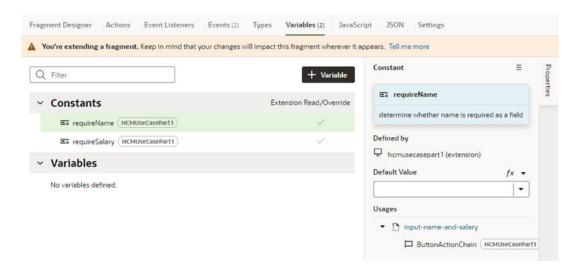
When extending a fragment in the Fragment Designer, or any of the fragment's other editor tabs, a warning is displayed at the top of the editors to let you know you're now extending a fragment:



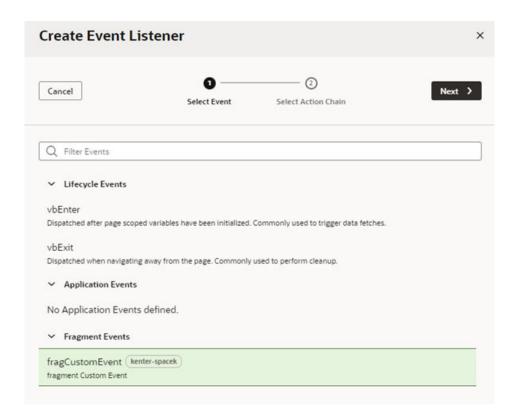
4. Configure the fragment in the fragment's editor tabs.

Once the fragment is open, you can configure it as needed. Some elements in the fragment will be accessible, as you can see in the image below of the fragment's Variables tab. The fragment developer has made two constants in the fragment extendable—extensions can set the default value of each constant:





Depending on the fragment, you might also want to use the other fragment editors to extend the fragment, for example, to add and configure variables and events, just as you would in a page. For example, in the fragment's Event Listeners tab, you can create an event listener for custom fragment events that the developer has made accessible. When creating the event listener, the accessible fragment events are listed in the Create Event Listener wizard. For more, see Work With Events and Event Listeners.



Part IV

Build an App UI or Fragment

Use the chapters in this part to create your own App UI, then deploy it to your Oracle Cloud Applications instance alongside other App UIs, whether built by Oracle or by others at your company. Or create a fragment, a reusable block of user interface components you can include throughout your extension.

From a development perspective, there's a great deal of overlap between App UIs and fragments. All of the chapters in this part apply to both types of assets, unless explicitly stated otherwise.

Topics:

- Develop an App UI or Fragment
- · Work With Pages and Flows
- Work With Components
- Work With Variables, Types, and Constants
- Work With JavaScript Action Chains
- Work With Events and Event Listeners
- Work With Resource Files
- Work With Code
- Work With the Diagram View
- Work With Fragments
- Common Use Cases



Develop an App UI or Fragment

An *App UI* is simply an application that includes Visual Builder pages and flows. Some App UIs are created by Oracle—like Oracle Cloud Applications—but you can build your own App UIs and deploy them as peers alongside Oracle's App UIs in your Oracle Cloud Applications ecosystem.

A *fragment* is a reusable user interface component that you can use across many App UIs. For example, suppose you want a common greeting in the form "Hello, <user name>" to appear at the top of several pages. Rather than building this code individually on each of the target pages, you can create a simple fragment and drop the fragment on the page in the appropriate location. Fragments also make it easy to bundle related objects together for layout templates.

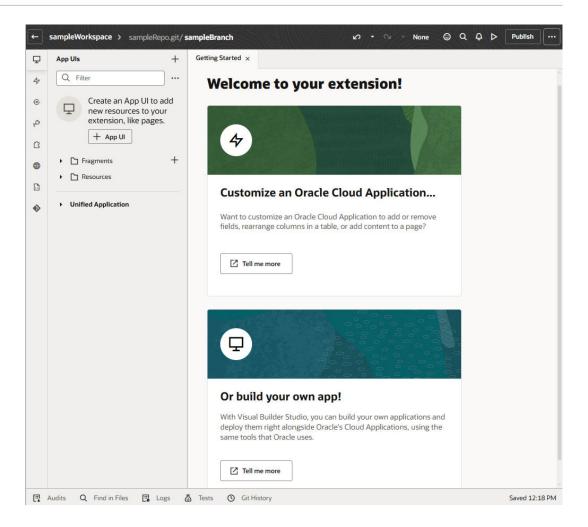
From a development perspective, the options available to you when building App UIs and fragments are nearly identical. Although a fragment can be as full-featured as an App UI, it is meant to be deployed *within* an App UI, as opposed to standing alone as a separate deployable unit.



As a best practice, it's a good idea to create only one App UI in each extension for typical scenarios.

This chapter—and all the chapters in this part—assume that you have some key concepts already under your belt, namely:

- You know how App UIs and/or fragments are used and why you might want to build one. If you don't, you may want to read What Is an Extension? and/or What Are Fragments?
- Before you can create an App UI or fragment, you must have a workspace which, among
 other things, provides access to a Git repository in which to store your work. If you're
 already in the Designer (that is, your screen looks something like the one shown below),
 that means you have a workspace (called myext, in this image) and you're ready to get to
 work.



If your screen doesn't look like this, you need to create an extension so you have your own workspace. See Create an Extension.

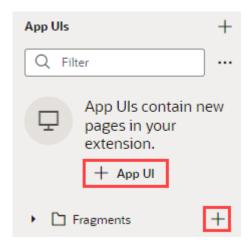
Once you've landed in the Designer, you have quite a few options:

- Take a moment to familiarize yourself with how App UIs and fragments are structured;
- Create an App UI or fragment, if you're ready to start developing;
- Import resources to use across your extension, or;
- Add another extension as a dependency, so you can employ its App UI's service connections, backends, global resources, and so on in your own App UI.

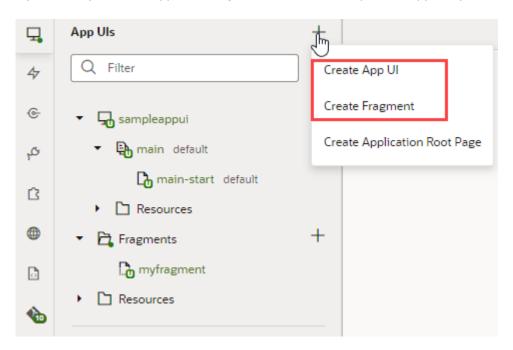
Create an App UI or Fragment

The steps for creating an App UI or fragment differ slightly, depending on whether you already have one of these assets in your extension. To get started:

If you haven't yet created an App UI or fragment in this extension, click + **App UI** or + next to **Fragments** in the App UIs pane:



If you already have an App UI or fragment, click + at the top of the App UIs pane:



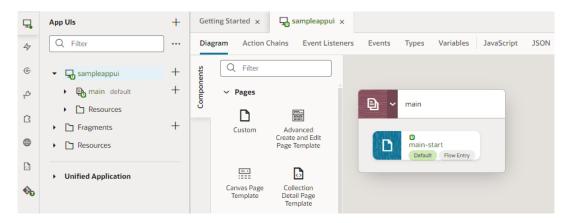
Note:

As a best practice, it's a good idea to create only one App UI in each extension for typical scenarios. This can help you isolate each App UI's code and manage deployments more easily.

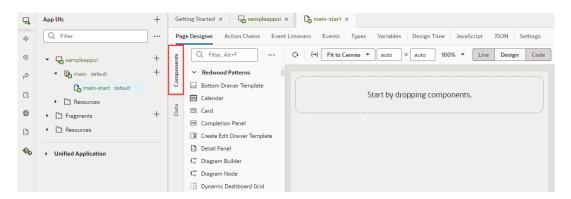
- For a new fragment, enter a Fragment ID. You are immediately placed in the Fragment Designer, where you can start developing your fragment using the chapters in Build an App UI or Fragment to help you.
- For a new App UI, enter a name in the **App UI Name** field. This name is by default added to your Unified Application's base URL, as in https://UnifiedAppName/redwood/AppUIName, to create the App UI's URL. However, you can use the **App UI ID** field to override the App UI name and supply your own string for the final portion of the URL, if

needed. Make sure you retain the x- prefix, which is used to distinguish customer App UIs from those created by Oracle in case they have the same name.

The first thing you'll see is a visual representation of your App UI on the Diagram tab:



The Diagram view can be quite handy as your App UI increases in complexity, as it displays your App UI's default pages, navigation flows, and even audit status at a glance. For now, though, you probably want to click main-start and start dropping components from the Components palette onto your page:



Use the chapters in Build an App UI or Fragment to help you build your App UI. When you're done, see Preview, Share, and Publish Your Extension to finish up your work.

Note:

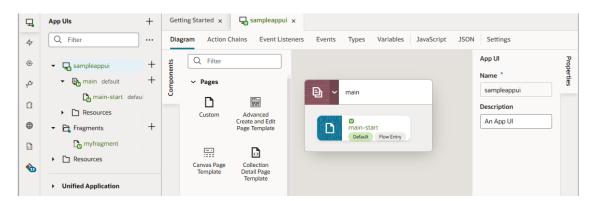
If you're having trouble previewing your page in the Page Designer, you might need to check your browser settings to make sure that the browser isn't blocking your VB Studio instance from accessing your Oracle Cloud Application instance. This can happen when the instances are not in the same host domain and your Chrome browser's settings are set to the default. To resolve this error, open the browser's Cookies settings page (chrome://settings/cookies) and disable Block third-party cookies, if it's not already disabled. You can also select Allow all cookies to disable the "Block third-party cookies" option.



How Are App UIs Structured?

If you've ever built a *visual application* with Visual Builder or VB Studio, you should already be familiar with the concepts needed to build or configure App UIs and fragments. If you haven't, you may want to quickly read through this topic before you start developing or configuring in earnest.

Let's begin by taking a look at a simple App UI in the Navigator:



In this example, we've created an empty extension called <code>sampleWorkspace</code> and added an App UI called <code>sampleappui</code> to it. All App UIs are represented in the Navigator by an expandable

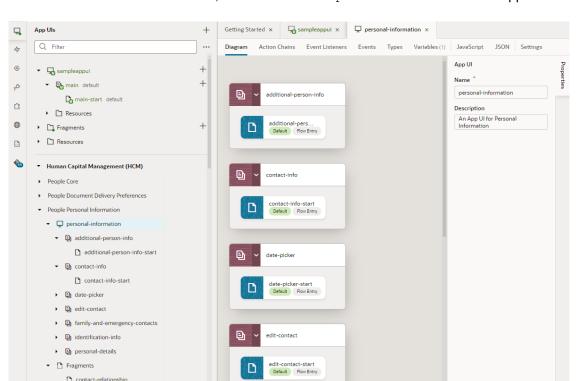


node.

There are several things to note about sampleappui, all of which apply to App UIs (and fragments) in general:

- When you first create an App UI (not fragment), a *flow* called main is automatically created for you. A flow is a group of one or more pages that is treated as an independent unit to perform some function in your application. For example, you might have a flow that contains the pages and artifacts used to register a new person. Each App UI can have as many flows as you need. Although the default flow's name is main, you can rename it by right-clicking the flow and choosing **Rename**.
- By default, the main-start page in the main flow is set as the default page in the flow, and the main flow is set as the default flow for the App UI. This means that they are the flow or page that is first invoked when your user interacts with your App UI. However, you can choose to have another page serve as the default for a flow, and/or choose another flow to be the default for the App UI. To do this, highlight the page or flow, then click Settings. The default flow and page within an App UI are always badged default for easier identification.
- A page (or fragment) typically contains one or more UI components, usually bound to data provided by a service connection.
- If you create a custom root page, the Application Root Pages folder contains pages that you can use to brand your App UI with a common header, footer, background, or other elements, to provide a consistent look and feel.
- Your extension has two Resources folders for images, style sheets (css), and JavaScript functions. The Resources folder under sampleappUI is available to that App UI only. The other Resources folder contains resources available to all App UIs and fragments in your extension.





Now let's say we add an extension as a dependency, for example, the Oracle Hcm People Personal Information Extension, which contains the personal-information App UI:

Unlike the newly created sampleappui, this App UI has multiple flows: additional-personinfo, contact-info, and so on, each with pages called additional-person-info-start and contact-info-start, and so on.

Now that you understand the basic structure of extensions and App UIs, you're ready for the concept of *scopes*.

What Are Scopes?

coworker-contacts-lov

Scope refers to how and where certain artifacts—like variables, action chains, root pages, and more—can be used within VB Studio.

In a nutshell, where you define something determines where it can be referenced. That is, within an extension, artifacts can be scoped at different levels:

- Extension scope: Artifacts are available to all App UIs, page flows, and pages in the
 extension. Examples: Root pages in the Root Page folder, as well as resources in the
 highest-level Resources folder. In addition, anything defined through the far left ribbon in
 the Navigator—Layouts, service connections, components, and so on—are available to
 everything in the extension.
- App UI/fragment scope: Artifacts are available to all the page flows and pages in the App UI or fragment. Examples: Resources in the App UI or fragment's Resources folder; action chains, events, event listeners, types, variables, and Layouts defined at the App UI or fragment level.
- Flow scope: Artifacts are available to only the page 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 establish settings that are scoped to various VB Studio entities, which control how the entity looks and behaves:

- Extension-Level Settings
- App UI Settings
- Flow Settings
- Page Settings

Add a Dependency

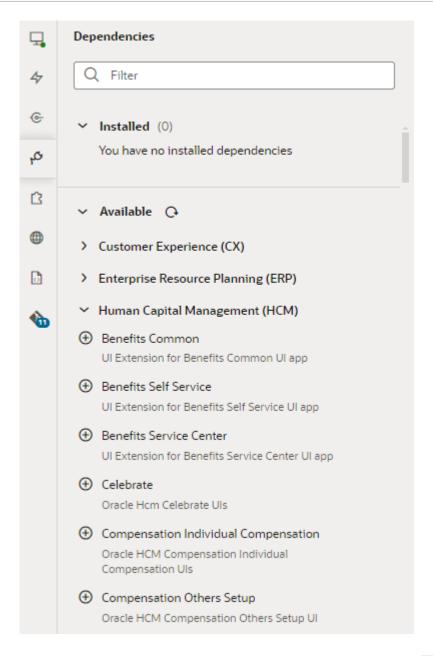
You add an extension as a *dependency* when you want to access the resources in the extension—like service connections, fragments, or Layouts—from your own App UI or fragment.

Be sure you understand the concepts discussed in What Are Dependencies? before continuing with this topic.

To add an extension as a dependency:

- 1. Click **Dependencies** in the left Navigator.
- 2. Locate the dependency you want in the Dependencies pane, where dependencies are grouped by pillar for easier identification:





To view dependencies published after the App UI was opened, click $^{\bigcirc}$ next to Available.

3. Click next to the extension you want. Remember, you don't add an individual App UI or fragment as a dependency, you add the extension *containing* the App UI.

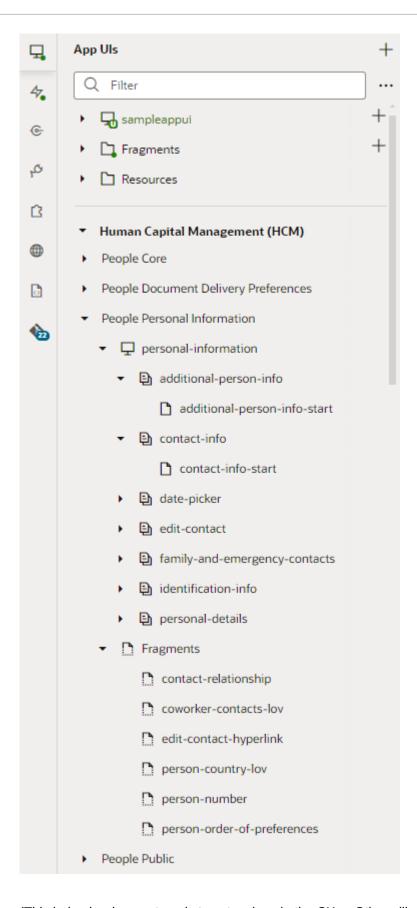
After you add an extension as a dependency, you'll see it listed under the Installed category on the Dependencies pane. You may also see a dependency's dependencies. For example, if an extension named <code>Extension1</code> depends on <code>Extension1a</code>, adding <code>Extension1</code> as a dependency also implicitly adds <code>Extension1a</code> as a dependency. If your extension uses resources such as fragments or service connections in <code>Extension1a</code>, the extension is explicitly added as a dependency.

(To ensure your extension continues to work, Extension1a remains a dependency as long as your extension uses its resources, even if Extension1 is removed as a dependency.)

If you switch to the App UIs pane, you'll see your dependencies listed by pillar, along with a bunch of other extensions that aren't really dependencies. That's because *all* extensions with

extendable resources in the same pillar as your dependent extension become available for you to explore. For example, when you add the HCM People Personal Information extension, you can explore resources within other extensions in the Human Capital Management pillar without actually adding those extensions:





(This behavior does not apply to extensions in the CX or Other pillars, where only the extensions you've added show.)



If you look at your extension in Source view at this point, you'll find that dependencies are not listed there until you have actually configured them in some way. In other words, they are not considered part of your extension until you have modified them in your Git repo.

You can now access resources such as fragments in your dependency's App UIs to suit your business needs. You can also use service connections and Layouts from these extensions that become available to you on the Services and Layouts panes.

When you finally publish your branch, the changes you make will be laid on top of the base App UI (which is the personal-information App UI in our example) at runtime.

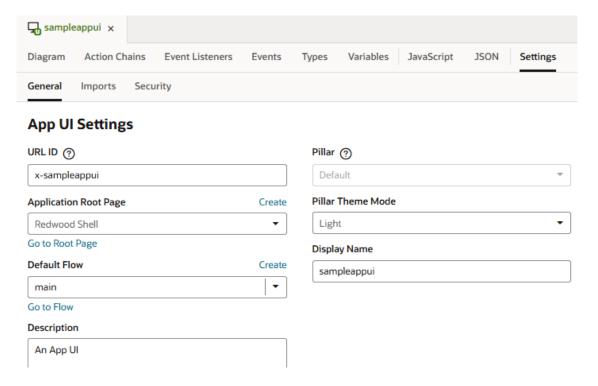
If for some reason you want to remove the dependency, you can—but remember that deleting a dependency once the extension is published may break App UIs that use that dependency. You'll see a warning to this effect. If the extension is yet to be published, you can safely remove the dependency so long as its resources are not used in your App UI. In fact, it's a good idea to remove dependencies that are not used anywhere in your extension for performance reasons.

To remove an installed dependency, hover next to the dependency's name under Installed in the Dependencies pane, then click \boxtimes . This option appears only for the extension you explicitly added as a dependency.

Establish App UI Settings

Because an App UI can include multiple pages and flows, settings at the App UI level apply to all the pages and flows within it.

To configure an App UI's settings, open the App UI, then click **Settings** to open the Settings editor:





Here's how you can use the different App UI-level settings:

Tab/Setting	Description
General tab	Contains general App UI settings:
URL ID	ID to use in the App UI's URL, as entered when the App UI was created. Most often, the App UI's ID and name are the same. The App UI's name is by default added to your Unified App's base URL to form its runtime URL, something like https://UnifiedAppName/redwood/AppUIName. If you don't want to use the App UI's name in the URL, enter an ID that will override the App UI name in the URL, as in https://UnifiedAppName/redwood/YourAppUiId. Make sure you retain the x-prefix, which is used to distinguish customer App UIs from those created by Oracle in case they have the same name.
Application Root Page	Root page for the App UI. A root page is a special type of page which provides the entry point for an App UI. It typically contains a shell with common elements like headers, footers, and backgrounds, besides other navigation elements that appear on every page in your App UI. The default root page for an App UI is the Unified App's redwood page, which invokes the Ask Oracle shell by default. The Ask Oracle shell provides a common experience for every App UI in your extension, with global elements such as a Navigator menu, search, and a user menu. The redwood page which defines this shell cannot be updated. If you want to use a different shell for your App UI, say to provide your own global header that will appear on all App UI pages, you need to define an alternate root
	page and make it available to your extension. See Brand Your App UI with a Custom Root Page.
Default Flow	Default flow used by the App UI. Every App UI has a default flow, and every flow has a default page. When you first create an App UI, the main flow is automatically created for you and set as the App UI's default flow. The main-start page, also created automatically in the main flow, is set as the flow's default page. When you run the App UI, the default page in the default flow is rendered. If you create other flows and want the App UI to open to a page in a different flow, select that flow as the default. You can also click Create to create a flow directly from here, then select it as the default. To change the default page in the default flow, go to Flow Settings.

Note:

The default flow is always added to the Oracle Cloud Application or Ask Oracle Navigator menu (depending on the application you're extending), allowing users to directly access the default page in the default flow under the App UI when they open the menu. In addition, navigation from other App UIs is always enabled to the default page in the default flow. Make sure then that **Default Flow** reflects the flow (and the page) you want displayed in the Navigator menu. Changing the default flow automatically updates your entry in the Navigator menu.

You can also add other flows alongside the default flow in the Navigator menu by selecting **Add to Oracle Cloud Applications menu** in that flow's settings.

Tab/Setting	Description
Description	Description of the App UI.
Pillar	The App UI's product family or pillar (for example, HCM or CX in the Oracle Cloud Applications ecosystem), inherited from the Pillar setting at the extension level.
Pillar Theme Mode	Mode used by the selected pillar. Each pillar has a corresponding mode which changes the Redwood texture stripe. Select a mode, either Light or Mixed, to go with your pillar theme.
Display Name	Name displayed for the App UI in the Ask Oracle Navigator menu, if you mark a flow/page to show in the Ask Oracle menu. See Manage Flow Settings and Manage Page Settings.
Imports tab	Contains settings to manage resources such as custom CSS files, modules, and components imported at the App UI level, allowing you to create declarative references that can be shared between flows and pages in the App UI. See Create Declarative References to Imported Resources.
Security tab	Allows you to add permissions that control user access to the App UI (as well as individual flows and pages in the App UI). Only users granted one of the assigned permissions can navigate to the App UI, flow, or page. See Control Access to Your App UI.

Brand Your App UI with a Custom Root Page

A *root page* brands the pages in your App UI with a common header, footer, background, or other elements, to provide a consistent look and feel.

By default, the root page for all your App UIs is the redwood root page, which is provided by the Unified Application. The redwood page contains design elements from Oracle's new Redwood design platform, and looks like this:

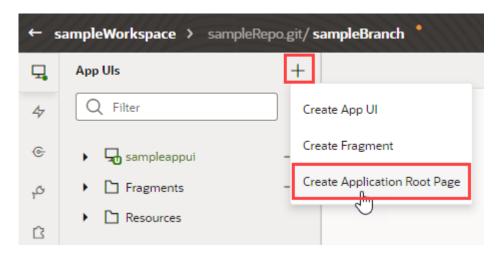


The Ask Oracle components on the shell page include:

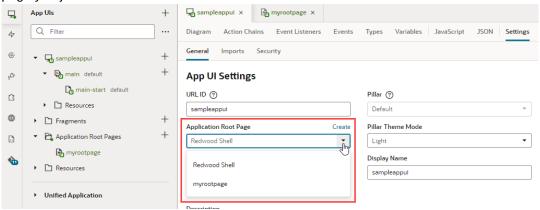
A	Search, offering type-ahead support to filter the Navigation List
В	Navigation List, a menu of suggested applications
	User Profile, providing access to preferences, help, log out, and other user- specific actions

You can override the redwood root page by creating your own custom page, then specifying that page in the App UI's Settings panel. To do this:

1. In the Apps UI pane, click +, then select Create Application Root Page:



- 2. In the Create Page dialog, enter a name for your root page and click **Create**.
 - The root page is added to the **Application Root Pages** folder.
- 3. Design your page, taking care to make certain areas customizable by your users, if desired. Certain Redwood components are available for you to use directly from the Components palette; the Components Exchange also has elements you may want to use.
- 4. Select the App UI node in the Navigator, then click **Settings**.
- 5. In the **Application Root Page** field, use the drop-down menu to apply the custom root page you just created.



Alternatively, you can create a custom root page directly from the Settings page, by clicking **Create** next to the Application Root Page field.

Defining a custom root page adds the hostRootPage property to the App UI's JSON metadata. For example, here's what you might see in the app.json file:

```
{
  "id": "sampleappui",
  "urlId": "sampleappui",
```



```
"hostRootPage": "myrootpage"
}
```

The hostRootPage's value myrootpage identifies your custom root page's ID. This value can also be an expression. Say you wanted to switch root pages between anonymous and authenticated users; you would create two root pages: one called anonymousRoot for anonymous users and another called authenticatedRoot for authenticated users, then update hostRootPage as follows:

```
"hostRootPage": "{{ $application.user.isAuthenticated ?
'authenticatedRoot' : 'anonymousRoot' }}"
}
```

Remember, only \$application, \$global, \$initParams, and existing global JavaScript variables can be evaluated at the time the expression needs to be resolved.

Note:

A custom root page is typically used by App UIs within the same extension. To use a custom root page *across* extensions, you must configure the App UI in the other extension to point to the extension where the custom root page is defined. To do this:

- 1. Open the App UI in the other extension and select its **JSON** tab.
- **2.** Add the hostRootPage property to the JSON metadata in the format "hostRootPage": "extId: customRootPage", where:
 - extId is the ID of the extension where the custom root page is defined (look for this value in the extension's Settings), and
 - customRootPage is the name of your custom root page.

For example:

```
{
  "id": "sampleappui",
  "urlId": "sampleappui",
    ...
  "hostRootPage": "site_myAppUiExtension:myrootpage"
}
```

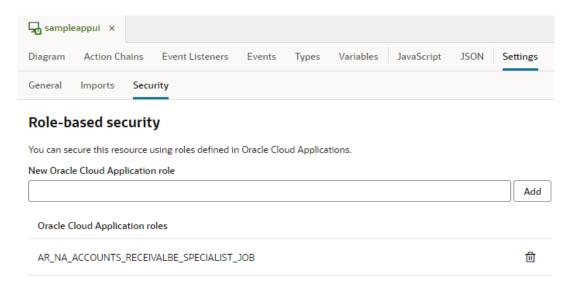
Control Access to Your App UI

You can use *roles* defined in Oracle Cloud Applications to control access to your App UI, as well as individual flows and pages in the App UI. Only users granted one of these roles will be able to navigate to the App UI, flow, or page.

Roles can be assigned in the Settings editor at the App UI, flow, or page level, but remember that access is inherited from the parent. So a page inherits access permissions from the parent flow, and a flow inherits access permissions from the App UI.

To control access to an App UI, flow, or page:

- Select the App UI, flow, or page in the Navigator.
- Click Settings, then Security.
- 3. In the **New Oracle Cloud Application role** field, enter the name of a role defined in Oracle Cloud Applications. If you're a developer, talk to your security administrator to know which roles you need to control access in your App UI.



Make sure you use the correct spelling for the role. No validation is done to confirm the role is correct.

The roles you enter are assigned by security administrators and include privileges that grant access to tasks that someone performs as part of a job. As a security administrator, you can use the Security Console to view the privileges that are assigned to roles. You can review roles assigned to a user as well as create roles.

4. Click Add.

Work With Pages and Flows

As a fully functional app, an App UI can contain multiple flows, each of which can contain multiple pages. Every App UI 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 to do some function in your App UI. For example, you might have a flow that contains the pages used to register a new person. Your App UI's pages are what your users see and interact with. You can build just about any type of page in VB Studio—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 service connections.

VB Studio 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.



App UIs and fragments are developed pretty much the same way—you add UI components to a page or fragment and wire them to action chains and events, but pages and flows apply only to App UIs, not fragments. Fragments are meant to store page sections and components, not pages and flows.

Your App UI can access data from any data source available to your extension. It can be a source defined in a dependency, or a service connection you've added to your extension, including those used to access custom objects for Oracle SaaS applications.

Create and Manage Pages

When you first create an App UI, a starter page called main-start is created for you within the main flow. You can create more pages under the main flow, or create a new flow and add pages to that flow. Every flow you create, by default, has its own starter page.

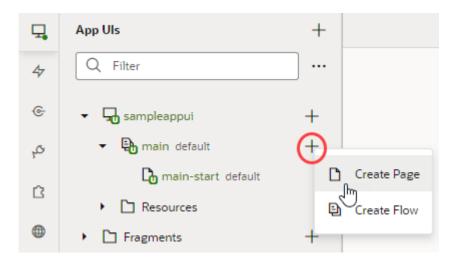


Tip:

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 patterns as well as fragments in your extension.

To create an empty page:

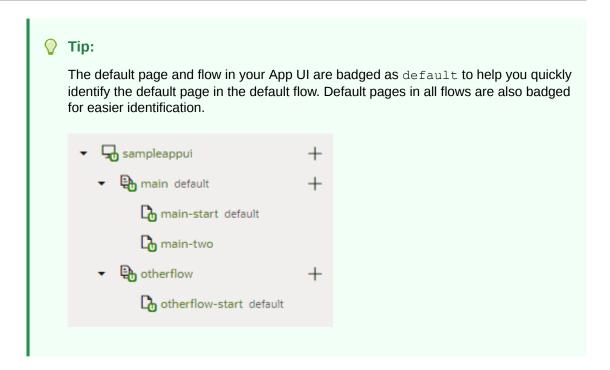
- 1. Open your App UI and expand the flow where you want to add a page.
- 2. Click the Create Page icon (+) next to the flow, then select Create Page.



3. 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. Click **Create**.

After the page is created, you can customize it as needed in the Page Designer. You can also duplicate, rename, even delete the page needed. Right-click the newly created page in the Navigator and select an action.

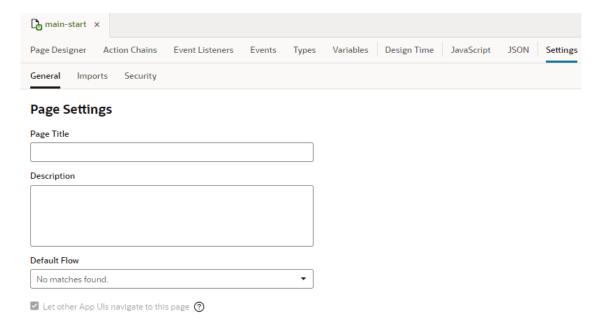
By default, the main-start page in the main flow is set as the default page in the flow, and the main flow is set as the default flow for your App UI. This means that when the App UI is run, the main-start page is rendered. If you want any other page to be rendered, change the Default Page setting in the flow's Settings editor.



Manage Page Settings

Each page in your App UI includes a Settings editor, which you use to primarily manage imported resources such as custom components, CSS files, and modules. You can also allow other App UIs to navigate to a page other than the default page of the default flow.

To configure settings for an App UI'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	Contains 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 .
Let other App UIs navigate to this page	 Option to allow App UIs in your Oracle Cloud Application instance to navigate to this page when configuring the Navigate action. For the default page in the default flow, this option is selected by default and cannot be changed. So when an App UI is selected in the Navigate action, navigation is always enabled to that App UI's default page in the default flow. For all other pages, you must explicitly select this option to enable navigation to that page.
	Note: Because this option exposes your page to other App UIs, you can no longer rename, delete, or change any input parameters marked as required for the page, as extensions may depend on them.
Imports tab	Contains settings to 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 Create Declarative References to Imported Resources.
Security tab	Allows you to add permissions that control user access to the page. Only users granted one of the assigned permissions can navigate to the page. Note that permissions are inherited from the parent, so the page inherits permissions from the parent flow. See Control Access to Your App UI.

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	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)	Grid	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	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 popups. 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:

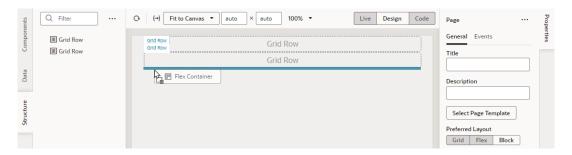
 Drag the layout container or component from the Components palette and place it on the canvas.

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:

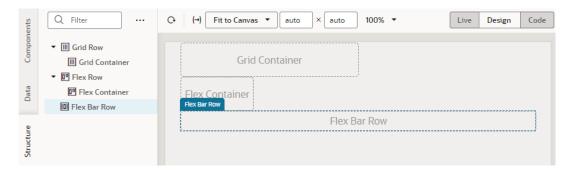


If you were dragging the same component on a page that uses the Block layout, your view might look like this:

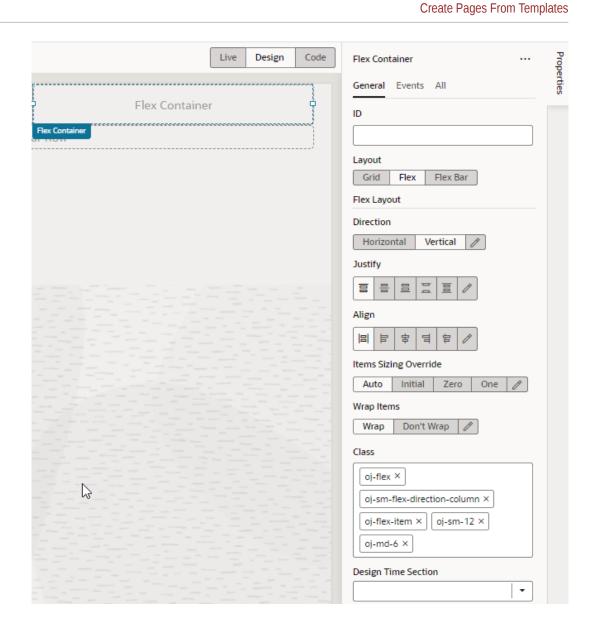


Notice how the Grid Container components in the Structure view aren't automatically enclosed in rows. That's because unlike the Grid or Flex layout, components dropped on a page that uses the Block layout aren't automatically wrapped in Grid or Flex rows.

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.



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 App UI, 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 App UI'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.

Create Pages From Patterns

You can leverage Redwood resources in your App UI 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.



Creating a page based on a pattern is supported only if your app is leveraging version 2301 or later of the Redwood components.

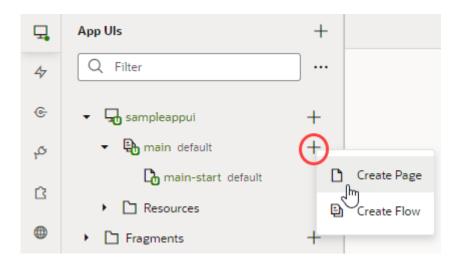
- 1. To add a pattern to your App UI:
 - a. Open the Components tab () in the Navigator.
 - **b.** Select the **Browse** tab and search for "redwood pattern".

Double-click a pattern to open it in the canvas area and learn more about it. You can view detailed information such as its features and steps on how to use it. Take note also of the **VB Characteristics** detail that tells whether the component will show in the Components palette.

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.

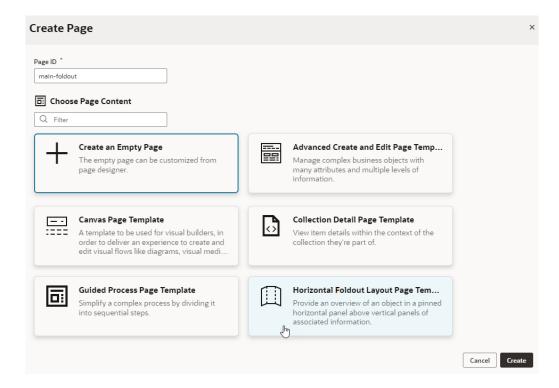
Once you install a pattern, it becomes available to you when creating a page.

- 2. To create a page based on a pattern:
 - a. Open your App UI 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 extension (and its dependencies) 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:

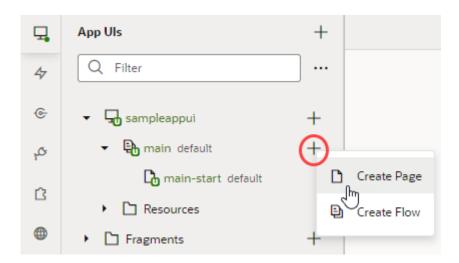
- Update your Redwood components to the latest version.
- Migrate your extension's runtime dependencies.
- Clear your browser cache.

Create Pages From Fragments

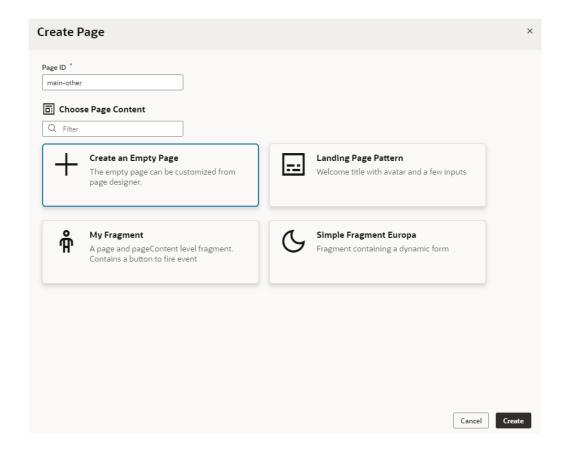
You can create pages starting with the contents of a fragment, essentially using the fragment as a page template. You can do this so long as the fragment is available for use in pages.

- 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 App UI 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.

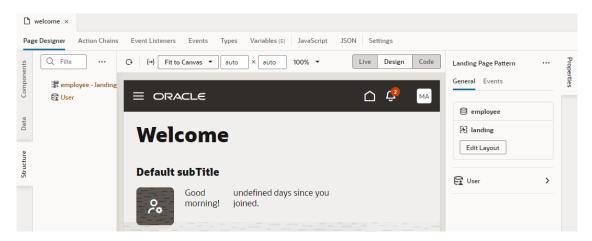


- **c.** Enter the name of the new page in the **Page ID** field of the Create Page dialog box. By convention, a page name has its flow name as a prefix.
- d. Under Choose Page Content, select the fragment that you want to use with the page.



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.

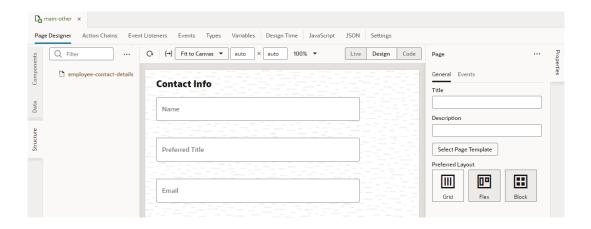


Caution:

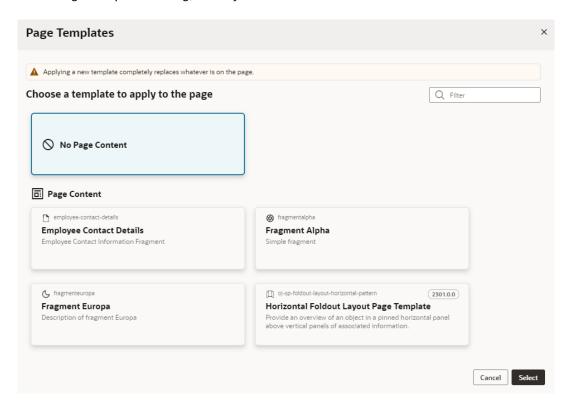
Applying a new template removes everything on the page, including associated files—it's as if the page was deleted and a new one created with the same name. Previous user work is removed and all content is reset to an initial state (which may be an empty page or a basic template). Proceed with caution.

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:



- 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.

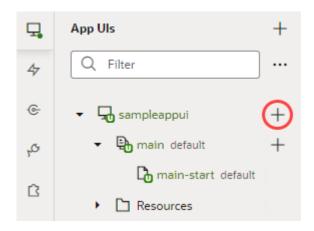
Create and Manage Flows

When you first create an App UI, the main flow is automatically created for you and set as the default flow for the App UI. You can create other flows as needed, even a flow within a flow.

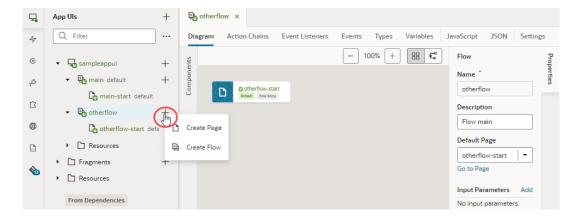
To create a flow in your App UI:

default page.

1. Open your App UI, then click the Create Flow icon (+) next to the App UI node:



- In the Create Flow dialog box, enter a name for the flow and click Create.Creating a flow automatically creates a start page for that flow and sets it as the flow's
- 3. Expand the flow to see pages and artifacts within a flow and make your selection:



Artifact	Description
Flow artifact	Open the flow to edit metadata such as variables, types, action chains, and JavaScript functions that can be used on every page in the flow. If you don't want the starter page created automatically as the default page in the flow, use the flow's Settings editor to change it.
	To create a new page in the flow, click the Create Page icon (+) next to the flow artifact, then Create Page .
	To create sub-flows that let you navigate between pages in the sub-flow without leaving the page containing the sub-flow, see Embed a Flow Within a Page.
Page artifact	Open each page to edit the page's layout and other page metadata.

You can duplicate, rename, or delete a flow if needed by selecting an action from the flow's right-click menu.

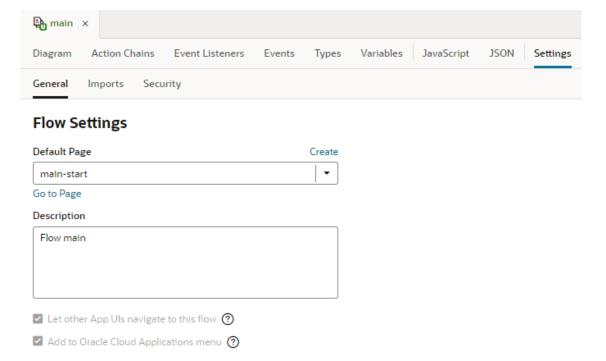
Each flow has access to the App UI's Resources folder containing images and CSS files that can be used in the flow's pages, in addition to the Global Resources folder available to all App UIs in the extension. See Work With Resource Files.

Manage Flow Settings

Each flow in your App UI 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. You can also allow other App UIs to navigate to a flow other than the default.

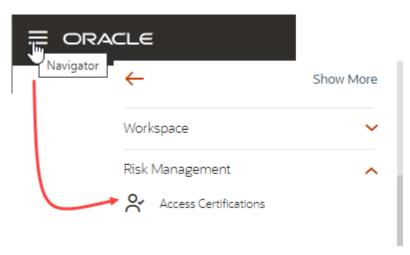
Because an App UI can have multiple flows, you configure settings for each flow individually, so your settings apply only to pages within that flow.

To configure settings for an App UI's flow, open the flow, then click **Settings** to open the Settings editor. For example, here's what you might see for the default main flow:



Here's how you can use the different flow-level settings:

Setting	Description
General tab	Contains general flow settings:
Default Page	Default page of the flow. Every App UI has a default flow (defined in the App UI'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 UI 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.
Add to Oracle Cloud Applications menu	Option that adds this flow to the Oracle Cloud Application or Ask Oracle Navigator menu (depending on the application you're extending), so users can directly access the default page in this flow under your App UI when they open the menu:



- For a default flow, this option is selected by default, so the default page in the default flow always displays in the Navigator menu.
- For a non-default flow, you must explicitly select this option to display the default page in that flow in the Navigator menu.

Let other App UIs navigate to this flow

Option to allow App UIs in your Oracle Cloud Application instance to navigate to this flow when configuring the Navigate action.

- For a default flow, this option is selected by default and cannot be changed. So when an App UI is selected in the Navigate action, navigation is always enabled to that App UI's default page in the default flow.
- For a non-default flow, you must explicitly select this option to enable navigation to that flow. If **Add to Oracle Cloud Applications menu** is selected for a flow, then this option is automatically selected to allow navigation to the default page in this flow.



Because this option exposes your flow to other App UIs, you can no longer rename, delete, or change any input parameters marked as required for the flow, as extensions may depend on them.

Setting	Description
Imports tab	Contains settings to manage resources such as custom CSS files, modules, and components imported at the flow level, allowing you to create declarative references that can be shared among pages in the flow. See Create Declarative References to Imported Resources.
Security tab	Allows you to add permissions that control user access to the flow. Only users granted one of the assigned permissions can navigate to the flow. Note that permissions are inherited from the parent, so the flow inherits permissions from the App UI. See Control Access to Your App UI.

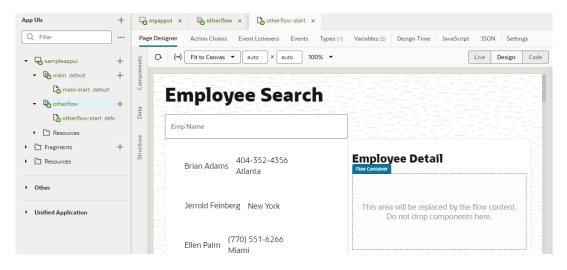
Embed a Flow Within a Page

Each flow in your App UI can contain one or more sub-flows, allowing you to embed pages within other pages. To do this, you define a sub-flow inside another flow, then embed the sub-flow within a Flow Container component on the page that serves as the "parent" flow.

Embedded flows (also known as nested flows) let you isolate content from the page containing the flow and allow navigation between pages in the sub-flow without leaving the page containing the sub-flow. This is useful in Single Page Applications (SPAs), where it allows you to stay within the page and present contextual information in a section of the page. It also makes for reuse as the same sub-flow can be embedded in multiple pages in the parent flow. Note though that only one flow can be embedded in a page and the URL represents the current flow structure.

To embed a flow in a page:

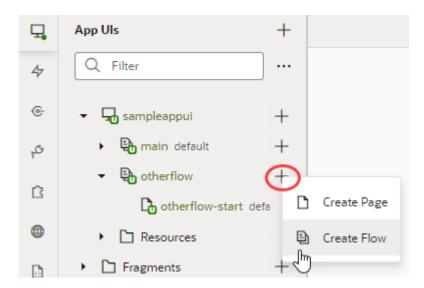
- Open the page where you want to embed a new flow (for example, the otherflowstart page under otherflow).
- 2. In the Page Designer, drag the Flow Container component from the Layout category in the Components palette and place it on the canvas.



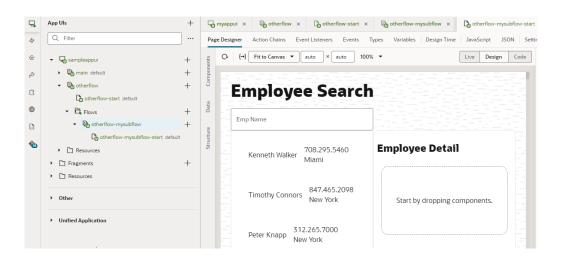
For the page containing the embedded flow—let's call it the parent container page—you can only edit content outside the Flow Container component; the content of pages you embed won't be visible on this canvas.

3. Create a sub-flow and design its UI to meet your requirements:

a. Click Create Page (+) next to the flow containing the parent container page (otherflow in our example). Select Create Flow.



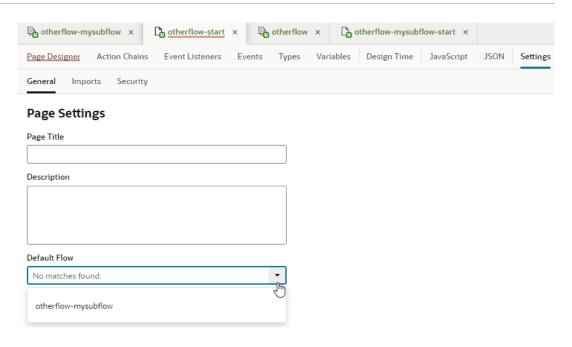
- b. Enter a name for the new sub-flow in the Create Flow dialog box (for example, otherflow-mysubflow). By convention, a sub-flow takes the parent flow name as a prefix. Click **Create**.
- c. Edit the sub-flow's starter page in the Page Designer to design the UI. To help you visualize a page in a sub-flow, the canvas displays the content of the parent page alongside content of the sub-flow page, but you can only edit the sub-flow's page content in this view.



You can add more pages to the sub-flow as needed.

4. When you are done designing your sub-flow, set it as the default flow for the parent container page. to do this, open the parent container page's Settings editor and select the sub-flow as the Default Flow in the General tab.

The Default Flow drop-down list displays all flows within the current flow.



5. Run the App UI to preview the embedded content in the page.

Display SaaS Data In Your App UI

You can create pages that interact with built-in as well as custom business objects in your Oracle Cloud Applications (for example, HCM or Help Desk) and run them directly on your Oracle SaaS infrastructure.

Let's say you want to show employee information in HCM, you can create a catalog-based service connection to access built-in objects and interact directly with HCM endpoints in the Oracle Cloud Application service catalog. Now suppose you want to track parking spaces for employees in HCM, you'd then use Application Composer to define a custom object for employee parking, then create an ADF Describe-based service connection to access that custom object.

There are, however, some limitations to this approach:

- You can define third-party REST services only if they don't need authentication.
- You cannot allow users who don't have access to your production Oracle Cloud Applications environment to access the App UI anonymously.

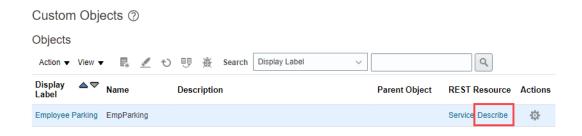
If your application requires these capabilities, you'll need access to a separate Visual Builder cloud instance.

Let's take a look at how you can create App UI pages that interact with custom objects:

1. In your Oracle Cloud Application environment, select a sandbox that allows you to access the Application Composer and define your custom objects. For details on how to create a custom object and add fields to it, see Define Objects and Define Fields in Configuring Applications Using Application Composer.

For example, assume you've defined an employee parking custom object, with the Employee Name and Assigned Parking Spot as fields. To use this object in your own pages, you'll need its Describe file, which can be found via the custom object's Describe link, as shown here:

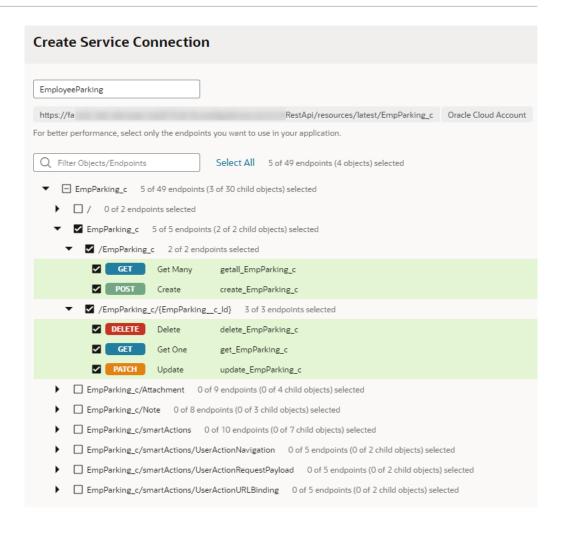




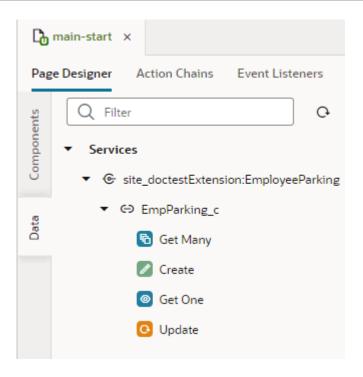
Click the link to open the Describe file in a new tab, then copy its URL, something like https://host-demol.oracledemos.com/hcmRestApi/resources/latest/EmpParking c/describe.

- 2. In your VB Studio environment, define a service connection to access the custom object, then use your App UI's pages to access the object's data.
 - a. Click Services in the Navigator.
 - **b.** In the Services pane, click the **+** sign and select **Service Connection**.
 - c. Click **Define by Specification** in the Create Service Connection wizard.
 - d. Enter a name for your connection in the Service name field, select the API Type as ADF Describe, then in the Web address option, paste the URL to the object's Describe file that you previously copied. Select your authentication option, then click Next.
 - e. Select the custom object's endpoints you want to add and click **Create**.

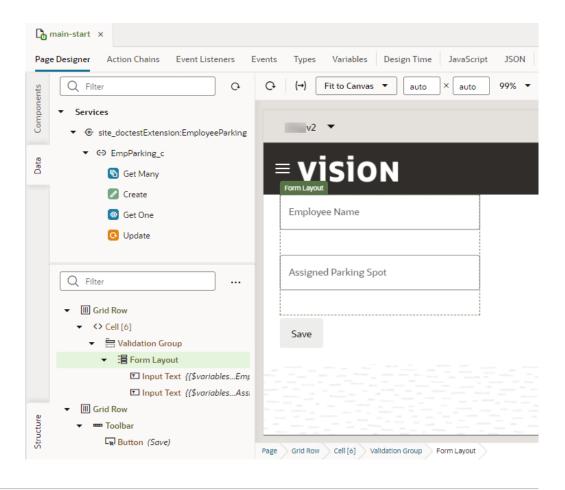




- f. Switch to the App UIs pane, then select your App UI.
- g. Create a new page or open an existing one in the Page Designer.
- h. Click the **Data** palette and locate your custom object. The **Data** palette lists the accessible endpoints from your extension's dependencies and service connections.



i. Drag and drop the object (or a particular endpoint) onto the page canvas, then select the component you want use to display data. For example, you might display the object's data as a Create Form containing the Employee Name and Assigned Parking Spot fields.



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, both within and across App UIs.

Navigate Between Pages in the Same Flow

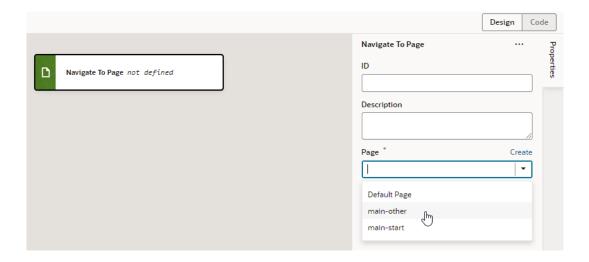
To navigate between pages in the same flow, you associate a page component with an event that sets off a navigation action chain.

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:

- 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.
- 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).
 - 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 App UI to test navigation between the two pages.

Navigate Between Pages in Different Flows

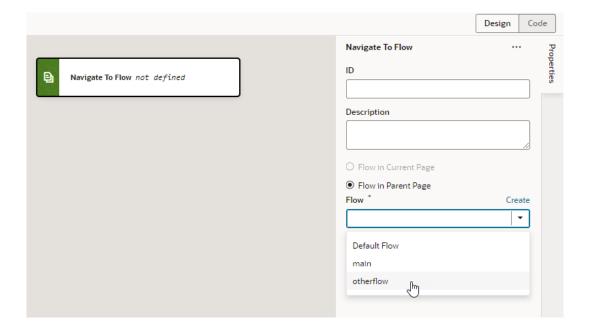
Navigating between pages in different flows within an App UI is similar to how you'd navigate pages within the same flow, but instead of selecting the page to navigate to, you select the flow containing the page.

To navigate between pages in different flows within an App UI (for example, to navigate from myApp/main/main-start to myApp/otherflow/otherflow-newpage):

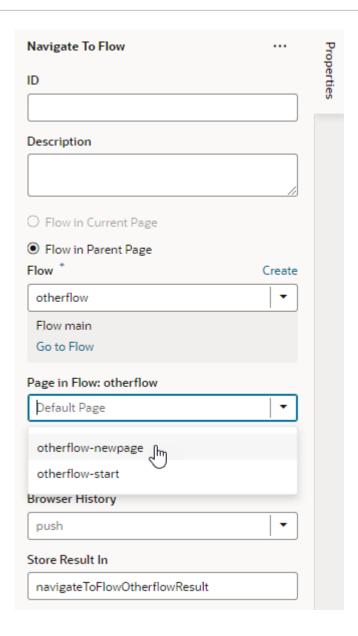
- Open the page you want to navigate from (main-start, for example).
- 2. In the Page Designer, drag a component that you want to set off navigation and drop it onto the page canvas. Here's an example of a tab bar, with each tab meant to navigate to a different flow:



- 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.
- 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.
- 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 App UI to test navigation from one page to another in a different flow.

Navigate Between Pages in Different App UIs

It's possible to navigate from pages in one App UI to pages in another App UI.

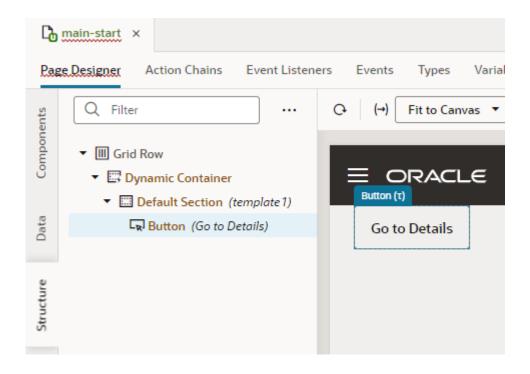
Note:

If you want to allow navigation to a page in an App UI that isn't the default page of the default flow, navigation to that page and its flow must be enabled in the page- and flow-level Settings editors. To do this:

- 1. Open the page in the App UI that you want to allow other App UIs to navigate to.
- 2. Click **Settings** to open the page-level Settings editor.
- 3. Select Let other App UIs navigate to this page.
- 4. Open the flow that this page belongs to.
- 5. Click **Settings** to open the flow-level Settings editor.
- 6. Select Let other App UIs navigate to this flow.

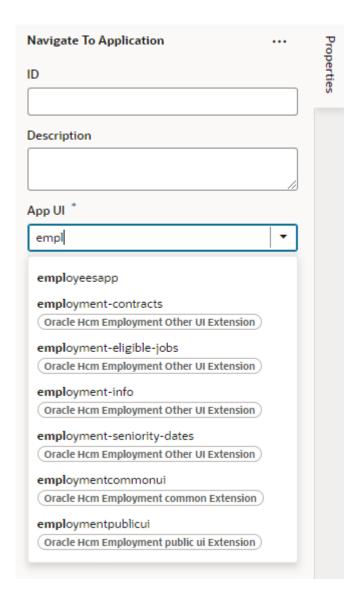
To navigate from a page in your App UI to a page in another App UI:

- 1. Open the page you want to navigate from (main-start, for example) in your App UI.
 - It's also possible to navigate from a fragment consumed by a page or a layout to a page in another App UI. In this case, remember to set up navigation in the fragment or layout rather than the page. See Navigate From Fragments and Layouts to App UIs.
- 2. In the Page Designer, drag a component that you want to set off navigation and drop it onto the canvas. Here's an example of a button added to a dynamic container template in a page:

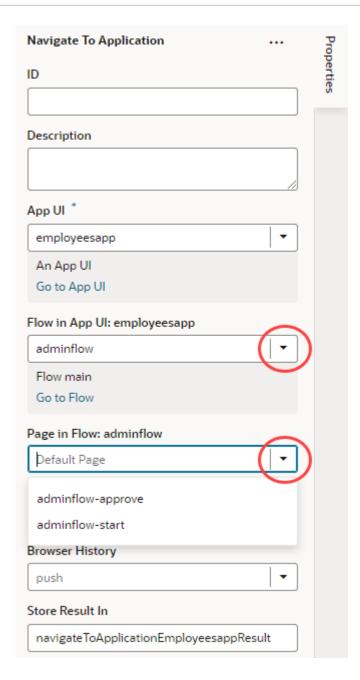




- 3. Select the component (for example, the button), then click the **Events** tab in the Properties pane.
- Click the + Event Listener button and select On 'ojAction', the default action for a button click. You might see other options suggested for your particular component.
- 5. When a new action chain is created, drag the **Navigate to Application** action from the Navigation section of the Actions palette and drop it onto the canvas.
- 6. In the Navigate to Application action's properties, select the App UI you want to navigate to in the App UI list. Start to enter the name of the App UI you want (for example, employeesapp) to filter the results.



You'll be able to navigate to any published App UI, in effect the default page in the default flow of the App UI. If you want to navigate to a non-default page in the App UI, you'll need to drill down further to select a flow, then a page in the flow:



Remember that only pages and flows marked as navigable in the page- and flow-level settings will be available for selection.

7. Preview your App UI to test navigation.

Navigate From Fragments and Layouts to App UIs

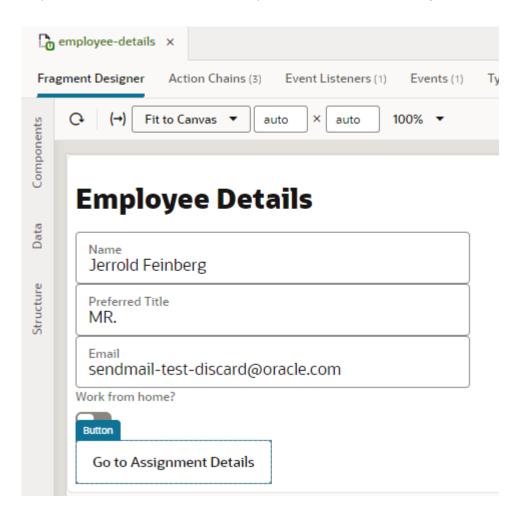
When you use fragments or layouts in an App UI's pages, you can navigate from those fragments or layouts to other App UIs.

Navigation from a fragment or layout can only be to other App UIs. That's because the fragment or layout is unaware of the structure of the application where it is used. As a result, you need to raise custom events to allow the page that consumes the fragment or layout to navigate to the fragment or layout. See Raise Fragment or Layout Events that Emit to the Parent Container.

Note:

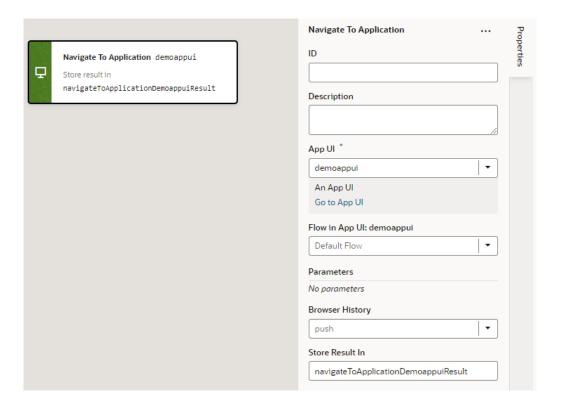
If you want to allow navigation to a page in an App UI that isn't the default page of the default flow, navigation to that page and its flow must be enabled in the page- and flow-level Settings editors. To do this:

- 1. Open the page in the App UI that you want to allow other App UIs to navigate to.
- 2. Click **Settings** to open the page-level Settings editor.
- 3. Select Let other App UIs navigate to this page.
- 4. Open the flow that this page belongs to.
- 5. Click **Settings** to open the flow-level Settings editor.
- Select Let other App UIs navigate to this flow.
- To navigate from a fragment consumed by your App UI's page to a page in another App UI:
 - Open the fragment you want to navigate from in your App UI (for example, employee-details).
 - 2. In the Fragment Designer, drag a component that you want to set off navigation and drop it onto the canvas. Here's an example of a button added to a fragment:





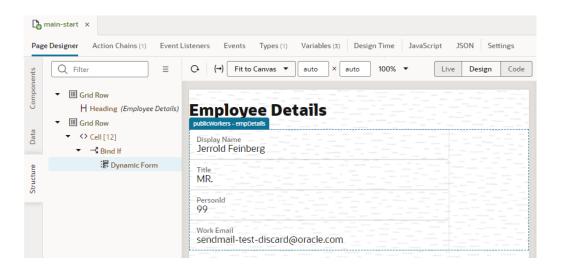
- 3. Select the component (for example, the button), then click the **Events** tab in the Properties pane.
- 4. Click the **+ Event Listener** button and select **On 'ojAction'**, the default action for a button click. You might see other options suggested for your particular component.
- **5.** When a new action chain is created, drag the **Navigate to Application** action from the Navigation section of the Actions palette and drop it onto the canvas.
- 6. In the Navigate to Application action's properties, select the App UI you want to navigate to in the **App UI** list. Start to enter the name of the App UI you want (for example, demoappui) to filter the results.



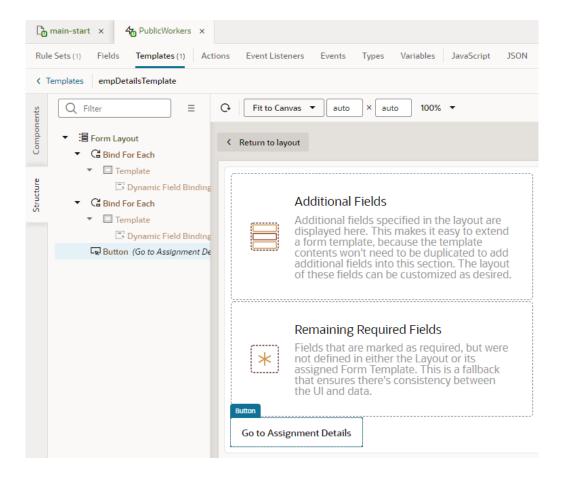
If you want to navigate to the default page in the default flow of the App UI, you can stop here, but if you want to navigate to a non-default page, you'll need to drill down further to select a flow, then a page in the flow. Remember that only pages and flows marked as navigable in the page- and flow-level settings will be available for selection.

- To navigate from a layout in your App UI's page to a page in another App UI:
 - 1. Open the page you want to navigate from in your App UI (for example, main-start).
 - 2. In the Page Designer, drag a dynamic component onto the canvas and create your layout. Here's an example of a dynamic form that uses a template to show employee fields:



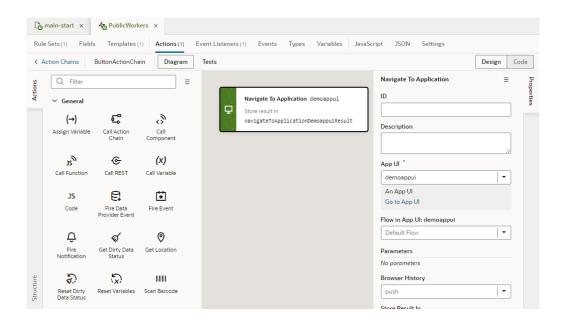


3. Because our example uses a dynamic form template, the component that set off navigation must be defined in the template. To do that, go to your Layout, select the dynamic form template in the **Templates** tab, then drag and drop the navigation component onto the template canvas. Here's an example of a button added to the empDetailsTemplate in the PublicWorkers layout:



4. Select the component (for example, the button), then click the **Events** tab in the Properties pane.

- 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.
- **6.** When a new action chain is created, drag the **Navigate to Application** action from the Navigation section of the Actions palette and drop it onto the canvas.
- 7. In the Navigate to Application action's properties, select the App UI you want to navigate to in the **App UI** list. Start to enter the name of the App UI you want (for example, demoappui) to filter the results.



If you want to navigate to the default page in the default flow of the App UI, you can stop here, but if you want to navigate to a non-default page, you'll need to drill down further to select a flow, then a page in the flow. Remember that only pages and flows marked as navigable in the page- and flow-level settings will be available for selection.

Work With Components

Components are the basic building blocks of your App UI'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 App UI'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 Components to Pages

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.

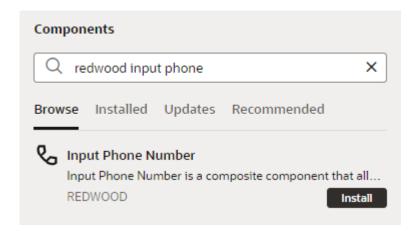
The Components palette contains an extensive list of Oracle JET components that you can add to your page, including dynamic components. Because dynamic components help you develop UIs that dynamically change what's displayed to users based on your own rules, working with them also involves *layouts* and *display logic*. See Add Dynamic Components to Pages.

VB Studio 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:

The next generation of JET components, known as Core Pack components, are available for use in your extension's pages. See Work With JET Core Pack Components on how you can take advantage of them in your pages.

- 1. If you want to use Redwood components in your page, follow these steps:
 - a. Explore the Redwood catalog of components, templates, and patterns and take note of what you'd like to use in your page.
 - b. Open the Components tab () in the Navigator.
 - 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).

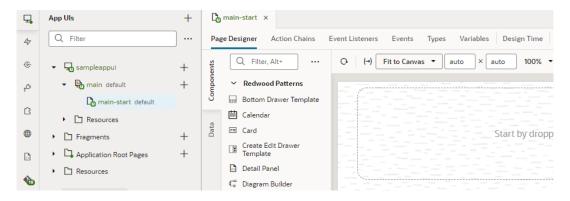
Once you install a component, it will become available to you in your page's Components palette. Simply filter to find the component.



If the component does not show in the Components palette, go to the Navigator's Components tab, locate the component, and check its **VB Characteristics** detail to see why.

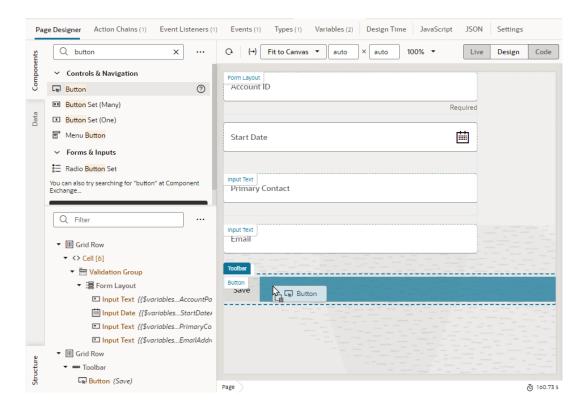
Open the page you want to work with in the Page Designer.

When you first create an App UI, the main flow with the main-start page is created automatically for you. Expand the main node under your App UI in the tree view and select main-start if this is the page you want to work with:



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 VB Studio.

From the Components palette, drag a component 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.

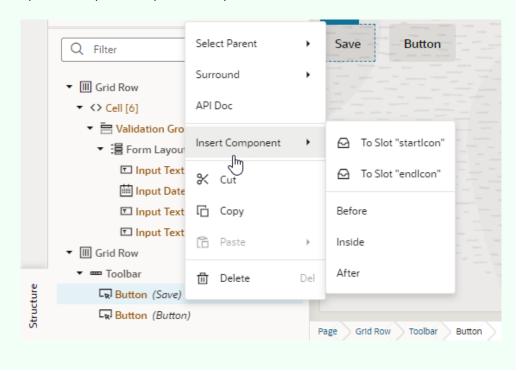


Elements in the Components palette are organized by type. For each type, there are some rules that determine where they can be used on the canvas, as well as the types of pages where they can be placed. An error message will be displayed if you try to place a component on the canvas where the component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



Tip:

You can also add a component by clicking **Insert Component** in the right-click 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 particular slots 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.

Propert ies Tab	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.
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.

Propert ies Tab	Description
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.

Add a Component Using Code Completion

You can edit a page's HTML directly 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 the code editor.
- 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. You can also drag components from the Components palette and drop them directly into valid places in the code editor.

For example, when you start typing <oj- in the editor, the code completion window displays a list of component tags that match the text you type:

```
Code
                                                                Live
                                                                       Design
    <div class="oj-flex">
2
       <h1 class="oj-flex-item oj-sm-12 oj-md-12">Heading</h1>
3
    </div ⋒ <mark>oj</mark>-switch
4

∅ oj-switcher

    <oj ≫ <mark>oj</mark>-tab-bar
6
                                                                        map>

∅ oj-table

∅ oj-tag-cloud

∅ oj-tag-cloud-item

∅ oi-text-area

            oj-thematic-map

∅ oj-thematic-map-area

∅ oj-thematic-map-link

∅ oj-thematic-map-marker
```

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



Tip:

Add a component using the code editor's right-click menu, where you can select to 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** will give you the option of dropping the component you want to insert into a particular slot, as shown here for a button component:

```
<div class="oj-flex">
39
        <div class="oj-flex-item oj-sm-12 oj-md-2">
40
      <oj-button label="Button"></oj-button>
41
42
        </div>

☐ To Slot "startlcon"

      </div>
43
44

☐ To Slot "endlcon"

45
46
                                          Before
                                          Inside
                                          After
```

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 view.

Work With JET Core Pack Components

Starting with VB Studio 24.07, you can build your extension's pages using the next generation of JET components, known as Core Pack components. Core Pack components provide new implementations that improve performance and, in many cases, introduce extra functionality.



Note:

Core Pack components are designed for Redwood, so your extension must use the Redwood theme. This functionality also requires Oracle Cloud Applications 24C or later (with JET 16.x or later).

Some Core Pack components are brand new and available to you in the Components palette under the Early Access category. Some others, typically those that supersede Legacy components, are available in their usual category. For example, you'll find the Avatar Core Pack component (oj-c-avatar) listed under Controls & Navigation, same as the Legacy Avatar component (oj-avatar). Note how Core Pack components use the oj-c- prefix, instead of oi- used by Legacy components.

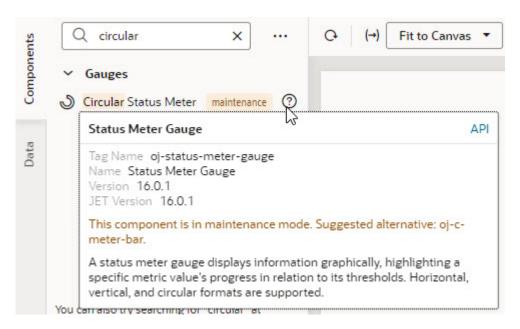
Though some Core Pack components are meant to replace existing Legacy versions, the change won't impact your existing extension because Core Pack components can run side-byside with Legacy components. You can keep a replaced component in your page and continue to update its properties (though dragging and dropping the component afresh uses the Core Pack version by default). It's important, however, to plan time in your development cycle to move to the Core Pack components. Note that no updates are planned for Legacy components; all updates and new functionality will be available only through the Core Pack. For more information, see Core Pack overview in JET documentation.

To move to Core Pack components in your extension's pages:

- 1. Open the Components palette and look for available Core Pack components.
 - To see all Core Pack components and their categories, enter oj-c- in the Filter field. To see Legacy and Core Pack versions for a particular component:
 - a. Click Components Menu and select Show Maintenance.
 - b. Filter for the component. For example, here's how you might see the Avatar versions:

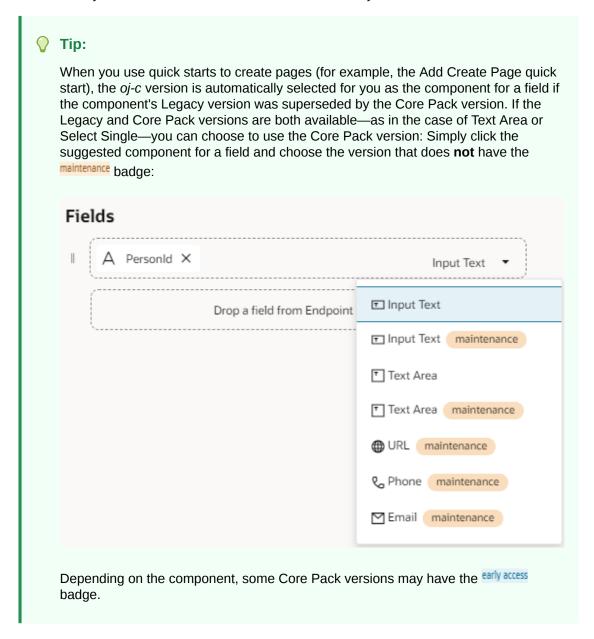


c. Hover your cursor over the component badged maintenance to view its Info icon, then see what the suggested Core Pack alternative is. In some cases, the alternative may be a direct replacement, like oj-c-avatar for oj-avatar. In other cases, it may be an entirely different component. For example, the alternative suggested for Circular Status Meter gauge (oj-status-meter-gauge) is Meter Bar gauge (oj-c-meter-bar):



Update the Legacy component with a suitable Core Pack component.

Preview your extension to make sure it renders correctly.



Bind a Component to Data

To display data in your user interface, you bind the component used in your App UI'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 <code>name</code> variable as the data source for an Input Text component, you should see the Value set to <code>{{\$variables.name}}</code>, which is shorthand for saying your component is now bound to the page-level variable <code>name</code>. The double curly braces around the value <code>{{}}</code> 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 <code>[[]]</code> 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?

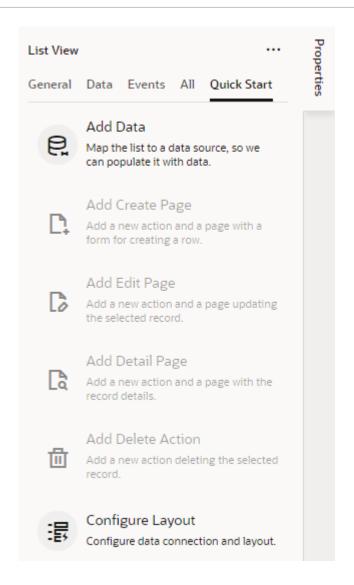
How Do Quick Starts Work?

A *quick start*, as the name suggests, provides a way for you to build the basics of your App UI quickly. It's a wizard that walks you through complex processes, helping you do the tasks required to build functionality in an App UI.

To add functionality for common 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), you need to create the artifacts that perform that function. Depending on how complex the behavior is, adding a functionality might involve creating several variables, types, events, and action chains. 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 need to do when creating the App UI's pages. 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. Typically, you'd select one or more data source endpoints to fetch data and select the fields you want to show in those pages. When your data source is a service with expected endpoints (such as GET, POST, PATCH, or DELETE), VB Studio automatically selects the correct endpoint 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.

You can use quick starts for standard as well as dynamic components. When working with either set of quick starts, keep one key difference in mind: standard component quick starts add pages to your App UI (for example, the Add Edit Page quick start creates a separate page for editing the details of a record); dynamic component quick starts *configure* an existing form or table on the current page.

Add an Image to a Page

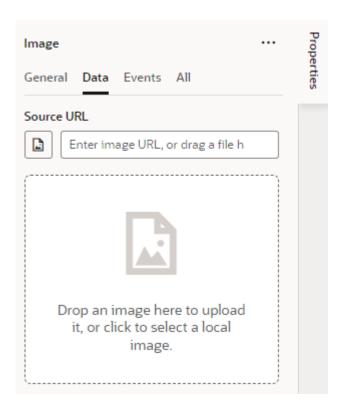
To add an image to a page, you position an image component on the canvas and specify the path to where the image is stored in your App UI's resources.

Images that can be used in your App UI's pages are stored in an images folder, either under the Global Resources folder for use by all App UIs in the extension, or under an particular App UI's Resources folder for exclusive use by that App UI. You can import an image to one of these locations before you actually add the image to a page, or you can import one directly when adding it to your page canvas. In both cases, the image becomes part of your App UI's resources and is available to you through the Image Gallery.

To add an image to a page:

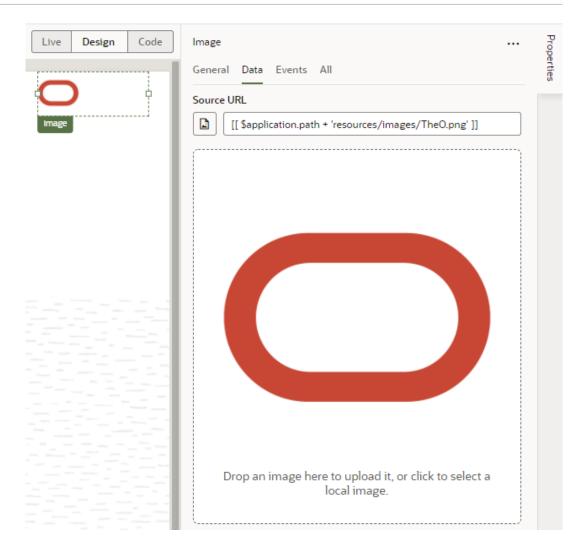
- 1. Open the page in the Page Designer and drag an image component from the Components palette onto the canvas.
- 2. Specify the image's height and width in the **General** tab of the component's Properties pane. Also enter the alt-text for the image.
- 3. Click the **Data** tab in the Properties pane.

The Data tab contains a Source URL field that displays the path to the stored image and can be a string or a variable. This field is empty when an image has not yet been defined.



4. If you want to explicitly use the Image Gallery to select an image that was already imported or add a new one from your local system, click the Image Gallery icon () and work with the Image Gallery. Otherwise, drag your image into the drop target area.





Dragging an image in the drop target area also adds the image to the App UI's <code>images</code> folder and sets the path to the image in the Source URL field, similar to what happens when you use the Image Gallery. The only difference is that the Image Gallery won't open. When you work with the Image Gallery, you can also select (or add) images that are available to all App UIs in the extension.

Note:

To ensure that the relative path to the image is built correctly when the App UI is deployed, the path to the image in the Source URL field must include either <code>\$application.path</code> to refer to images stored in an App UI's resources or <code>\$extension.path</code> to refer to images stored in the extensions resources. You can use the Audits window to help you locate image paths in your App UI that might not be formed correctly.

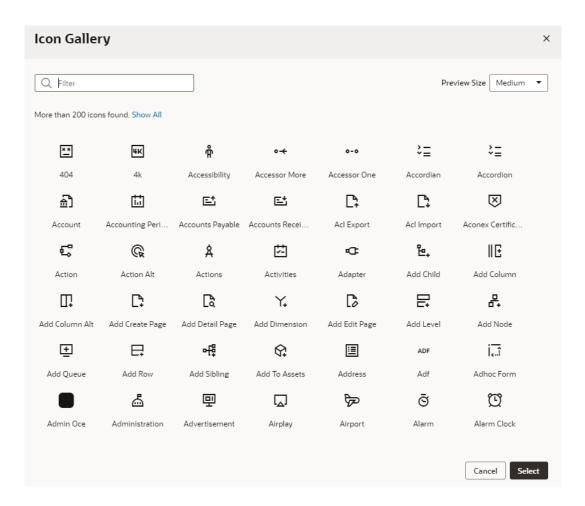
5. Click the **Events** tab to trigger an action chain from a component event. Click the **All** tab to view and edit all of the component's attributes.

Add an Icon to a Page

VB Studio provides an extensive gallery of icons that you can add to your pages using the icon component in the Components palette.

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. Click the image button () under Icon in the **General** tab of the component's Properties pane to bring up the Icon Gallery.
- 3. Select the icon you want to use, then click **Select**.



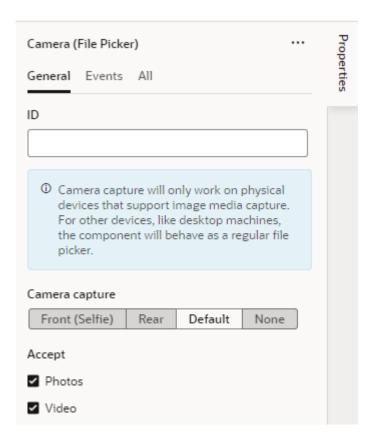
4. Specify properties as needed in the Properties pane. Or click the **Events** tab to trigger an action chain from a component event.

Add a Camera Component to a Page

When you want your App UI to access the camera capabilities of the devices on which it is installed, you can add the camera component to your App UI'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.

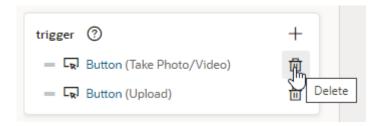


- To provide the user with choices, say, choose from the photo library or take a
 photo or video (instead of directly launching the camera), select None.
- b. Set the Accept property to the file types you want to accept from the device: photos, videos, or both. By default, both Photos and Videos are 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:

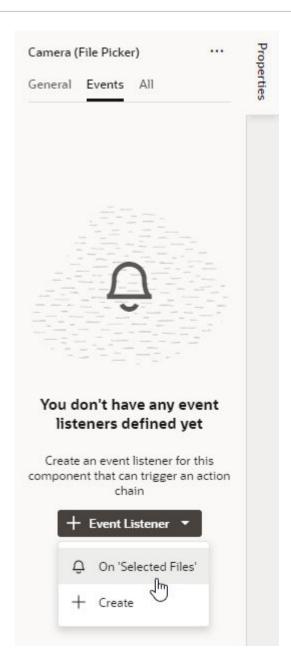
- To add an Upload button that users click to upload files, select 'Upload' Button.
- To add a placeholder for an image that users click to upload images, select **Image**.
- To create a basic drop zone/trigger that can be further customized with other components, select Custom Drop Zone.

Remember to delete the original Take Photo/Video trigger button:



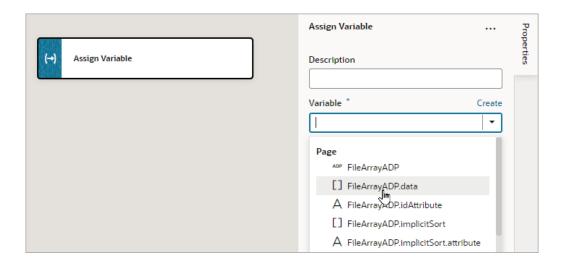
- 3. In the **Events** tab of the Properties pane, configure the camera component's Selected Files event to trigger an action chain when a user selects the camera component. Make sure to select the Camera (File Picker) component, not the button component within the camera component.
 - a. Click + Event 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.

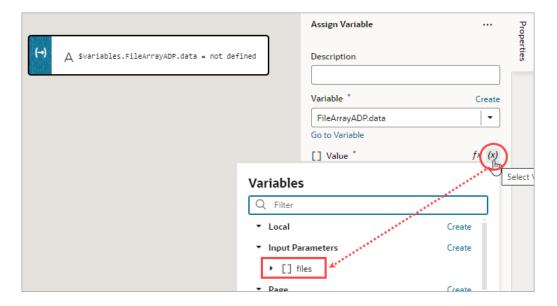
b. Add the **Assign Variable** action on the canvas. For the **Variable** property, select the variable to hold the data from the Camera component (for example, FileArrayADP of type Array Data Provider):



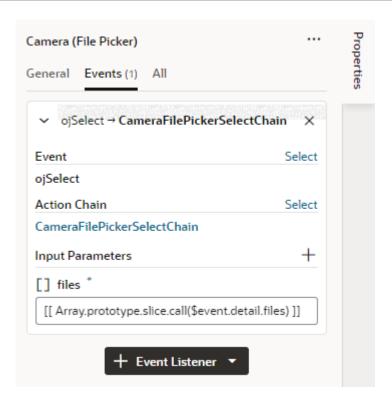
Note:

The items in the array are of type <code>File</code> (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 VB Studio, the files are passed to the action chain as an <code>Array</code> of type <code>Object</code>. 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 (x) 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 a component is bound 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. All of this is done automatically for you when you use the Add Data Quick Start. 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 advanced filtering, you can write JavaScript 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, and value. The filter expression is converted into an appropriate "q" query string when sent to the endpoint. You can create a filter expression using the Assign Variables action, or you can edit the JSON file where the action is defined (for example, main-start-page.json). You can build complex filters by combining multiple expressions. 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.
ор	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:

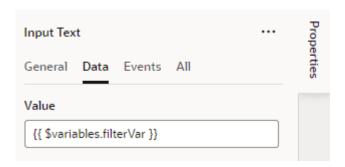
- 1. Create a page variable to hold the user's value for the filter criterion:
 - The filter criterion is a condition that each record must satisfy in order for it to be shown, for example, "If name contains user-value".
 - The user's value is the text that's used to evaluate the filter criterion for 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

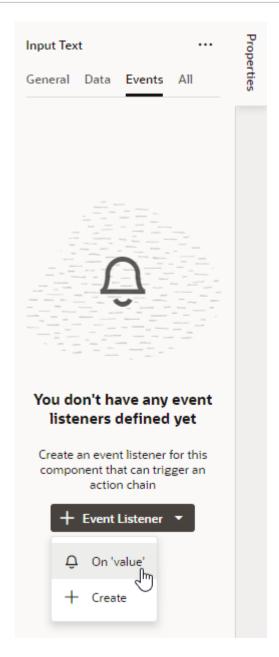


- component to allow users to enter text, or to select a value from a Combobox. In this example, we'll use an Input Text component to enter a value for the filter criterion and to trigger an action chain to filter the displayed data.
- 2. In the Page Designer, add an Input Text component to the page 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':



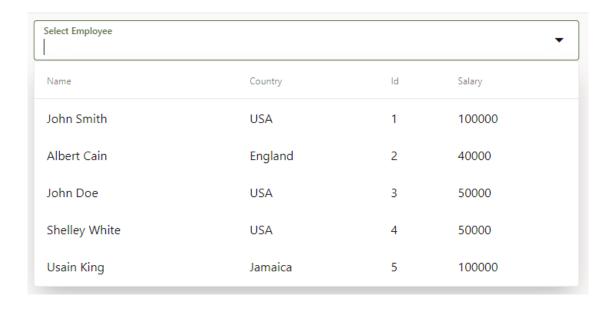


A new action chain is created and associated to the event. You're now taken to the Action Chain editor to implement the action chain for filtering. See Use Filter Builder to Create Filter Criteria for an SDP on how to create the action chain using an Add Variable action.

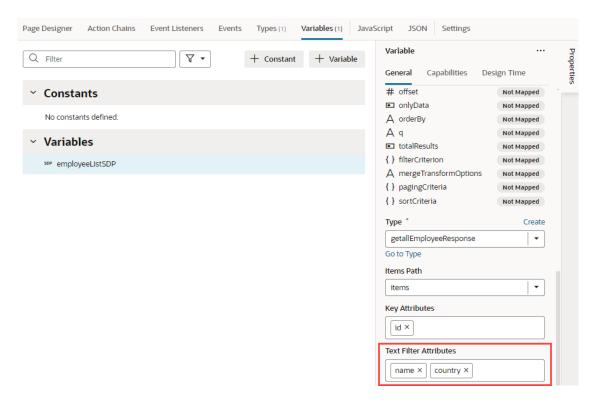
Filter Component Data by Text

When you want user-specified text to filter results shown in a list component like Select Single, you can configure the <code>TextFilterAttributes</code> property on the Service Data Provider (SDP) used to retrieve the list's data. The <code>TextFilterAttributes</code> 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 <code>TextFilterAttributes</code> property on the SDP variable used to bind the component to an endpoint. For example, your Select Single component might use the <code>employeeListSDP</code> variable to request and receive data from the <code>getall_Employees</code> 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 <code>Variables</code> tab. Update the variable's <code>TextFilterAttributes</code> property to specify one or more data fields, for example, <code>name</code> and <code>country</code>:



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, VB

Studio 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:



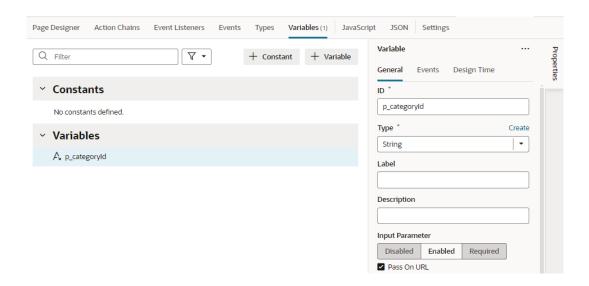
You can also include data fields that don't display to the user—as long as those fields are included in the SDP's definition. For example, if you selected <code>salary</code> and <code>phoneNumber</code> as the text filter attributes, a user who entered 50 would see three results because the text matches 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

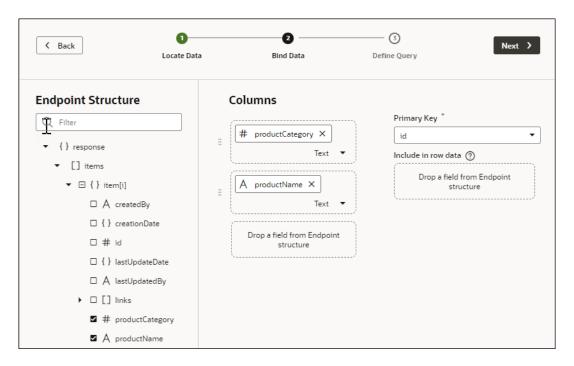
You can specify filter criteria for a collection component, such as a list or table, and append the parameter to the application page URL to apply the filter. This makes it easy to share the page URL with the filter already applied.

To enable the search criteria, set the filterCriterion attribute. In this example, we bind the value of the variable $p_{ategoryId}$ to the $p_{ategoryId}$ to $p_{ategoryId}$

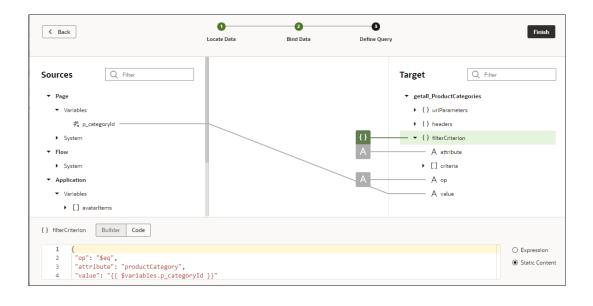
- 1. Click the **Variables** tab for your page.
- 2. Click + Variable, then update the ID as needed and select the Type.
- 3. Set the Input Parameter to **Enabled** or **Required** and select **Pass On URL**.



- 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.

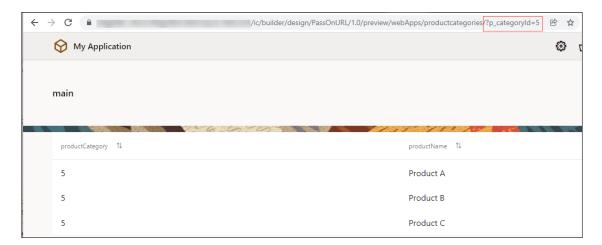


 On the Define Query page of the quick start, expand the filterCriterion builder icon on the Target pane and build your filter by specifying values for attribute, op, and value individually.



9. Click Finish.

To filter the table, add ?paramName=Value to the end of the app URL.



Use Conditions to Show or Hide Components

You can use an oj-if component to conditionally show or hide UI components in your App UI. Use oj-if to surround other components and set conditions to determine whether the components should be displayed on a page.

Note:

The oj-if component provides functionality similar to oj-bind-if, except that where the oj-bind-if element is not removed from the DOM after bindings are applied, oj-if stays in the DOM and can therefore directly be used as the slot content of a custom element. If you want to conditionally render the content of a slot or if you want the element to stay in the DOM, oj-if is recommended over oj-bind-if. To use oj-if in your pages, your extension requires Oracle Cloud Applications 25A (with JET 17.x) or later.

Here's how you can use $\circ j$ -if to conditionally display its contents: Let's say you have a form for users to submit expense reports with fields like Country and Amount (in US Dollars). 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 $\circ j$ -if component (available as If in the Components palette).

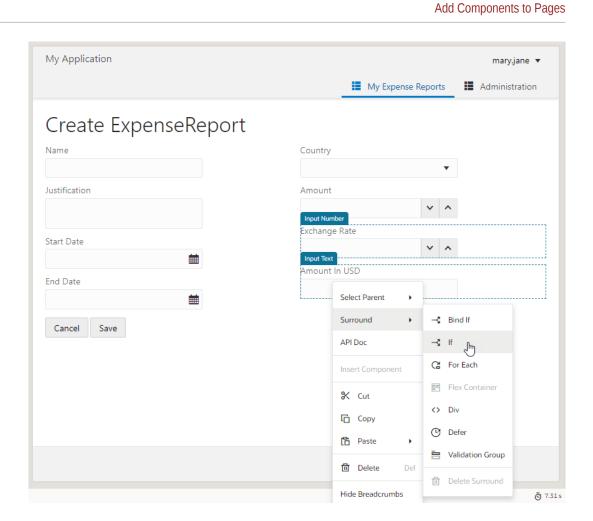
When you add an <code>oj-if</code> component, you also set the conditions under which the component should be displayed by adding an expression in its Test property. For example, you can use an expression that evaluates if the value of a page variable does not equal a predefined value. The surrounded content is displayed if the values are not equal (the expression is true), and hidden if the values are equal (the expression is false).

In our example, we'll build an expression 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.

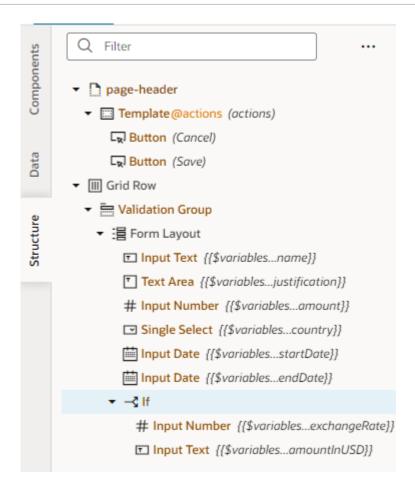
To use an oj-if component to control when a component is displayed in a page:

- In Design view, locate the component that you want to control dynamically, for example, Input Number for the Exchange Rate field and Input Text for the Amount in USD field.
- Right-click the component on the canvas or in the Structure view and select Surround > If from the context menu:

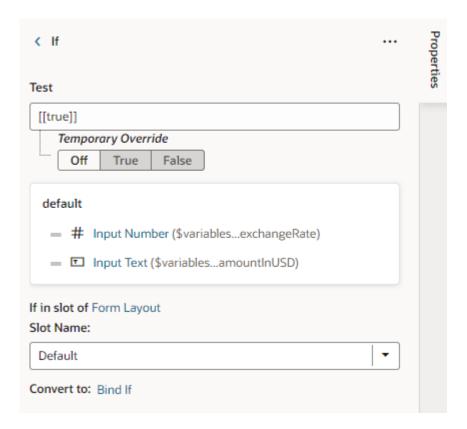




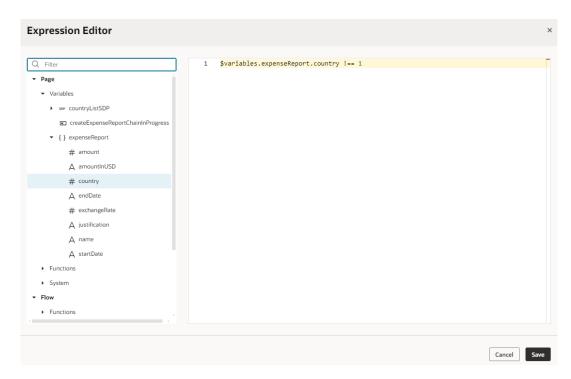
In the Structure view, you'll see the components surrounded by an **If** component. (In Code view, you'll see oj-if.)



3. Select the **If** component in the Structure view and open the Properties pane to 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 <code>country</code> 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:

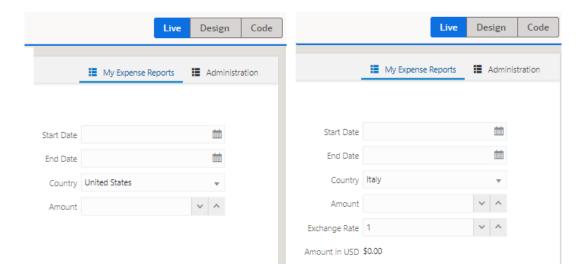
```
[[ $variables.expenseReport.country !== 1 &&
    $variables.expenseReport.country]]
```

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's 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, in the component's context menu on the canvas, or in the Structure view.

6. Test your App UI in Live view. 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.



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.

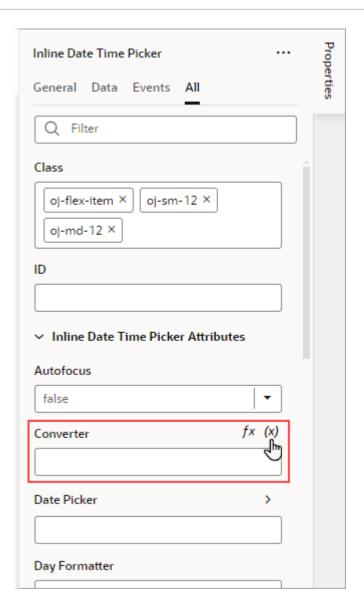
To create your own converter function:

- 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:

```
Page Designer
              Action Chains
                            Event Listeners
                                            Events
                                                    Types
                                                            Variables (1)
                                                                                     JSON
                                                                                            Settings
      define(["ojs/ojconverter-datetime"], function (datetimeConverter) {
 2
        class PageModule {
                                 Include the Oracle Jet IntlDateTimeConverter API
           constructor() {}
 3
 4
           formatDateTimeNewYork(date) {
 5
             return new datetimeConverter.IntlDateTimeConverter({
 6
               "dateFormat": "short",
  7
 8
               "formatType": "datetime",
                                                 Provide parameters to the API
               "timeFormat": "full",
 9
10
               "timeZone": "America/New_York"});
11
12
13
14
         return PageModule;
15
      });
```

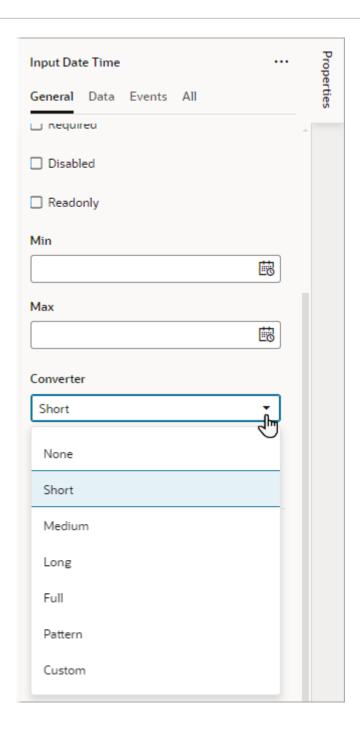
To assign a converter function to an Inline Date Time Picker and Inline Date Picker component:

- 1. In the Designer, click the component, then click the All tab.
- 2. Hover over the **Converter** field and click (x):



To change the date/time format for an Input Date Time, Input Date, or Input Time component:

 In the Designer, click the component, then find Converter in the Properties pane's General tab:



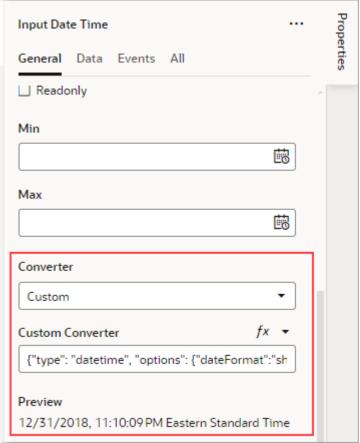
- 2. From the **Converter** list, choose a value:
 - Short, Medium, Long, or Full, if you want a predefined format
 - Pattern, to specify a pattern for the format (like MM/dd/yy hh:mm)
 - Custom, to provide formatting parameters to the built-in converter, or to assign your own converter function to the component.

If you chose anything other than Custom, your work is done.

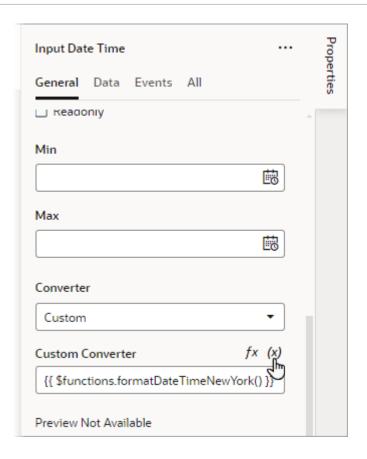
- 3. If you chose Custom, provide the parameters for the built-in converter, OR select your own converter function:
 - To provide the parameters for the built-in converter, enter them in the Custom Converter field. These parameters use the Oracle JET IntlDateTimeConverter API.



For details about the parameters that can be provided to the API, see Final Class:



To select your own converter function, click the (x) symbol that appears beside the **Custom Converter** field label:



Add Dynamic Components to Pages

Using **dynamic components**, such as a table, form, or container, in your App UI's pages allows you to dynamically control what's displayed in the page. 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 has a manager role. 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-if) both use conditions to determine what's displayed on a page. While you can use oj-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.



There are two ways to specify display logic: *rule sets* and *business rules*. The main difference between them is that whereas a rule set operates on a single dynamic table or form, business rules allow you to impact a field no matter which dynamic component (or which page) it appears on. In most cases you'll use a rule set for a dynamic component first, then optionally use business rules to control the display at a field level.

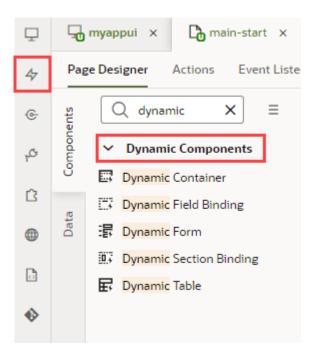
You have two main objectives when creating pages using dynamic components: one, to configure the component's content the way you want it using *layouts* and *templates*, and two, to define the display logic that determines the layout and templates displayed in the component. In most cases you define the logic first, then configure the content that will be used in your logic.

There are three key types of dynamic components that can be used in your App UI's pages: *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 are listed in the Components palette under Dynamic Components. You can enter dynamic in the palette's filter field to help you locate them.



You also have the option of creating a layout from scratch on the **Layouts** tab in the Navigator, where you 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 forms or tables, you'll need to have your component's data source (a service connection that receives data from REST APIs) ready before you can work with layouts.

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:	
Add a dynamic form or table to a page	Add a Dynamic Table to a PageAdd a Dynamic Form to a Page	
2. Configure the rule set's display logic and layouts.	 Determine What's Displayed at Runtime With Rule Sets Create a Layout for a Dynamic Table or Form 	

For each dynamic form or table, you usually have a default rule and an accompanying default layout. This default set is created for you when you configure the component on a page using a Quick Start (or, when you create a rule set in the Layouts' **Rule Sets** tab). With this default set in place, you can 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.

A Layout's tab contains editor tabs for editing and configuring dynamic forms and tables:



Add a Dynamic Table to a Page

Add a dynamic table to your App UI'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.

Before you begin, make sure you've defined a service connection that lets you access external REST services, so your App UI can retrieve and send data to and from its REST endpoints. See Work With Services.

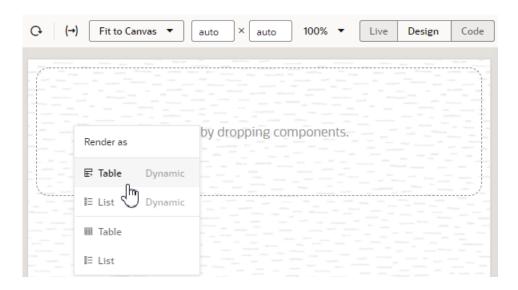
To add a dynamic table component to a page:

1. With your page open in the Page Designer, drag the Dynamic Table from the Components palette onto your canvas.





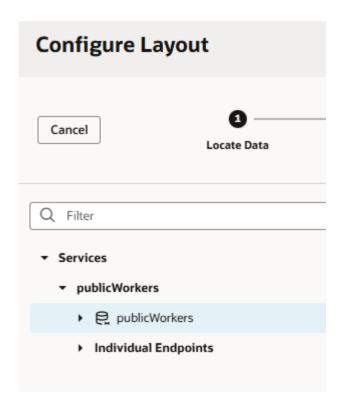
You can also drag the Get Many endpoint for the data source you want to bind to the table from the Data palette onto the canvas. After dropping the endpoint on the canvas, select **Table Dynamic** in the Render as menu:



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.
- In the wizard's Locate Data page, select the data source to bind to the dynamic component. Click Next.

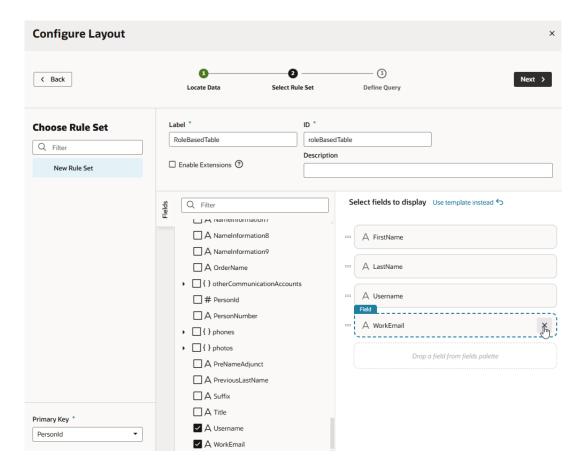
This example uses a service connection to a Workers service connection as the data source:



4. In the Select Rule Set page, select New Rule Set (if necessary) and enter a label and ID (and optionally, a description) for the rule set.

Let's say you want to show one layout with some employee fields (for example, the Send Credentials Email Flag) only when the user is an IT manager, and another layout (without the Send Credentials Email Flag 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 check box 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:

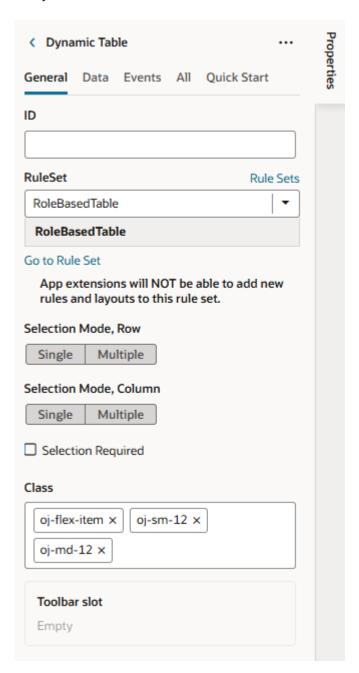


Remember that because this is the initial default layout, the SendCredentailsEmailFlag isn't selected.

Select Enable Extensions if you want to allow other extensions to extend the rule set. Click Next.

If you select Enable Extensions, you'll need to provide a description of the rule set. You can easily change this setting later, if needed.

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 <code>Employee</code> role. You could then add another rule to show a different layout when the user has the <code>Manager</code> role. See Determine What's Displayed at Runtime With Rule Sets.

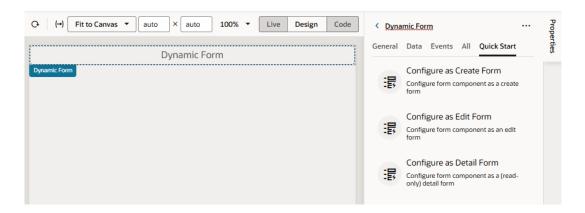
Add a Dynamic Form to a Page

Add a dynamic form component to your App UI'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.

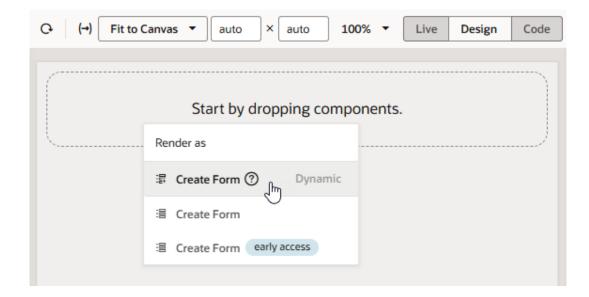
Before you begin, make sure the service connection you want to use is available. The service connection lets you access external REST services, so your App UI can retrieve and send data to and from its REST endpoints. The service connection can be one defined in one of your extension's dependencies, or it might be one you've defined in your extension. See Work With Services.

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.

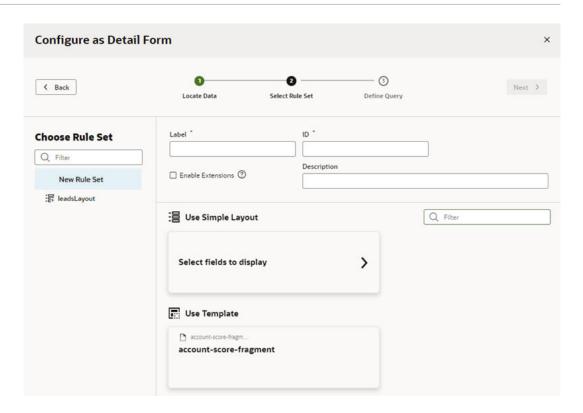
If you plan to use the **Configure as Edit Form** and **Configure as Detail Form** quick starts, you'll be prompted to provide endpoint parameters to be able to fetch and update the data of a particular row in the form. Make sure you create variables that you can map to these parameters before you use 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 created 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 **Enable Extensions** if you want to allow other extensions to extend the rule set.
 - If you select Enable Extensions, you'll need to provide a description of the rule set.
- 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.

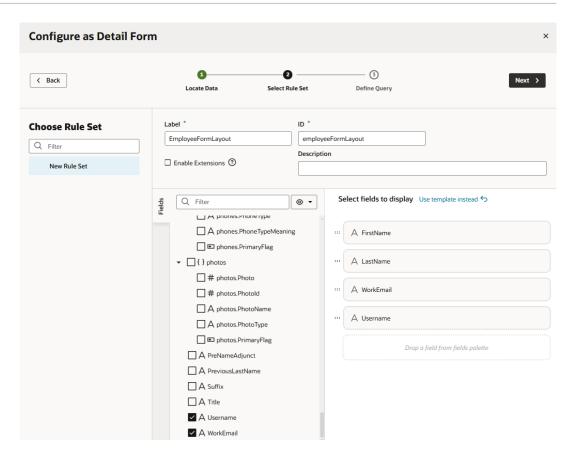




If you select a template and it isn't right for this form, you can return to this pane and choose a different template, or click **Select fields to display** to create a layout without a template.

7. Select the fields to display in the form.

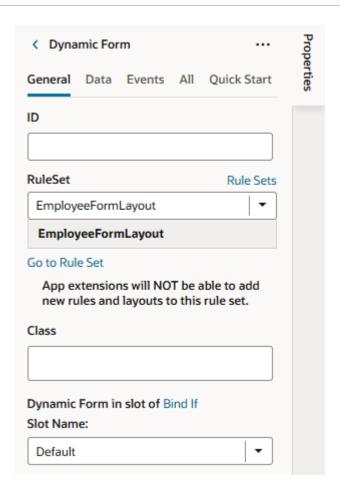
This example shows the **Configure as Detail Form** quick start wizard, with fields selected from an Employees service connection. These fields are added to the form's default layout. The form is also configured to fetch the data of a particular row in the form.



Click Next when you are done.

8. In the Define Query page, 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 users with a specific user role. See Determine What's Displayed at Runtime With Rule Sets.

Determine What's Displayed at Runtime With Rule Sets

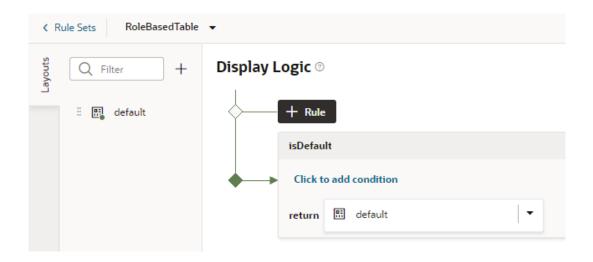
You can control what's displayed in dynamic forms and tables by using display logic and layouts, which you configure on the Layouts' **Rule Sets** tab.

Suppose you want to show employee data (say, whether an employee's credentials can be sent by email) 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 IT Manager role, the page shows the layout that includes the Send Credentials Email Flag 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 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 (available when working with dynamic forms) 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 \numberValue ($\fields.ConflictId.value()$). 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 \numberValue is limited by the maximum precision allowed by the Number type in Javascript.

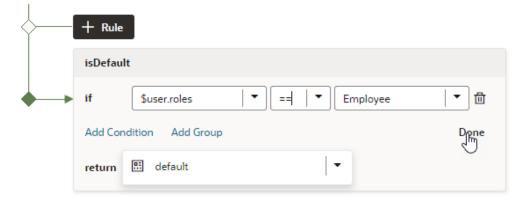
- \$responsive variables determined by the screen size of the device the app is currently displayed on. For example, the responsive.mdUp variable's value is True if the current user is using a device where the screen width is 768 pixels or more, such as a tablet. Look for these variables under the **Responsive** 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, start entering the role name you need in the Value field. The Value drop-down will display up to 25 matching Oracle Cloud Applications job and abstract roles; keep typing until the desired role displays. (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.)

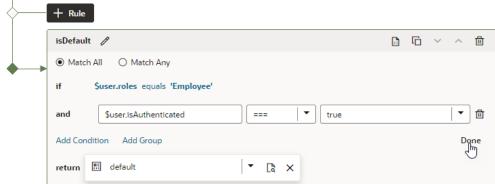
Here's the condition to check whether the current user has the Employee role:

Display Logic ®



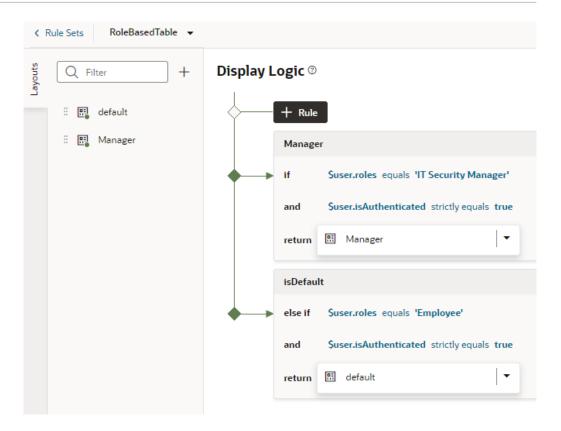
c. You can add more conditions and group conditions if you want to use more attributes to make the rule more precise, for example, you can make sure the user has the Employee role AND is authenticated. You would then create a rule with two conditions, and select **Match All** to require that both conditions be true.

Display Logic [®]



- d. Click Done.
- 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's selected by default in the return field. You can use the same layout with multiple rules.
- Create more rules as required, for example, to display a Manager layout only to authenticated users who have the IT Manager role:
 - a. Click the Duplicate icon (¹), 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 check box 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 IT Manager and extend it to show only to authenticated users:

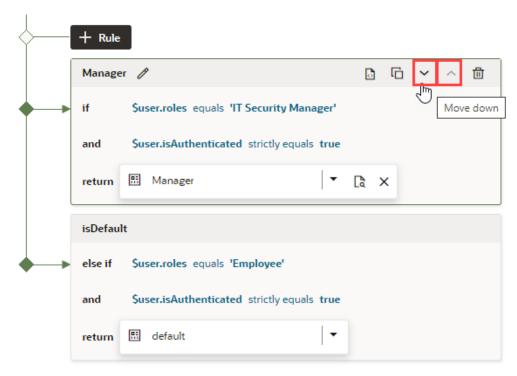


If you click $^{\odot}$, you can see and edit the rule's expression. For the rule above, you'd see the following expression:

```
$user.roles == 'IT_SECURITY_MANAGER_JOB' && $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.

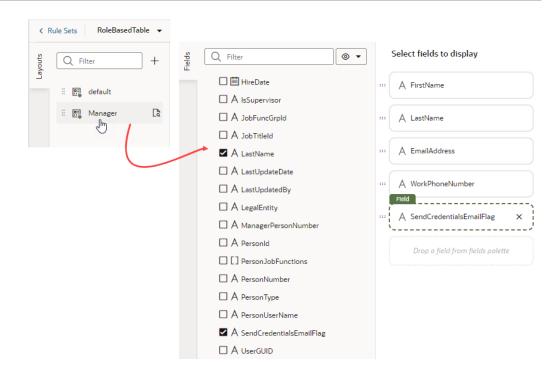
Display Logic ®



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 SendCredentialsEmailFlag):



Create a Layout for a Dynamic Table or Form

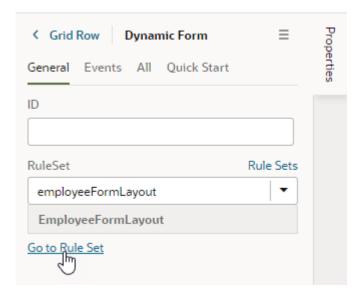
A *layout* defines the fields that are displayed in a dynamic component at runtime. You create and configure the layouts for a component's rule set in the Layouts' **Rule Sets** tab.

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.

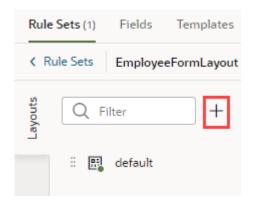
The fields you can display in a layout are determined by the fields available in the artifact's data source, say, a business object that has five fields. You can choose which of these five fields that you want to display in the dynamic component—and the order in which they should appear—but you can't 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:

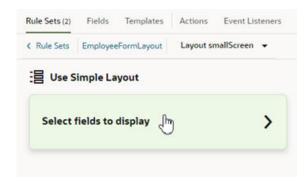


3. Click † in the rule set's Layouts pane, then enter a name for the new layout.



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:

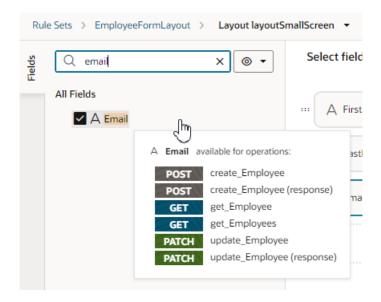
- A SmallScreen layout configured to show only Name and Email
- A MediumScreen layout configured to show Name, Department, and Email
- 4. Click the new layout name, then click Select fields to display to open the layout editor.



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**.

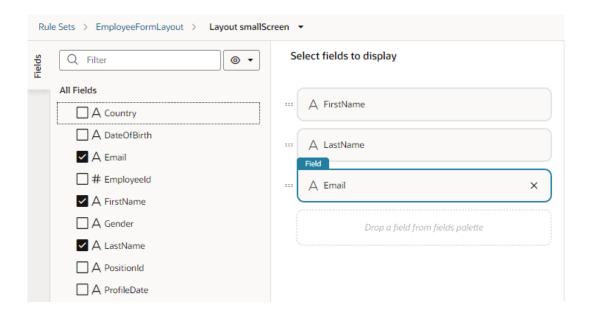
5. Add fields from the Fields palette to the layout.

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. You can hover your cursor over a field to see the available operations:

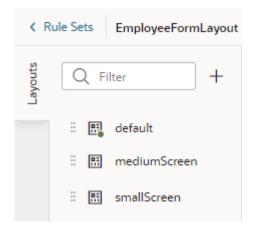


To add a field or object, select its check box in the Fields palette or drag it from the palette onto the drop target area on the right. The columns appear in the order selected; use the handles to the left of each field if you want to re-order them. You can also remove a field by clicking its Delete icon.

For example, here's what our SmallScreen layout might look like:



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.

How To 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 <code>\$fields.USMType_c?.value()</code>, JavaScript will check to make sure that <code>\$fields.USMType_c</code> is not null or undefined before trying to access <code>\$fields.USMType_c?.value()</code>. If <code>\$fields.USMType_c</code> 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.

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

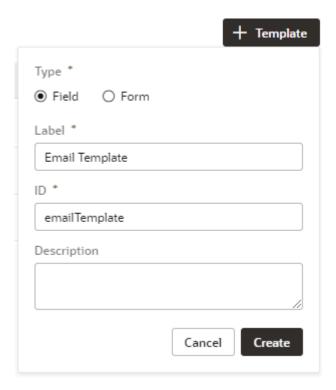
You can customize how a field is rendered in a layout for a dynamic form or table by applying a *field template*. A field template contains UI components, for example, text fields or images, and defines their properties, such as styling details. Components in a template can access the variables, constants, 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.

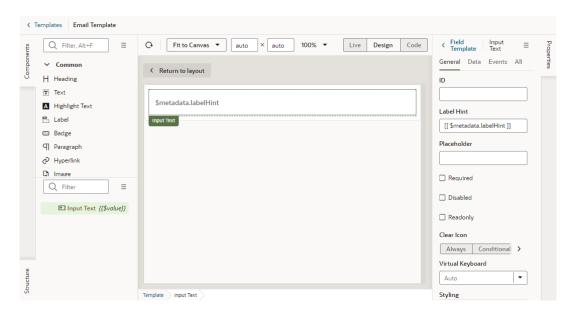
To create a field template for a field in a dynamic form or table:

- Open the layout's **Templates** tab.

 The Templates tab displays a list of field and form templates that are
 - The Templates tab displays a list of field and form templates that are already defined for the artifact.
- Click + Template. Select Field, specify the Label (the ID is generated for you), and click Create.



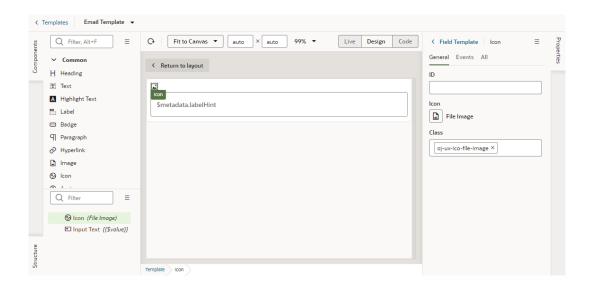
The new field template opens in the template editor, which contains a Components palette, Structure view, canvas, and a Properties pane. In the Structure view, you'll see that your new field template includes an automatically generated Input Text component. This is used to display the data and display 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:

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.

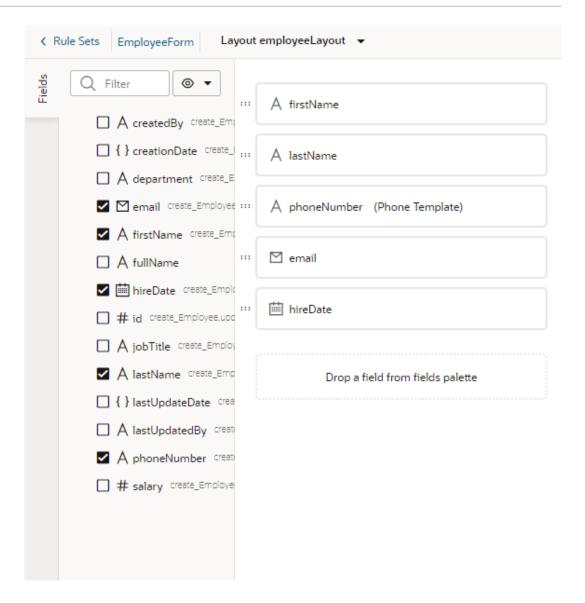
Apply a Template to a Field

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.

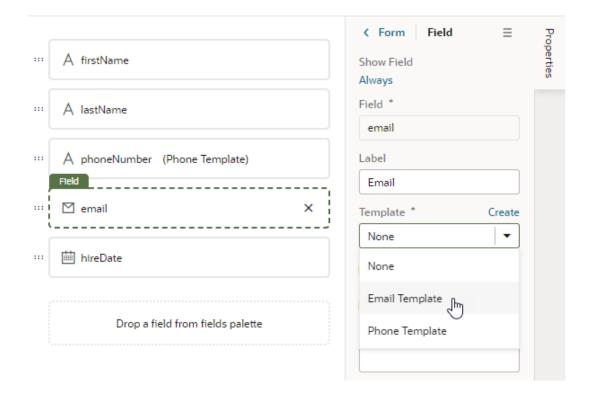
To apply a template to a field in a layout:

1. In the rule set editor, click the name of the layout name you want to work on.

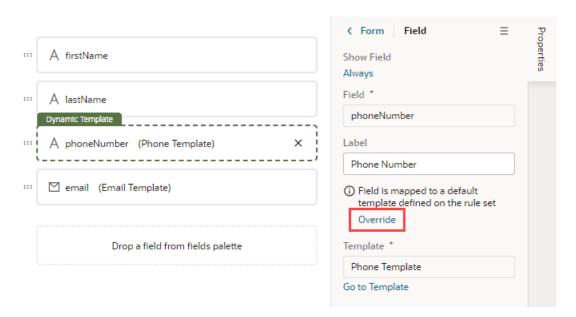
The center pane of the layout editor lists the fields that will be displayed in the layout and any templates that are applied to them (as shown in the image for the phoneNumber field). If you duplicated an existing layout, your new layout might already list some fields, or have templates already applied to fields.



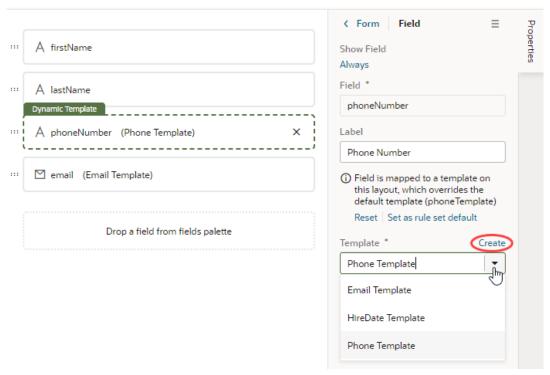
- 2. Select the field you want to apply a template to.
- 3. In the field's Properties pane, select a template from the Template drop-down list.
 If no template has been applied to the field, you can select a template in the list (as in this image):



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.

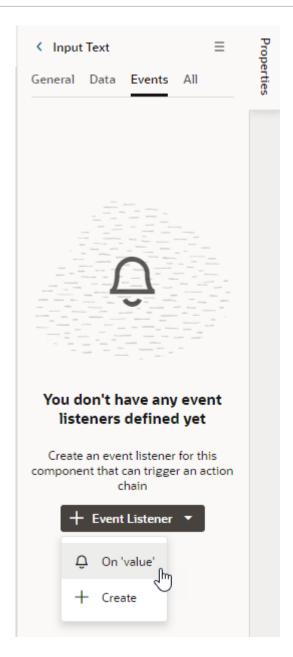
For example, you might want to display some additional details or options when someone changes the value in one of your form fields. You can add an event that's triggered when the value changes, and start an action chain that retrieves the data and displays it in your page. The Quick Start option in 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:

- 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.



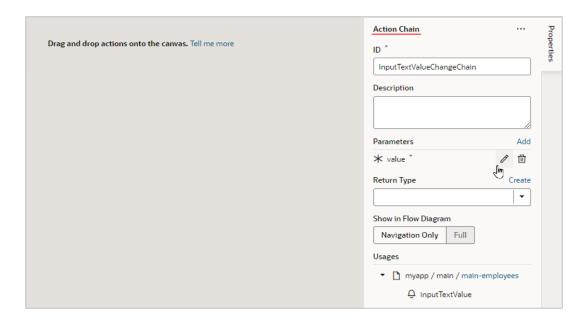


The On 'value' event is a quick start that suggests an event for your component. In the case of a text field, the suggested event is value, which is triggered when the text field's value changes, for example, when someone types in the field. If you don't want to use the suggested event, you can select **New Custom Event** 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:

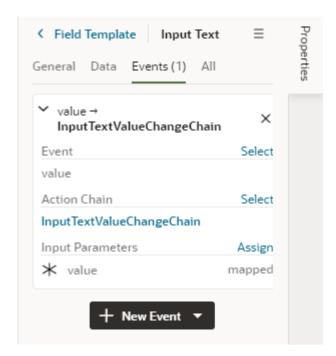
- an event is defined for the text field
- a new action chain is created
- an event listener is created that will trigger the new action chain when the event occurs
- you are navigated to the new action chain in the Action Chain editor.

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:

Open the layout's **Templates** tab.

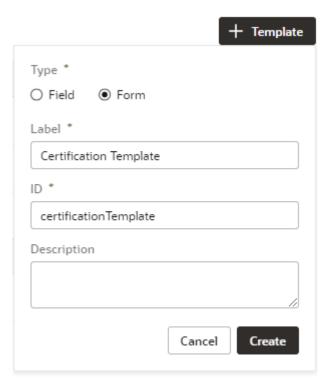
The Templates tab displays a list of field and form templates that are already defined for the artifact.



Tip:

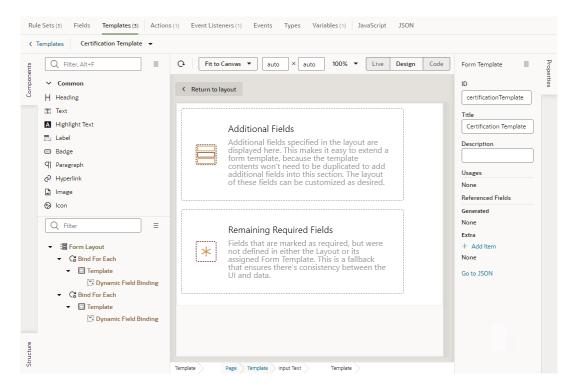
If you've already created a form layout and want to create a template for it, you can open the layout in the rule set editor, click Use Template, and then select Create a New Template in the Use Layout Template window.

Click + Template. Select Form, specify the Label (the ID is generated for you), and click Create.



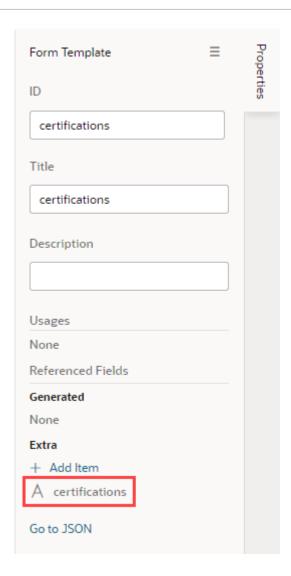
The form template opens in the template editor, which contains a Components palette, Structure view, canvas, and a Properties pane.

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.

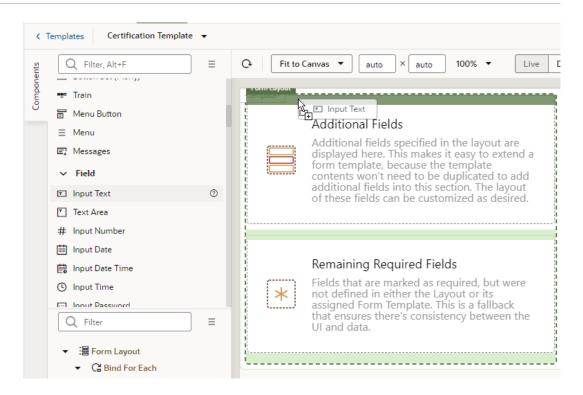


3. In the Form Template's Properties pane, click + Add item under Extra and select a field. Extra fields are defined in the template, not in the layout. You will want to add a field as an Extra field when you know it will be needed by the layout. Each field you add in the Extra section can be used in the form and will always be available when the template is applied.

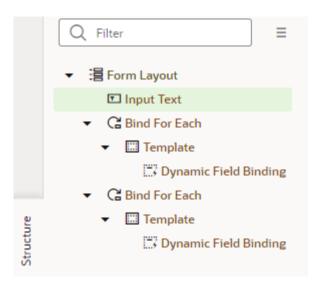
Each Extra field must be mapped to a component if you want it to appear in the form. This image shows the Properties pane after the <code>certifications</code> field has been added to the template as an Extra.



4. Drag the component you want to add from the Components palette and position it in the Structure view or on the canvas.

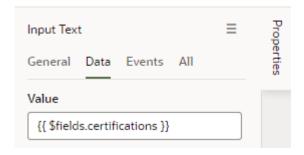


You can add components above and below the read-only template fields, but not within them. In the Structure view of this template, you can see an Input Text component that was positioned above the Additional Fields template in the Form Layout.



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 f^x to open the Expression Editor, or f^x to open the Variables picker.

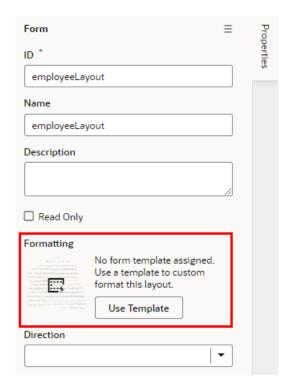


After you've added the components and fields to your form template, you can apply the template when you edit a layout in the Rule Sets editor.

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.

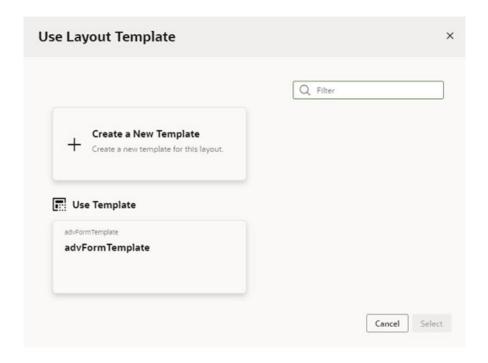


If a template has already been applied to the form and you want to switch to a different one (or remove it), click **Select** in the Properties pane.

3. Select the template you want to apply in the Use Layout Template window. Click **Select**.

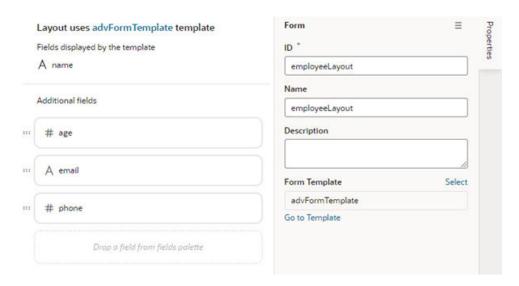
The Use Layout Template window lists the available templates you can apply to your form layout.





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).

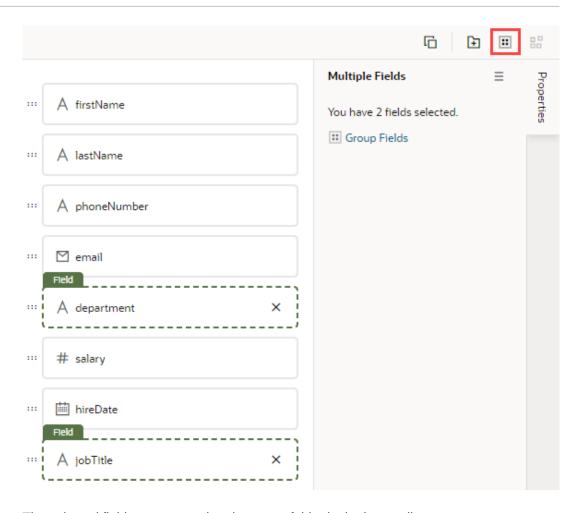
To group fields in a dynamic form layout:

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.

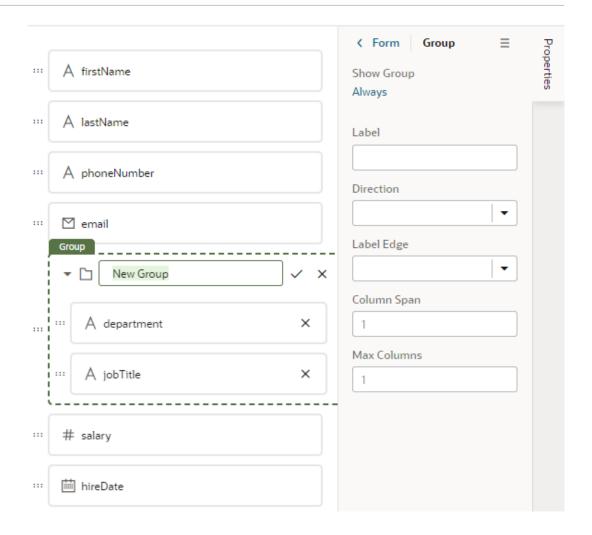
To do this, select a layout in the Navigator's **Layouts** tab, then find the dynamic form in the **Rule Sets** tab (as shown here); click the form on a rendered page in the canvas area; or select the form from the Properties pane.

- 2. Open the form you want to edit.
 - Use the icon in the toolbar if you want to duplicate the current layout.
- 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 III 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 \checkmark 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.

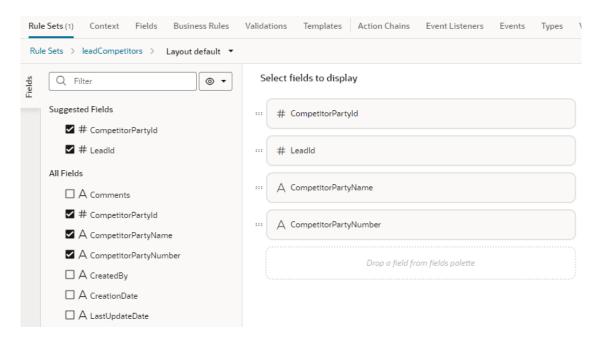
Compared to layouts in rule sets, business rules don't provide the fine grain control over all the details of a form or table layout, but they do allow you to quickly create rules that can override some field properties. Like rule sets, the overrides in business rules are applied at runtime, but

they take precedence over field properties defined in rule set layouts. For more details, see What Are Business Rules?

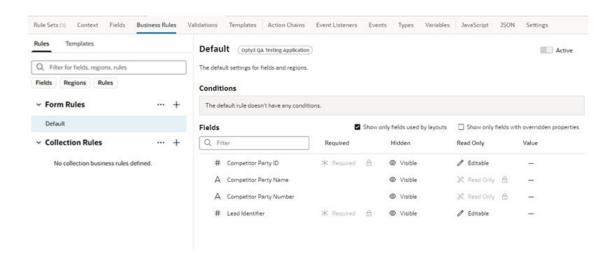


Like rule sets, business rules cannot override properties set in the Layout's Fields tab.

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:

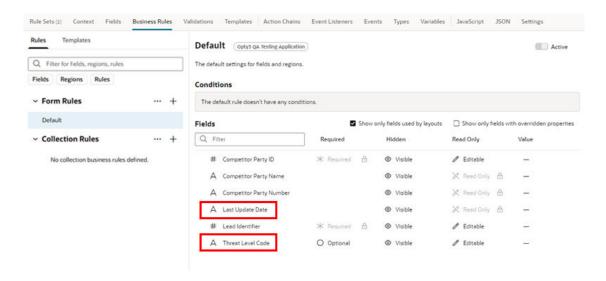


When you switch to the Business Rules tab for leadCompetitors, the four fields in the default layout are displayed in the list. The "Show only fields used by layouts" filter is selected, so only the four fields are shown:





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:



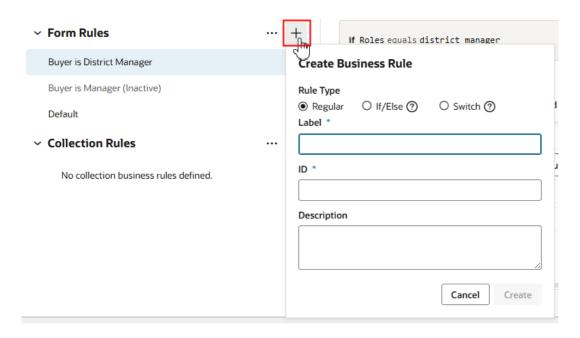
You could now write one business rule that would affect both forms, for example, to hide a field, even though the forms use different rule sets.

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.
- In the business rules editor, click + next to Form Rules to open the Create Business Rule dialog box.



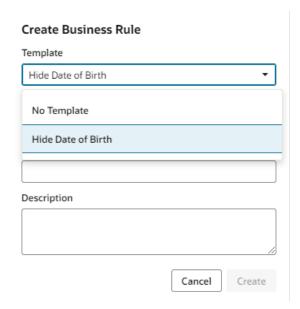
Rather than creating a rule from scratch, you can duplicate an existing rule and use it as the basis for a new rule, or create a rule from a business rule template. To duplicate a rule, right-click the rule in the list, and then click **Duplicate**.

Note:

When duplicating rules defined in extension dependencies (the rules listed under Built-in Rules):

- You cannot duplicate a rule if it uses advanced expressions, for example, in the rule's condition or to override a field's value.
- You cannot duplicate a rule if it uses any global functions that cannot be referenced in the extension. If the global functions are referenceable, when the rule is duplicated they are automatically imported, and the references to the function are updated in the rule.
- When duplicating nested rules, you can only duplicate the top-level rule. When you duplicate the rule, the child rules are also duplicated.

If there are any business rule templates defined in the Layout, you can select one using the Template dropdown list in the Create Business Rule dialog box:



3. Select the Rule Type in the dialog box.

Select **Regular** to create a standard business rule. The other rule types (If/else and Switch) are special types of nested rules. For more about these nested rules, 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.

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 "(Inactive)" is appended to the name, like this:



Inactive rules are not included in the rule evaluation process.



You can deactivate all the form rules or collection rules at once by clicking the three dots next to the Form Rules or Collection Rules headings, 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 Properties For Form Fields

For each form field, you can set some *properties* to override the values set by lower-level rules, including the built-in rules (as long as they are not locked).

You can set these properties for form fields:

- Required The options are Required or Optional
- Hidden The options are Visible or hidden
- Read Only The options are Editable or Read Only
- Value The value can be a static value or an expression

Property overrides affect the form only when the rule's conditions are met for the currently logged-in user.

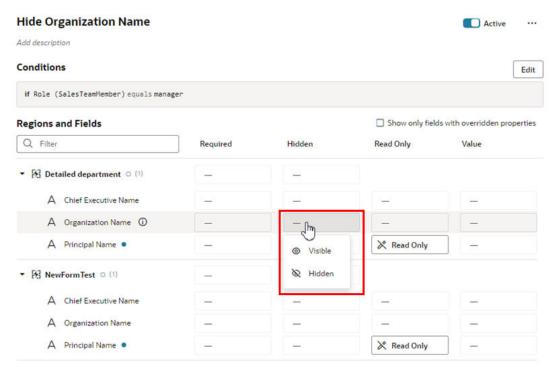
If more than one rule impacts a given field, it can be tricky to sort out what is finally displayed at runtime to each user group; Understand What Will Be Shown at Runtime can help you.

To set a property for a field:

- 1. Locate the field you want to modify.
- 2. 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:



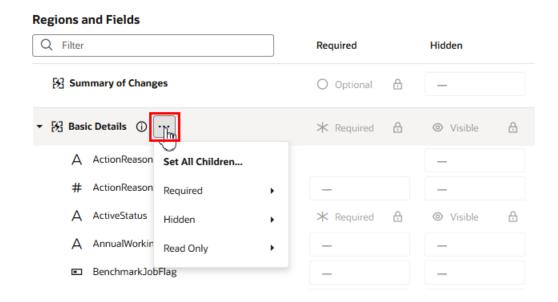


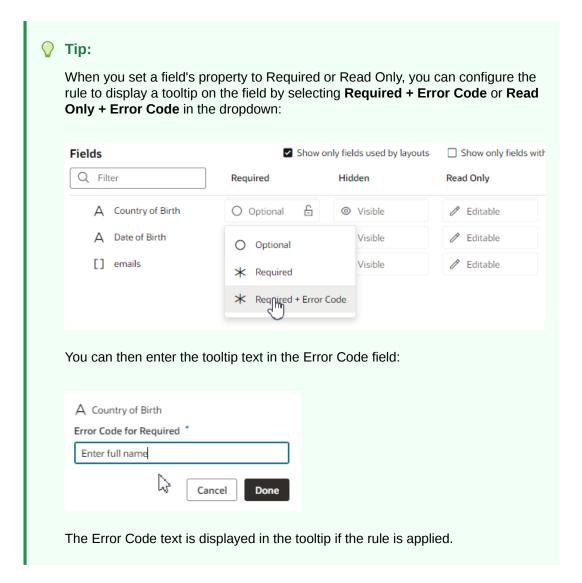
VB Studio automatically saves your work for you, so there's no need to do so explicitly.

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.

If you see child fields when you expand a region or an object field, you can use the Set All Children popup menu to quickly override the properties of all the child fields. For example, you can use the popup menu to set the Hidden property for all the child fields in a region to Hidden instead of setting the property for each field individually. To override a property, click ... next to the region to open the menu, and then select the property (Required, Hidden, or Read Only) and the override you want to apply:





Remember that the Default rule is always active, which establishes the out-of-the-box behavior. All other built-in and extension rules 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.



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 Studio 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.

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.

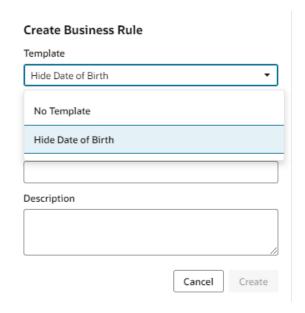
Rather than creating a rule from scratch, you can duplicate an existing rule and use it as the basis for a new rule, or create a rule from a business rule template. To duplicate a rule, right-click the rule in the list, and then click **Duplicate**.



When duplicating rules defined in extension dependencies (the rules listed under Built-in Rules):

- You cannot duplicate a rule if it uses advanced expressions, for example, in the rule's condition or to override a field's value.
- You cannot duplicate a rule if it uses any global functions that cannot be referenced in the extension. If the global functions are referenceable, when the rule is duplicated they are automatically imported, and the references to the function are updated in the rule.
- When duplicating nested rules, you can only duplicate the top-level rule. When you duplicate the rule, the child rules are also duplicated.

If there are any business rule templates defined in the Layout, you can select one using the Template dropdown list in the Create Business Rule dialog box:





4. Select the Rule Type in the dialog box.

Select **Regular** to create a standard business rule. The other rule types (If/else and Switch) are special types of nested rules. For more about these nested rules, 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.

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.

- 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.
- 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:

The Hidden property has three options:

- Hidden by Default hides the table column when the page is rendered. A user can still
 use the table's popup menu to hide and show the column.
- **Visible by Default** displays the table column when the page is rendered. A user can still use the table's popup menu to hide and show the column.
- **Always Hidden** hides the table column. When this option is selected, a user cannot use the table's popup menu to display the column.

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.



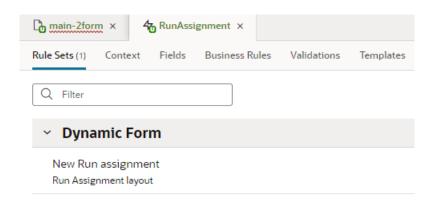
You can deactivate all the rules at once by clicking the three dots next to the Built-in Rules heading, then clicking **Deactivate All**. This can be useful when debugging a page, allowing you to see the page with no rules applied. Use **Activate All** to reinstate all the rules at once, or use the **Active** toggle to selectively activate them as you work through your debugging process.

Include and Exclude Rule Sets in Business Rules

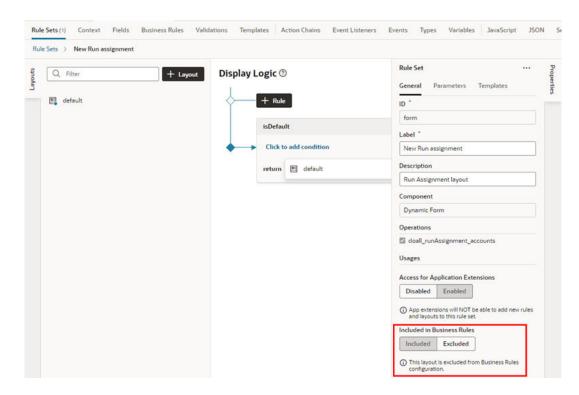
When you create a dynamic form or table, you can choose if you want to allow it to be governed by business rules. Rule sets are included in business rules by default, so you need to explicitly exclude a rule set if you don't want business rules to affect the component.

To exclude a rule set from business rules:

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.



Select Excluded in the rule set's Properties pane if you don't want it included in business rules

Create Templates for Business Rules

You can create business rule and validation templates to help users when they create new business rules. By providing templates, a user can create a rule from a template and modify it as needed, instead of starting with an empty rule.

A template can provide examples of how to configure business and validation rules, for example, how to specify rule conditions, how to write advanced expressions, or examples of how to override properties. Templates themselves are not evaluated.

Templates are defined in the Layout. If you create a new Layout in your extension, you can define business rule templates in the Layout (in metada-rules.json) using the business rules editor. You cannot define business rule templates if you're extending a Layout in a dependency.

Let's say you add a new service connection, and then add a dynamic form that is bound to a business object in that service. A new Layout is created representing the business object. If you plan to use business rules with the component, you can define a business rule template that defines a condition and some property overrides, for example, hiding some fields when the user is in the UK. Anyone extending the Layout can then choose to use the template when creating their own business rules. After creating a rule from the template, they could then edit the rule, for example, so that the fields are hidden for users in Canada instead of the UK.

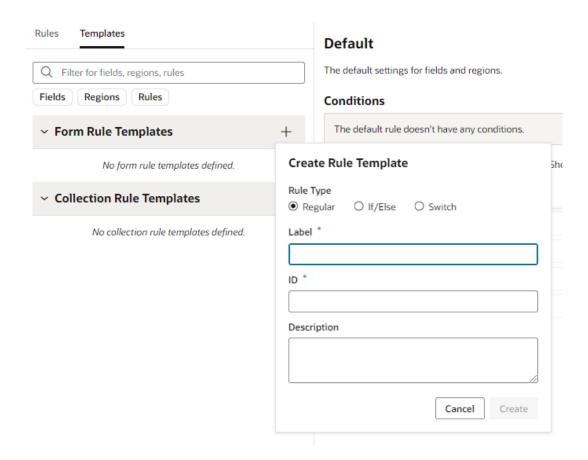
- In the Layout, open the Business Rules tab.
 If you want to create a validation rule, open the Validations tab in the Layout.
- **2.** Open the **Templates** tab in the rules editor.

The editor contains a Templates tab where you can create, view and edit templates. If you don't see the Templates tab, you can't create templates.

The Templates tab is available if you're working in a "base" dynamic layout, in other words, a Layout which you've created. The Templates tab is not available if you're extending a Layout in a dependency.

3. In the Templates tab, click to open the dialog box for creating a template.

For example, when creating a business rule template, click the + next to Form Rule Templates or Collection Rule Templates to open the Create Rule Template dialog box:



4. Select the Rule Type.

Select **Regular** to create a standard business rule. The other rule types (If/else and Switch) are special types of nested rules. For more about these nested rules, 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.

The template is added to the list of templates.

7. Create the template conditions, and define the template.

In a business rule template you can define examples of business rule conditions and property overrides.

In a validation template you can define examples of messages, usually error messages or warnings, that should be displayed in the page when the conditions are met.

When a user creates a business rule, if the Layout defines a business rule or validation template, the dialog box for creating the rule contains a Template dropdown list. The user can choose to create an empty rule, or a rule based on the template, and then modify the rule as needed.

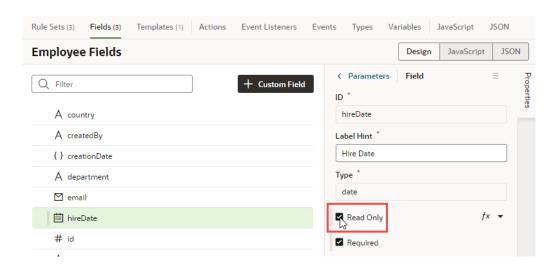
Edit a Field's Properties

When you edit a field's properties in a rule set 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 Layout's **Fields** tab.

Set a Field to be Read Only

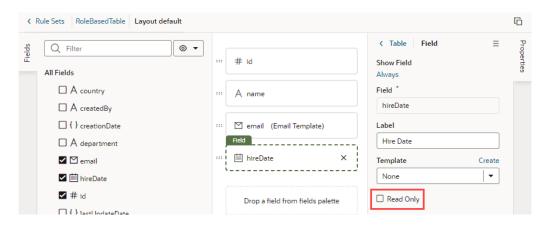
You can set a field to be Read Only when you don't want its value to be changed by everybody. For example, you might want only managers to be able to change an employee's hire date. You would then set the field to be Read Only in all rule set layouts, except the one seen by managers.

- 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.

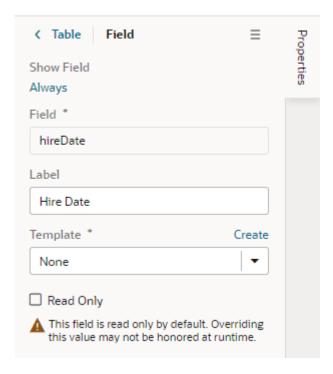


- To edit a field's Read Only property in a particular layout:
 - In the Rule Set editor, open the layout and select the field in the center pane.
 - 2. Edit the field's Read Only property in the Properties pane.





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.

To override a field's Read Only property using business rules:

The rule set must be included in business rules. You can only view a rule set's fields and create extension rules if the rule set is included in business rules.

- 1. In the Layout's Business Rules tab, create a form rule.
 - See Create a Rule For Forms.
- 2. Specify the condition for the rule.
 - See Set Conditions for a Rule.
- 3. In the list of fields, locate the field you want to make read-only.

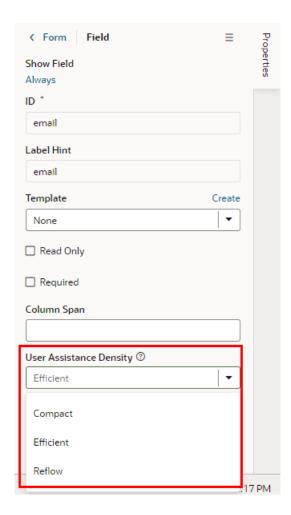
Click the dash—or an existing value—in the Read Only column, and select Read Only in the dropdown list.

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:

- 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.

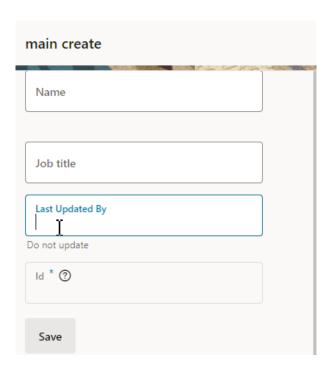


You can choose from three options:

- 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:

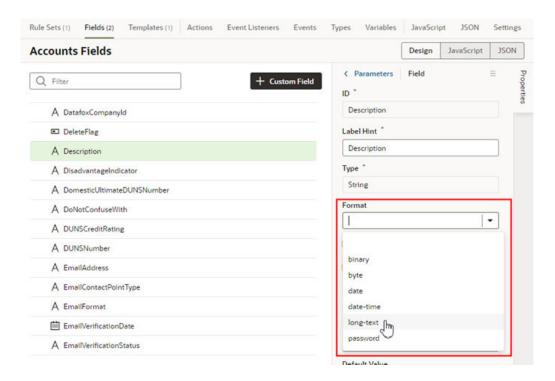


In this form, the User Assistance Density property for the first field (Name) is set to efficient, the second and third fields (Job Title, Last Updated By) are set to reflow, and the fourth field (Id) is set to compact. You can see that the cursor is in the Last Updated By field, and that the space below the field has expanded so that the user assistance text can be rendered below the field.

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.

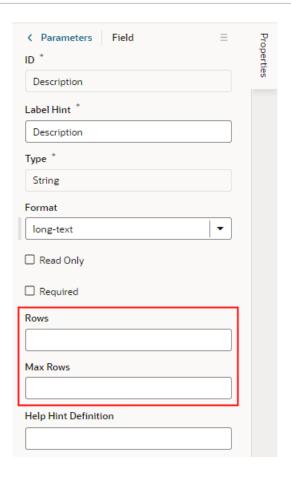


Notice that the field now has a blue dot bar next to it to indicate the field has been modified.

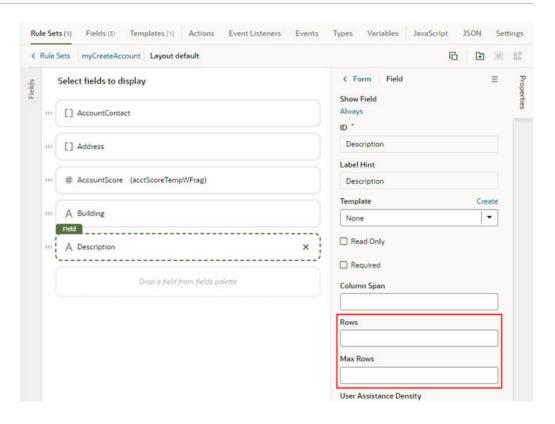
When you switch the format to long-text, two additional properties are displayed in the Properties pane.

- 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.





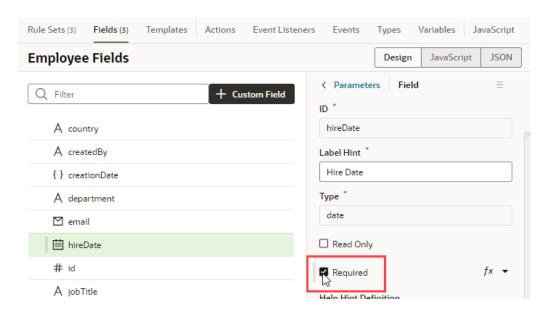
- To set a field to display as a text area in a particular form layout:
 - 1. In the Rule Set editor, open the layout and select the field in the center pane.
 - 2. Set the Rows property to the number of rows to display in the form by default.
 - 3. Set the Max Rows property to the maximum number of 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.



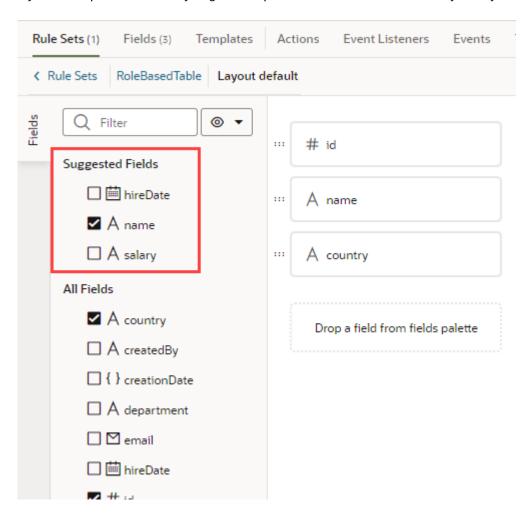
Set a Field as Required

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 rule set layouts or in a particular layout, but the Required property for a dynamic table applies to all rule set layouts and can only be set from the Layout's Fields tab.

- To edit a field's Required property in a Layout (applies to all the rule set layouts in the dynamic forms and tables in the Layout):
 - 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.

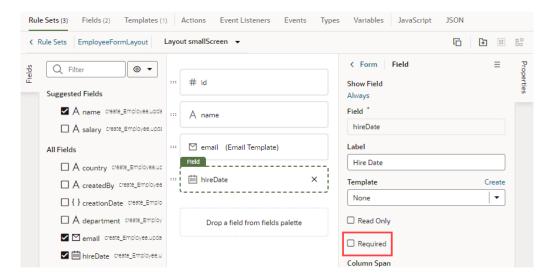


Fields that are marked as required show as Suggested Fields when you're building a layout to emphasize that they might be important or relevant to include in your layout:

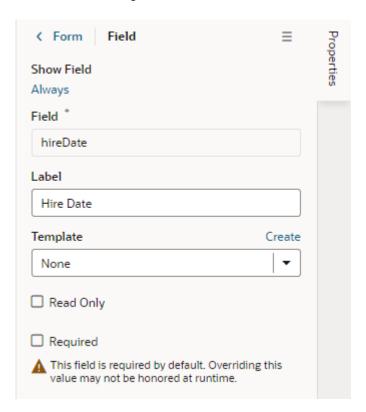


To edit a field's Required property in a particular layout (only for dynamic forms):

- 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.

Specify How a User Input Overrides a Field Value

The values of some fields are set using expressions or business rules. When this is the case, the default behavior is that a value calculated in this manner is overridden if a user enters a value. You can change this default behavior with the field's Value Precedence property.

To give an example of how a field's value might be calculated, say there is a field named Salary, and its value is determined by a business rule according to the value of a field named Title. If the value of Title changes, the value of Salary changes automatically. The default behavior for the field is that if a user enters a value for Salary, it overrides the calculated value for Salary set by the business rule. If you want a field's calculated value to override the user-entered value, you can use the field's Value Precedence property to change the default behavior.

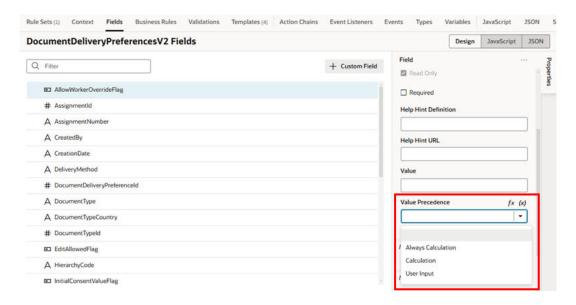
The following table describes the options available for the Value Precendence property.

Option	Description
User Input	Choose this option if you want the value entered by a user to always take precedence over any calculated value.
	The calculated value is used if the user doesn't enter a value. Once a user enters a value, the value will not be modified by any expressions or business rules.
	The User Input option is the default when no Value Precedence option is selected.
Calculation	Choose this option if you want the calculated value applied conditionally.
	The calculated value is used if the user doesn't enter a value. If a user enters a value, the value is saved until an expression or business rule further updates the value. In the Salary example above, if a user enters a value for Salary, and then changes the value for Title, the calculated value for Salary overrides the user-entered value for Salary.
Always Calculation	Choose this option if you want the calculated value to always take precedence over a user-entered value. You might want to do this to constrain a user-entered value to a value within a range or limit. For example, if a user enters a value for Salary that is too high, the calculated value will overrride the user-entered value when Always Calculation is selected.
	Note that, in this example, the user-entered value would be applied as long as it didn't conflict with the calculated value set by an expression or business rule.

To select an option for a field's Value Precedence property:

- In the Layout's Fields tab, select the field you want to work with.
- 2. Select an option in the Value Precedence dropdown list.





User Input is the default if no option is selected.

Map a Field to a Global Field Template

When global field templates are used in dynamic forms, you can choose the global field template that must be applied to a field by specifying the template's layout discriminant ID.

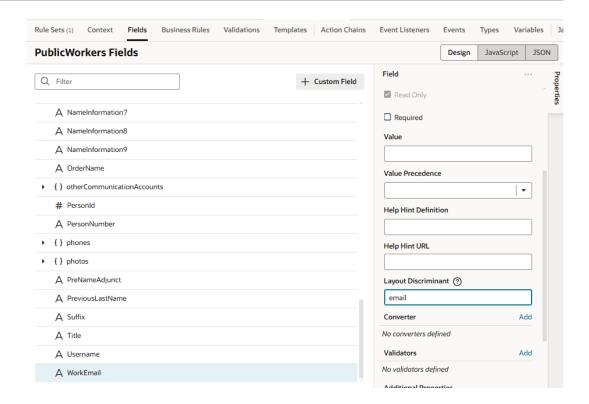
Global field templates are useful when you have very similar fields in dynamic forms, allowing you to standardize UI behavior (such as display attributes and default labels) for the fields. They are essentially the same as field templates, except that instead of defining a template and mapping for each field, they are defined and mapped once at the application level.



For information on how you can implement a global field template in a dynamic form, see the Global Field Template recipe in the Visual Builder cookbook. See also the Case-Insensitive LOV Search recipe.

After defining a global field template, here's how to map a field in a layout to a global field template:

- 1. In the Layout's **Fields** tab, select the field you want to work with.
- 2. In the **Layout Discriminant** field, enter the layout discriminant ID of the global field template you want to apply to the field.



You can find your template's layout discriminant ID in its field-templatesoverlay.json file, where the <code>\$byLayoutDiscriminant</code> property in the <code>addTemplateMap</code>
section defines the "discriminant": "globalFieldTemplate" key-value pair.
The <code>\$byLayoutDiscriminant</code> property specifies the mapping from layout discriminant ID to template ID. In this example, <code>email</code> is the ID you need:

After the layout discriminant ID is specified for a field, the corresponding global field template will be applied to the field at runtime.

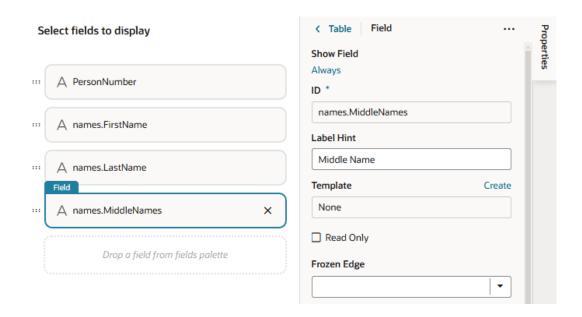
Use Conditions to Show or Hide Fields in a Layout

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:

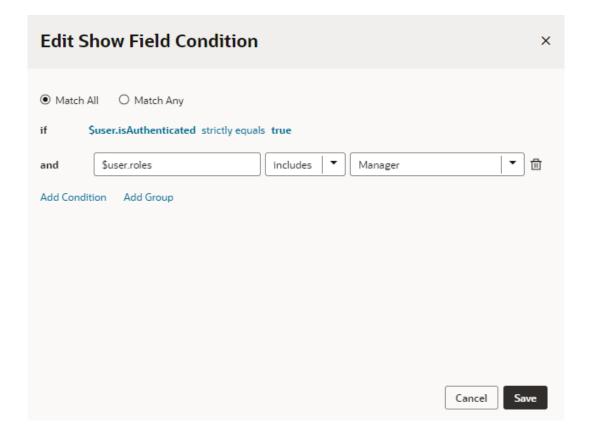
1. In the Rule Set editor, open the layout and select the field in the center pane.

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.



- In the Properties pane, click Always under the Show Field property to open the Edit Show Field Condition dialog box.
- 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.



Control How Fields With a Null Value are Displayed in Forms

If your dynamic form has any fields that might have no value (a null value), you can use the dynamic form's Null Value Field property to determine if those fields are displayed in the form.

To illustrate how this works, let's say you have a dynamic form that displays several fields, and, in some cases, the value of one of the fields (for example, JobCode) is null. By default, when the field's value is null, the Job Code field is displayed in editable forms, and hidden in read-only forms. Using the Null Value Field property, you can change this default behavior, for example, so that the field is always hidden when the field's value is null.

The following table describes the three Null Value Field property options.

Option	Description
Auto	This option is applied by default. The behavior depends on if the form is editable or read only.
	 If the form is an editable form, fields that have a null value are displayed in the form. If the form is read-only, fields that have a null value are not displayed in the form.
Show	When selected, fields that have a null value are always displayed in the form, regardless if the form is editable or read only.
Hide	When selected, fields that have a null value are never displayed in the form.

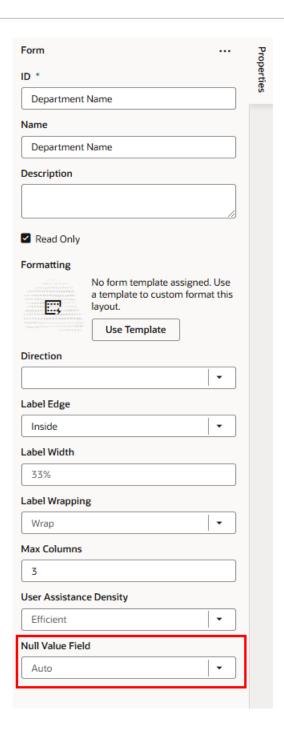


You cannot set the Null Value Field property in layouts that use a form template.

To set the Null Value Field property in a layout.

- 1. Open the the dynamic form's rule set.
- Open the layout where you want to set the property.The Null Value Field is a property on the layout, not individual fields, and it applies to all fields in the layout. You can set the property in each of the rule set's layouts.
- 3. In the layout's Properties pane, select a value for the Null Value Field property in the dropdown list.





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. You can also configure the field column's width to change the overall table display, and set the column name.

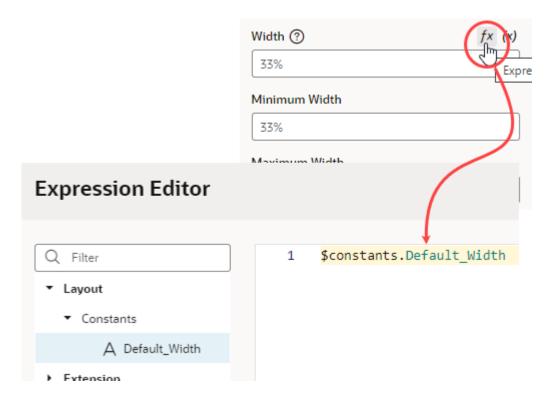
Setting these properties for a field in a layout only applies to the current layout. Other layouts are not affected.

To manage the sortability, sizing and names of 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 name of the column, enter the column name in the Label Hint field. This name is only used as the column header in the current layout, and overrides the name set in the Layout's Fields tab.
- 4. To set the default width for the field's column, 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 <code>Default_Width</code>, with a default value of 50%). Then, hover over the **Width** property and open the expression editor, define an expression using the <code>Default_Width</code> 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.

- 5. 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

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.

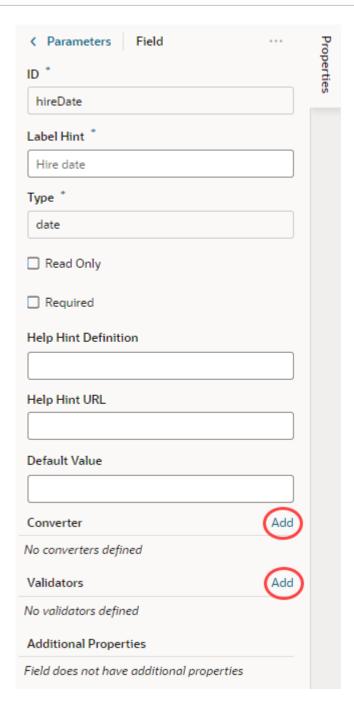
You can find details and examples in the Oracle JET Developer Cookbook:

- Built-in Oracle JET converters
- · Built-in Oracle JET validators

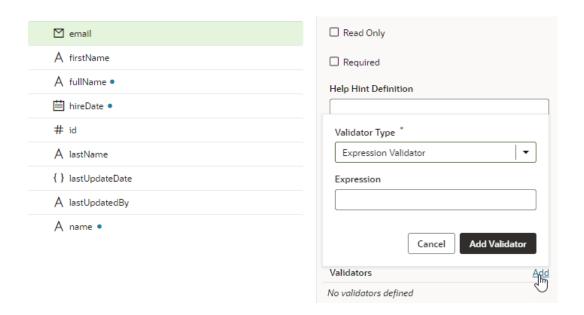
To add a converter or validator to a field:

- 1. In the **Fields** tab, select the field you want to work with.
- In the Properties pane, click Add next to Converter or Validators, then select one from the list.



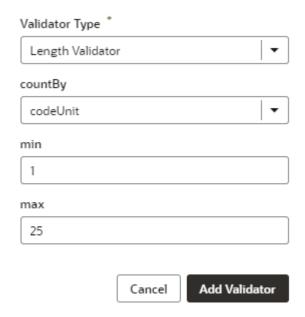


A default option is selected based on the field's type. For example, the default validator for an employee's Email field that uses the Email format is the Expression Validator:



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:

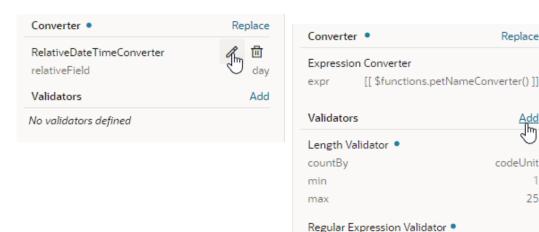


You can also create your own validator or converter by selecting the **From Code** option. With this type, the **path** field specifies the location of a JavaScript file that implements the custom validator or converter; the **name** field specifies the name of the constructor; and the **Option** field 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 relative Field.

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 4, 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.



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.

pattern

You can't create or modify the fields in your Oracle Cloud Application, or the service definition used by a Layout, but you can override some field properties, such as "Read Only" and

Replace

codeUnit

^[a-zA-Z]+[a-zA-Z0-9]*\$

25

"Required". So if a field's Required property is set to False in the service definition, you can override the property to make it more strict and set it to True. This won't change the description in the service definition, where the property will still be set to False. However, you can't override a property to make it less strict, meaning you can't set a Required property to False if it is already set to True in the service definition.

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.



The fields you create are only used in your rule set layouts; creating a field doesn't create a field in your Oracle Cloud Application, and doesn't change the service definition. For details on creating fields in an Oracle Cloud Application, see Define Fields in Configuring Applications Using Application Composer.

Create a Calculated Field

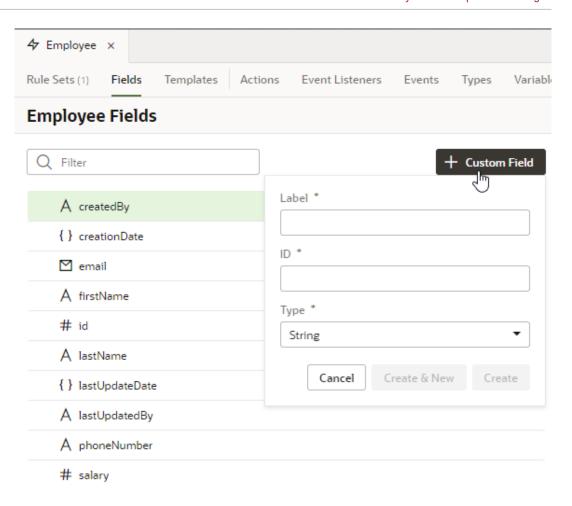
You can use a calculated field when you want to have a single field in your layout that, for example, contains some static string or an expression that is computed from the values of other referenced fields or objects.

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:

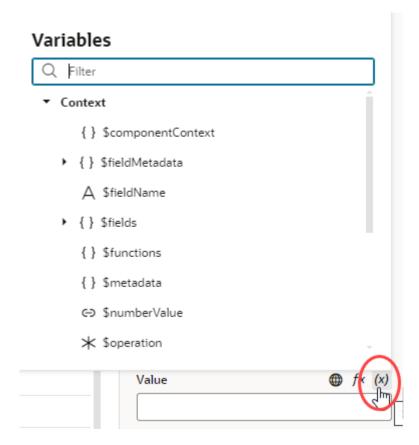
- 1. Open the dynamic table or form you want to work with in the **Layouts** tab.
- 2. Click the Fields tab, then + Custom Field.





- 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 $^{(x)}$ to open the Variables picker.



If you want to use an expression, click f^x 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 + ' ' $fields.lastName ]]
```

You can also use the <code>\$fieldmetadata</code> 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 To 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 blue dot to the right of the field name. In this screenshot, you can see the blue dot 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:

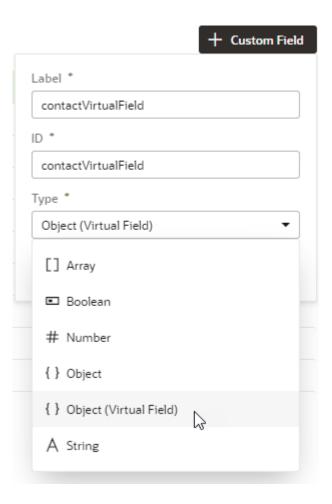
- the referenced fields can be edited in the layout; and
- the virtual field is rendered using a field template.

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.

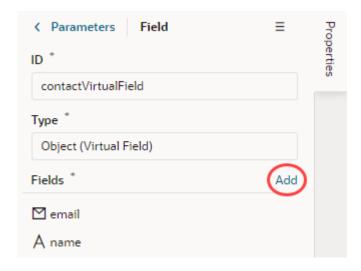
To create a custom virtual field:

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 subfields of objects.



5. Select a field in the Sort By dropdown list to define the field that should be used for sorting when the virtual field is used in a table.

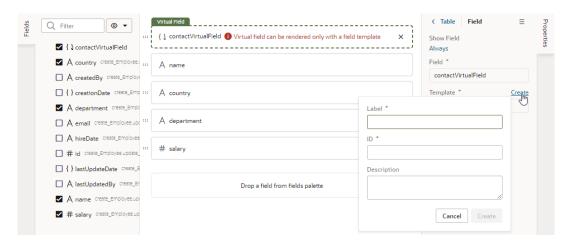
Only one field in a virtual field can be used for sorting. For example, if the virtual field FullName consists of a FirstName and LastName field, select LastName if you want it to be used when the table is sorted by FullName. The Sort By field will be used for sorting regardless of how the virtual field 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.
- 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.

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.

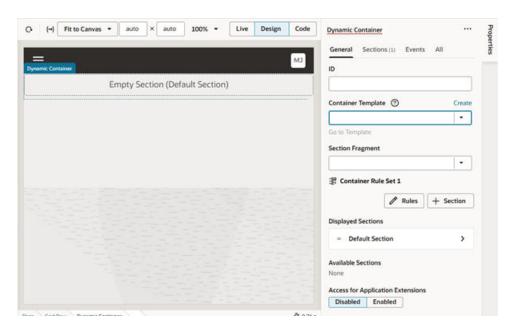
The easiest way to learn how to configure a dynamic container is to examine a real use case. Let's say you want to create a page that lets users toggle two layouts, one showing a form for adding a contact and another showing a table of assets. To do this, you'd create a dynamic container with two sections: 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.

Before you begin, make sure you've defined a service connection that lets you access external REST services, so your App UI can retrieve and send data to and from its REST endpoints. See Work With Services.

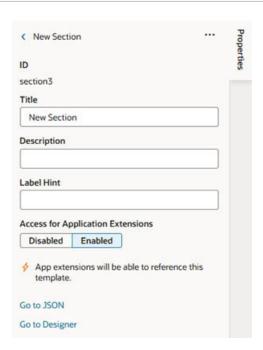
To add a dynamic container component to a page:

1. With your page open in the Page Designer, drag the Dynamic Container from the Components palette onto your canvas.

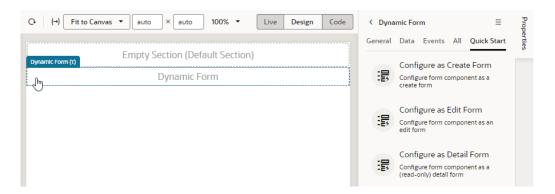
In this image, you can see one section (Default Section) was created for you automatically when you added the dynamic container. The Default Section section doesn't have any content yet, so you see the section's name on the canvas instead.

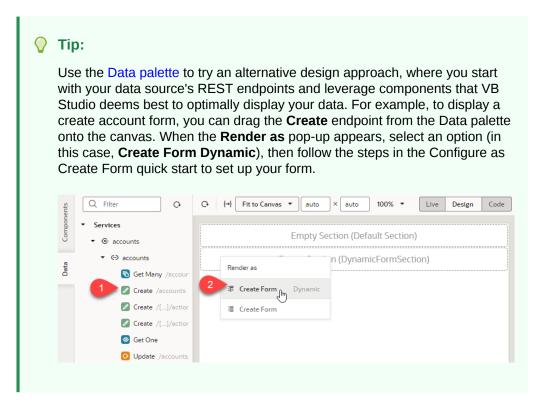


- 2. Create sections for the dynamic container. In our example, we'll create two sections, one for a dynamic form and another for a dynamic table.
 - a. In the Container Rule Set 1 section in the Properties pane, click + Section, then select + New Section in the dropdown list.
 - **b.** In the New Section's properties pane, give the section a name (for example, Dynamic Form). Return to the Dynamic Container pane.



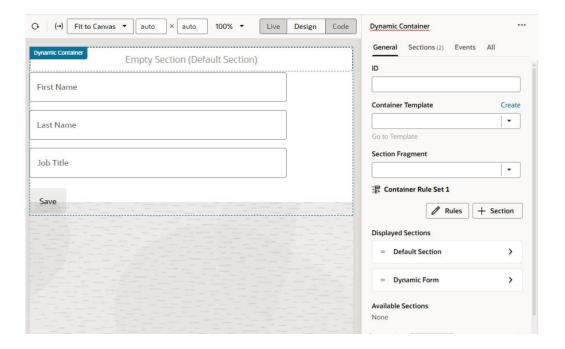
c. From the Components palette, drag a Dynamic Form and drop it onto the section on the canvas, select the Quick Start you want to use in the Properties pane, and follow the prompts to associate the dynamic form with a data source and a rule set. Click Finish.





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.

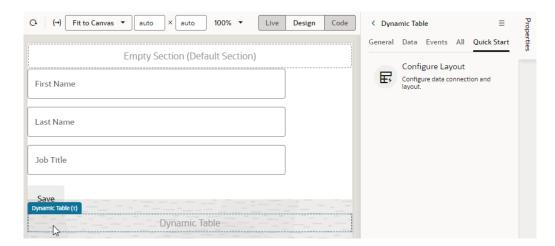
Click **Dynamic Container** in the Structure view to see the dynamic form added as a component to the page and its properties displayed in the Properties pane:



In the Properties pane, the Displayed Sections lists the sections currently displayed in the container. You can rearrange the order of sections by grabbing the handle next to the section name and moving it into the position you want.

The Properties pane also lists all the sections that you can display in the container under Available Sections. These sections are usually defined in dependencies, but also includes sections defined in your extension.

- d. In the Container Rule Set 1 section, click + Section again, and then select + New Section in the dropdown list to create another section; in our case, a section for a dynamic table.
- e. In the New Section's properties pane, give the section a name (for example, Dynamic Table). Return to the Dynamic Container's properties pane.
- 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.

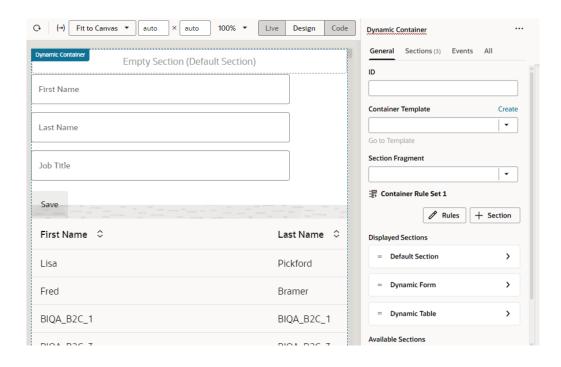


0

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:

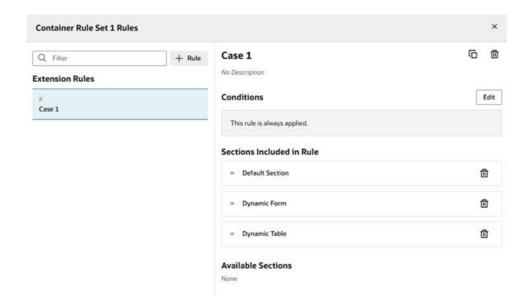


- 3. Set up the UI component and add the case logic to display your sections. Here, we'll add a Button component and a page variable to toggle the sections in the dynamic container.
 - a. In the Variables tab, create a variable that you can use in your conditions. For example, a Boolean-type variable 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 add the following actions:
 - 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 **Rules** to open the Container Rule Set editor.

The Container Rule Set editor contains one rule named Case 1, which is the default name. Sections Included in Rule lists the sections in Case 1. You can see that it lists the default section and the two sections you created:



c. In the Container Rule Set editor, click + Rules to create a new rule called Case 2. You can edit the rule name as required.

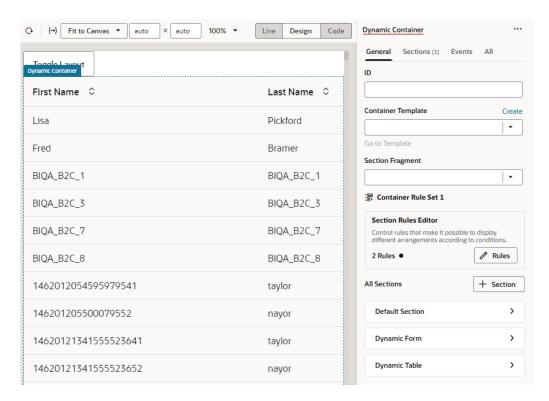
You can also duplicate a case if you want to create a new case containing the same sections.

When you create the new case, it is added before Case 1 in the list of rules. Remember, cases are evaluated from the top down. So in our example, at runtime the conditions for Case 2 will be evaluated first.

d. Select **Case 2**, and then add the sections you want to display by clicking + next to the section under Available Sections (Dynamic Table in our example).

- e. Select **Case 1**, and then remove the sections you don't want to display by clicking next to the section under Sections Included in Rule (Dynamic Table in our example).
- f. Click X in the corner of the editor to close it.

When you do this, you'll only see Case 2 rendered in the Page Designer, so in the dynamic container you'll only see Dynamic Table, which contains the table:



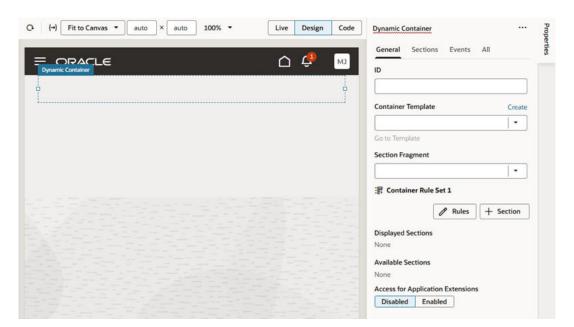
Notice that in the Properties pane, the Section Rules Editor panel displays "2 Rules". This because there are now two rules governing what is displayed in the container.

- Now let's edit the conditions of the two rules so that the button toggles between Case 1 and Case 2.
 - a. Click **Rules** in the Properties pane to open the Container Rule Set editor.
 - b. In the editor, select Case 1 and then click **Edit** to open the condition builder.
 - c. Click Code to open the Expression Editor and enter \$variables.layoutFlag == true. Click Done.
 - d. For Case 2, open the condition builder and enter \$variables.layoutFlag == false in the Expression Editor. Click Done.
- 6. Click **Preview** to preview the page, then click **Toggle Layout** to toggle between the two rules and change the sections displayed in the dynamic container.

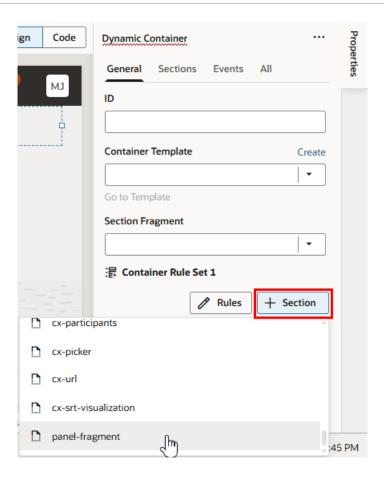
Add Fragments as Sections in a Dynamic Container

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.

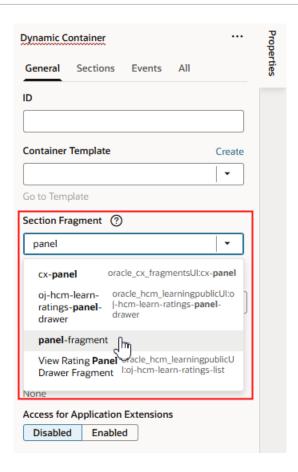
 With your page open in the Page Designer, drag the Dynamic Container from the Components palette onto your canvas. Here's a dynamic container with the Default Section removed.



- 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 Container Rule Set section, click + Section, then select a fragment in the drop-down list. This list displays all pageContent fragments (default tag). 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).



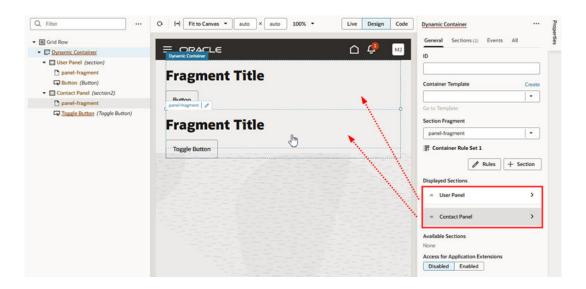
- b. Click + Section, then select the preferred fragment in the drop-down list. Only the fragment specified as the Section Fragment can be added to the section (in addition to any unused sections).
- 3. Select each section and change the section's title to make it more identifiable. For example, select the first Panel Fragment to open the details pane, and then click next to the section name to change its title, say, to User Panel:



Repeat the steps to rename the second Panel Fragment; this time, say as Contact Panel.

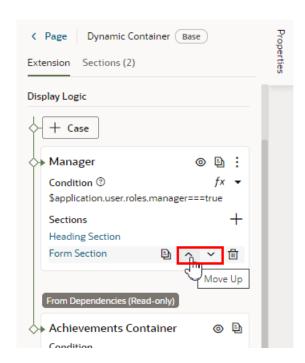
4. Select each section and configure its content (fragment) to suit your requirements.

Let's say you have a dynamic container and you want to use a fragment in two different sections, and then modify each section by adding other components. After adding the fragment, you would simply drag components from the Components palette into the sections to configure them as desired. In this example, you can see in the Structure view that each section contains the fragment containing the header component, and one additional component.



Re-Ordering a 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 for the section under Sections:



You can also use the \Box icon to remove a section from a case. Once removed from a particular case, the section is still available to use in other cases.



You can't change the section order or edit the contents of cases set in a dependency, but you can duplicate the cases and reorder the sections in the copy.

Guidelines for Working with Container Sections

Here are some things to keep in mind while working with container sections:

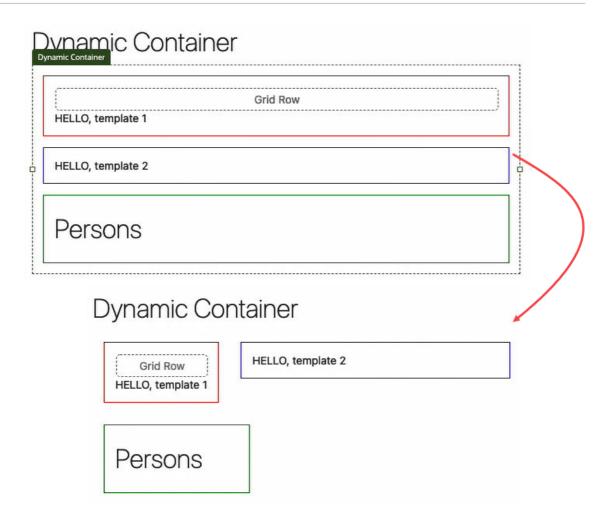
- When configuring a container, any section you create extends the full width of the
 container, although an App UI developer has more freedom in this respect. That is, while
 you can't have two sections side-by-side, the container can stretch to any height required
 to accommodate all the sections you 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, dynamic forms, or a button that starts an action chain in your extension
- 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.
- Unlike dynamic tables and forms, a dynamic container is not bound to a specific data resource, so a section can display data from any service connection available in the extension (and its dependencies).

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.

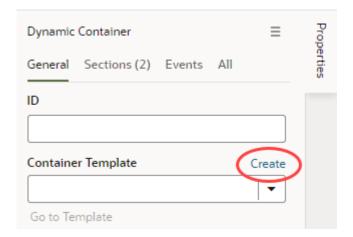




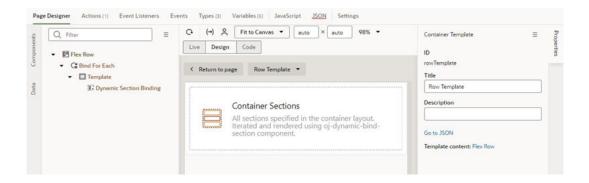
The preceding image shows a dynamic container's sections stacked vertically. Once you create a container template and switch its layout, the sections are placed next to each other. When they run out of space on the row, they automatically wrap to the next row (as shown at the bottom of the image).

To change the layout of sections in a dynamic container:

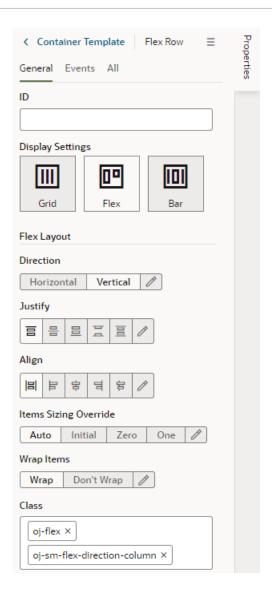
- 1. With your page open in the Page Designer, select the Dynamic Container component to view its properties in the Properties pane.
- 2. In the General tab, click **Create** next to Container Template.



- 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.



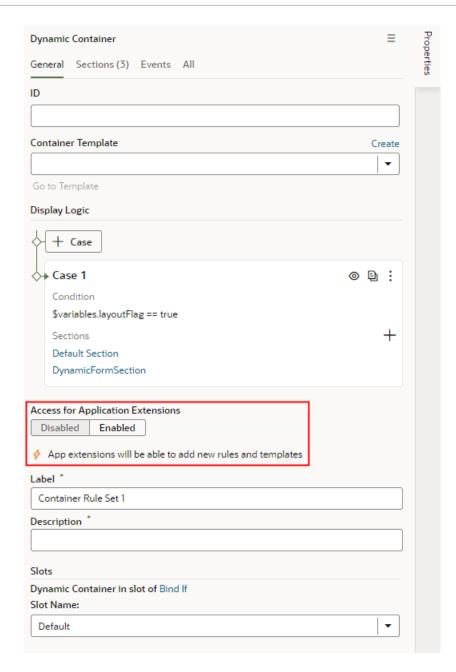
- 6. Select Horizontal. Take note of the oj-sm-flex-direction-column class, which sets the direction to a column and is included by default, is removed from the Class field.
- 7. Click **Return to Page** to see your sections laid out horizontally in a row, rather than vertically in a column.

Make a Container Available to Extensions

If you want to let others extend a container in your App UI, you need to explicitly mark the container as accessible to extensions. This lets someone add a dependency on the extension that contains your App UI, and then configure the container to add new cases and define sections.

To mark a container you've added to your App UI as accessible to extensions:

- 1. In the Page Designer, select the dynamic container.
- In the container's Properties pane, click Enabled under Access for Application Extensions.



The container's Label and Description fields are displayed in the Properties pane when you enable access for application extensions.

Enter the Label and Description so developers who extend the container know what the container is used for.

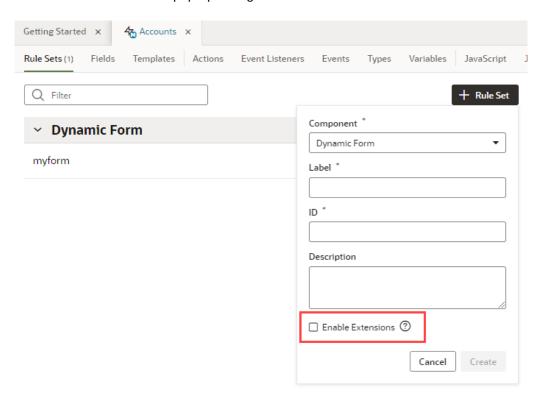


After you've made a container accessible to extensions, you should avoid renaming its ID. Renaming an ID might break the extensions that use it.

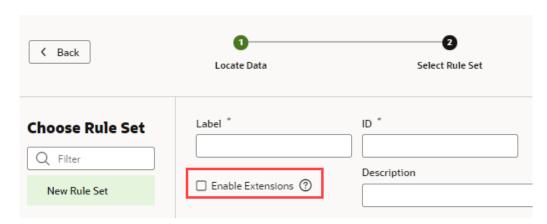
Make a Rule Set Extendable

If you want to let others extend a rule set in your App UI, you need to explicitly mark the rule set as accessible to extensions. This lets someone add a dependency on the extension that contains your App UI, then add new rules and layouts to the rule set, and use business rules to configure dynamic tables and forms.

- To mark a rule set as accessible to extensions when creating a new one in your App UI:
 - From the Layouts tab in the left navigator, click + Rule Set, and make sure you select Enable Extensions in the pop-up dialog.

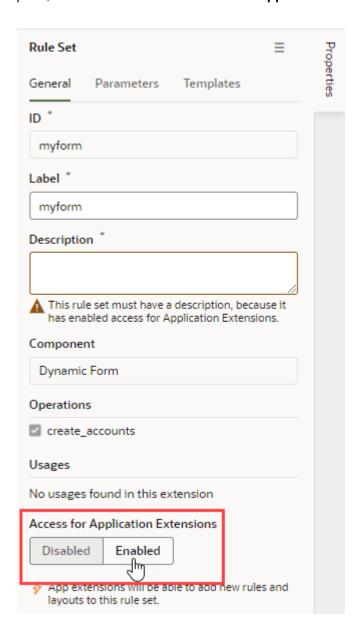


From any quick start that associates a dynamic component with a new rule set, select
 Enable Extensions in the Select Rule Set step of the wizard.



To mark a rule set as accessible after it's been defined:

1. From the Layouts tab in the left navigator, open the rule set, then in its Properties pane, select **Enabled** under **Access for Application Extensions**.



2. Enter a description so developers who extend its functionality know what the rule set is used for.



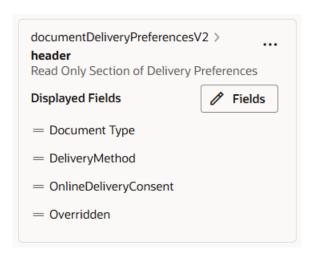
After you've made a rule set accessible to extensions, you should avoid renaming its ID. Renaming an ID might break the extensions that use it.

Users configuring the page can now see the layout listed as an extendable component in the Properties pane. To see how to customize how the components are listed, see Organize How Constants Are Listed in the Properties Pane.

Some dynamic layouts might only need to use one layout, so there is no need for a rule that has conditions, or for multiple rules. When this is the case, you can edit the rule set's layout.json file to hide the option to create rules in the Properties pane when users are extending the component. To hide the option to create rules, add @dt.dynamicFormOptions.allowConditions: "off" to the definition of the component in the JSON file:

```
"leadsDetailForm": {
    "type": "leadsRestApi",
    ...
    "label": "Detail",
    "@dt": {
        "dynamicFormOptions": {
            "allowConditions": "off"
        }
    }
}
```

When a user extends the component, the Properties pane displays a simpler pane, where the user can change the layout of the fields in the component:

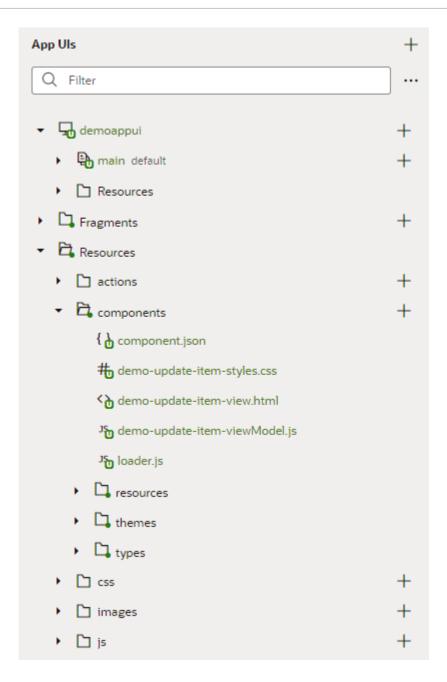


Add Custom Web Components to Pages

Oracle JET web components are reusable pieces of user interface code that you can embed as custom HTML elements. Web components can contain JET components, other web components, HTML, JavaScript, and CSS. You can create your own web component or add one from Oracle Component Exchange for use in your App Ul's pages.

To use web components in your pages, you can import them as a ZIP archive, or simply install them from the Component Exchange associated with your VB Studio instance. Similarly, web components that you create can be published to a Component Exchange to share with other developers, or shared as a ZIP archive.

Web components that have been imported to your extension for use in your App UI show under the global **Resources** and **components** nodes in the Navigator:



A web component must contain the files listed in the following table. It can also contain additional files and folders, for example, a SCSS file (optional file containing Sass variables to generate the web component's CSS) or resources such as translation files. For a more detailed description of web component architecture, see About Web Components in *Developing Oracle JET Apps Using MVVM Architecture*.

File	Description
loader	A RequireJS module that defines the web component dependencies for its metadata, View, ViewModel, and CSS. For web components, this name is always loader.js.
	A web component metadata file that defines its available properties and methods. For web components, this name is always component.js.

File	Description
view.h tml	Contains the View definition for the component and handles presentation of the data.
viewMo del.js	Defines the public variables and callback methods called at various stages of the component's lifecycle. Also defines public methods in the component's DOM element and the methods defined in the component's metadata. In brief, it implements most of the View's display logic, expose the methods for helping to maintain the View's state, updates the model based on the actions on the View, and triggers events on the View.
styles .css	Contains the custom styling for this web component.



Web components are currently not backward-compatible. This must be considered when importing and upgrading web components, and when upgrading your App UI. Your App UI and the web components it uses must be using the same JET version.

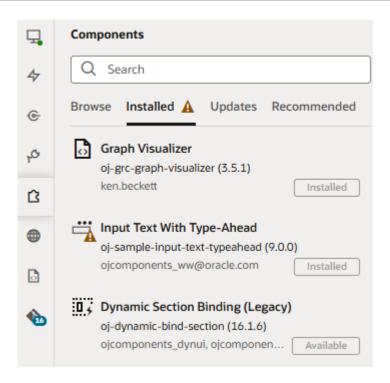
Install a Web Component from the Component Exchange

A Component Exchange is a repository of web components that can be installed in your VB Studio instance.

Your administrator specifies the Component Exchange that is associated with your VB Studio 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.

If your administrator has associated your instance with the Component Exchange, you can use the **Components** tab in the Navigator to add and manage components that you download from the Component Exchange:



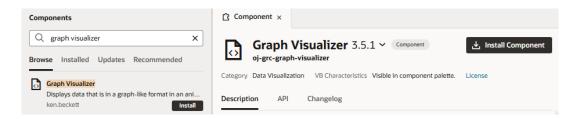


- Browse: Use to search the Component Exchange for components that can be installed, to open a component's details page, and to install components.
- **Installed**: Use to view a list of installed components. The tab displays details about each component, and components are badged to indicate warnings 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.

To add a component from the Component Exchange:

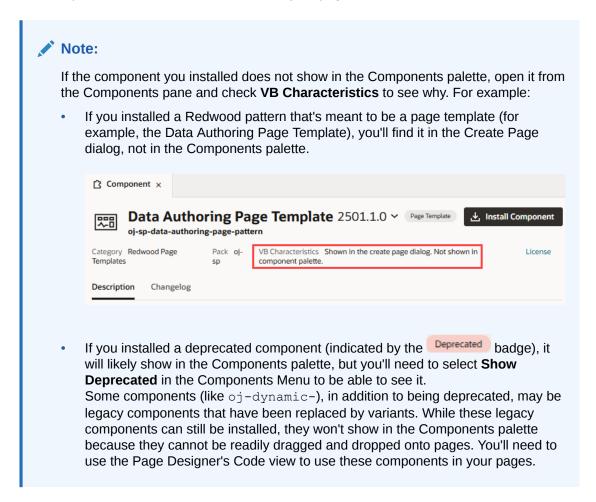
- Open the Components tab in the Navigator. Alternatively, open a page and click the Components palette in the Page Designer, then scroll to the bottom of the palette and click Get Components.
- 2. Locate the component you want to install and click **Install**.

Before you install, double-click the component to open it in the canvas area and learn more about it. You can view detailed information such as its features and steps on how to use the component. Take note especially of the **VB Characteristics** detail that tells whether the component will show in the Components palette.





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.



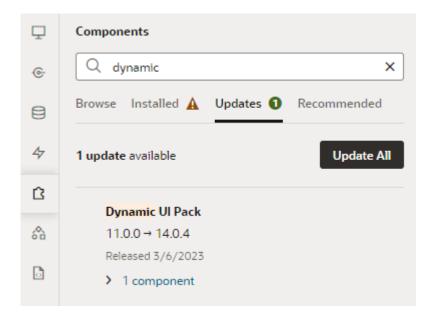
Update a Web Component from Component Exchange

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:

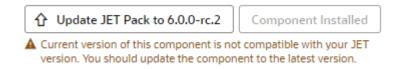
- 1. Open the **Components** tab in the Navigator.
- Open the Updates tab in the Components tab.

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.



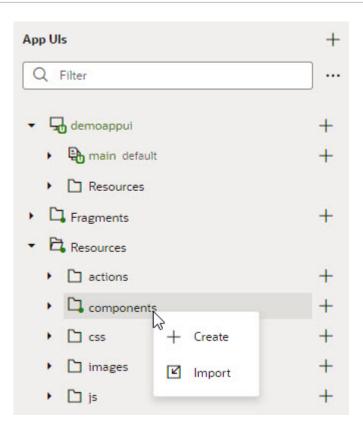
If the installed component is not compatible with the JET version in your VB Studio instance, you'll see a notice to that effect.

Import a Web Component Using a ZIP Archive

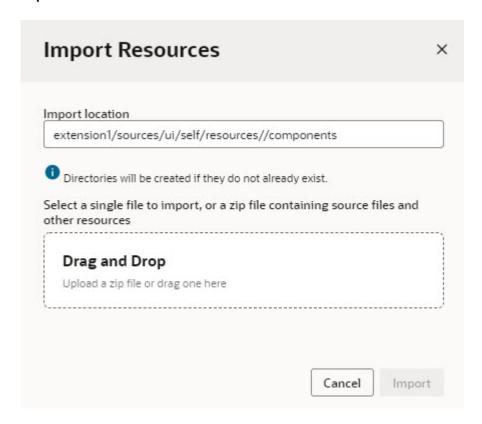
If you want to use a web component that's not available in your Component Exchange, you can import the component as a ZIP archive. For example, after your team member develops an applicable component, they can give it to you as an archive so you can try it in your application.

To import a custom web component archive:

- Select the App UIs tab.
- Expand the global Resources node.
- Right click the components node to bring up the context menu and select Import.



4. Upload your ZIP archive in the Import Resources dialog box by dragging it into the Drag and Drop area or by clicking the area and locating the file in your local system. Click Import to finish.



The imported web component is displayed in the list of components in the **Components** palette, under 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 App UI's extension. 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 for your App UI. 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 if you have multiple App UIs 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 VB Studio.

When you use the Create Component dialog box to create a component, the new component contains the JavaScript, HTML, style sheet, 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.

```
App UIs
                                            { } component.json ×
 Q Filter
                                                     "name": "demo-update-item",
                                                     "version": "1.0.0",
"jetVersion": "^12.0.0",
                                              4
                                                     "displayName": "demo-update-item",
     main default
                                                     "description": "A Web Component with form layout",
                                              6
   Resources
                                              7 ~
                                                     "properties": {
                                                       "itemId": {
                                              8 V
   Fragments
                                                         "type": "number"
                                              9
                                             10
   Resources
                                             11 V
                                                       "itemName": {
   actions
                                             12
                                                         "type": "string",
                                                         "description": "Description for the item-name attribute",
                                             13

→ Components

                                             14
                                                         "writeback": true
        { } component.json
                                             15
                                             16 \
                                                       "itemPrice": {
        # demo-update-item-styles.css
                                                         "type": "number".
                                             17
                                                         "writeback": true
                                             18
        〈 h demo-update-item-view.html
                                             19
         Jo demo-update-item-viewModel.js
                                             20 🗸
                                                       "itemDesc": {
                                                         "type": "string",
                                             21
         loader.js والمالية
                                                         "writeback": true
                                             22
        resources
                                             23
                                             24
        themes
                                             25
                                                     "methods": {},
                                                     "events": {},
       types types
                                             26
                                             27
                                                     "slots": {}
   css
                                     +
                                             28
     [☐ images
   is 🗀 is
```

To create a web component:

- Expand the global Resources node 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 displayed under the **components** node in the Navigator. If you need further details about creating a web component, refer to Create Web Components in the *Developing Oracle JET Apps Using MVVM Architecture* guide.



Manually creating web components makes them a part of your App UI's extension. 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 for your App UI. 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 if you have multiple App UIs that use the same components. Talk to your administrator for site-specific information on how to publish these components externally.

Uninstall a Web Component

When you no longer want to use an installed web component in your application, you can uninstall it to remove it from your Components palette.

To uninstall a component:

 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.
- Click Uninstall Component in the details page.



Work With Variables, Types, and Constants

When you use Quick Starts to build your App UI, you won't have to worry about variables and types because VB Studio 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 App UI's logic.

What are Variables?

Variables are named pieces of information that hold business state and are bound (via expressions) to *components* on your App UI'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 App UI.

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 VB Studio:

- 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 App UI based on the scope it is defined in:

- App UI: Variables defined at the App UI level are available anywhere in the App UI. They
 are useful for storing login names and other data that you want accessible both within and
 across an App UI'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	Used to retrieve the value of variables defined at the current App UI level. For example, if a variable called empName was defined at the App UI level, the \$application.variables.empName expression is used to get its value.	Current App UI
\$global	Used to retrieve the value of variables defined at the Unified Application level (for use by every App UI in your Oracle Cloud Application instance).	Across App UIs
\$flow	Used to 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	Used to 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	Used to 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.	Current fragment
	Every fragment has a unique ID, which is accessible within a fragment using \$fragment.info.id. You can use \$fragment.info.id used within expressions set on a component's ID property, or even the ID of a nested fragment.	
\$layout	Used to 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	Used to 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	Used to refer to event listeners of an App UI, flow, or page in a component, for example, \$listeners.onSelectionChange.	In a flow or page
\$event	Used to retrieve the content of an event's payload in an event listener. For an event listener on a custom event, \$event variable 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.	
\$initParams	Used to refer to initialization parameters defined at the Unified Everywhere Application level.	
\$modules	A shortcut to call methods exposed by an imported global functions module in the current extension. To access global functions in action chains, use the \$modules. <module-id>.<function-name>() syntax, for example, \$modules.commonUtils.titleCase() or \$modules.dateLocalUtils.dateToIsoString(), where titleCase and dateToIsoString are names of functions defined in the functions.json file for that extension. Any container that imports modules can expect \$modules to be available, for example, \$page.modules, \$fragment.modules, and so on.</function-name></module-id>	Advanced expression builder for Business Rules, Validation Rules, and Default Values, and JavaScript action chains



These variable definitions can be used much the same as any other variable in VB Studio, 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 <code>onValueChanged</code> event. (You can also add an <code>onValueChanged</code> 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, VB Studio 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

VB Studio provides several built-in variables that allow you to access application metadata.

currentAppUi

Use the currentAppUi variable on the global object to access some of the current App UI's metadata. The global object represents the Unified Application, available to every App UI in your Oracle Cloud Application instance.

Name	Description
\$global.currentAppUi.id	ID of the App UI (string).
\$global.currentAppUi.urlId	ID of the App UI as shown in the URL (string).
\$global.currentAppUi.displayName	Display name for the App UI (string).
\$global.currentAppUi.description	Description of the App UI (string).
\$global.currentAppUi.defaultPage	Default page of the App UI, if it exists (string).



Name	Description
\$global.currentAppUi.defaultFlow	Default flow of the App UI, if it exists (string).
\$global.currentAppUi.applicationS tripe	Stripe used by the App UI (string).
\$global.currentAppUi.pillarTheme	Pillar used by the App UI (string).
\$global.currentAppUi.pillarThemeM ode	Pillar theme mode used by the App UI (string).
\$global.currentAppUi.icon	Icon used by the Ask Oracle Navigator menu for the App UI (string).
\$global.currentAppUi.usage	Reserved for internal use.
\$global.currentAppUi.menuDisplayN ame	Name of the App UI as displayed in the Ask Oracle Navigator menu (string).
\$global.currentAppUi.extensible	Whether the App UI can be extended (Boolean).
·	·

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 App UI. 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 <code>\$page.currentFlow</code> variable to retrieve the ID of the current flow.

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.json.
\$application.info.description	Application description as defined in app.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.
<pre>\$page.info.title</pre>	Page title as defined in page-id- page.json.



Name	Description
\$page.info.description	Page description as defined in page-
	<i>id</i> -page.json.

path

Use the path variable to build the path to a resource, such as an image located in a folder.

Name	Description
\$extension.path	Path to retrieve a resource located in the extension-level resources folder, which can be used by all App UIs in the extension.
\$application.path	Path to retrieve a resource located in a particular App UI's resources folder.
\$flow.path	Path to retrieve a resource in a particular App UI's 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, if it exists (string).
\$application.user.fullName	Display name of the user (string).
\$application.user.roles	List of user roles (array of strings).
\$application.user.permissions	List of user permissions (array of strings).
\$application.user.isAuthenticated	Whether user is authenticated (boolean).
\$application.user.guId	User GUID (string).
\$application.user.userName	Name of the logged-in user (string).

Create Variables

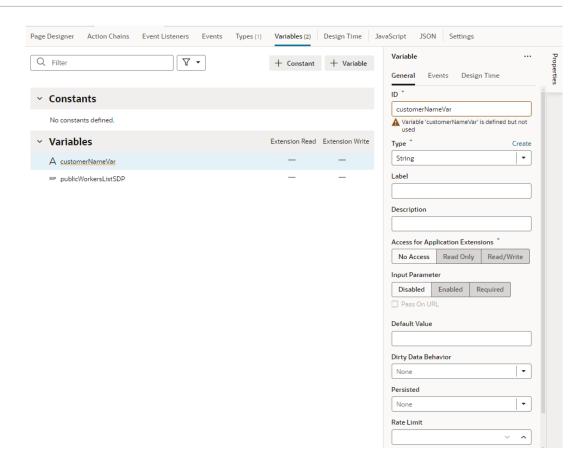
You can create variables and constants in App UI, flow, and page artifacts as well as in dynamic layouts and fragments. Variables and constants are assigned a scope based on where they're created, and this scope determines where they can be used. When deciding where to create a variable or constant, consider where you want to use it.

To create a variable or constant in an artifact:

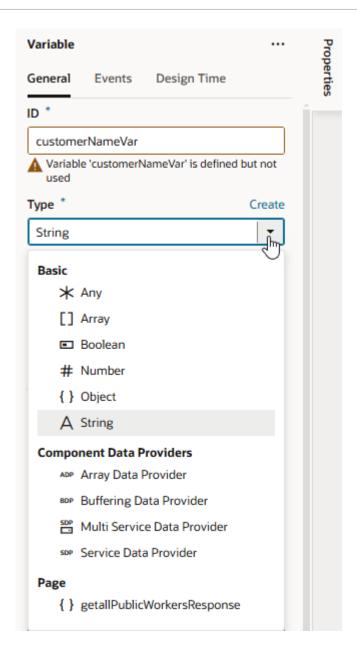
1. Click the **Variables** tab to open the Variables editor for your scope.

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:





- 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 a type in the **Type** drop-down list (default is **String**). The Type drop-down list displays the built-in types as well as any custom types that can be applied to the variable. For example, the <code>getallPublicWorkersResponse</code> type shown here is based on an endpoint:

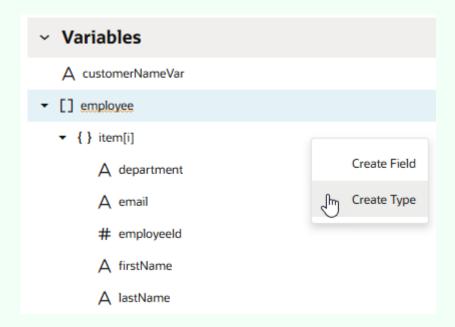


To use a custom type with your variable, make sure the type is already defined.



Tip:

You can quickly create a new type based on the structure of a variable, if the variable uses array or object as its type. Simply right-click the variable and select Create Type. This way, you can readily create a type based on a variable's existing structure and assign the variable (as well as other new ones) to that type.



Conversely, to quickly create variables for a custom type, switch to the **Types** editor, select your type, right-click, and select Create Variable.

5. After the variable is created, select it to edit its properties in the Properties pane. Here are some key properties:

Property	Description
Туре	Identifies the variable type, which could be a specific primitive type (string, boolean, number, and so on); a structured type such as an array or object; a dynamic type (any); or a built-in type such as Service Data Provider or Array Data Provider.
Access for Application Extensions	Makes a variable or constant available to App UIs in another extension. See Make Variables and Types Available to Extensions.
Input Parameter	Marks a variable as input, indicating that the variable is part of the contract for navigation. Can be set to Disabled (default), Enabled , or Required . Choosing Required implies that the variable must be set to allow navigation to that page, flow, or App UI. If you choose Enabled or Required , you can select Pass on URL to pass the input parameter on the URL of the page, flow, or App UI that you are invoking (by adding <code>?paramName=Value</code> to the end of the URL). You can use this approach to bookmark pages that will show specific data based on the parameter.
	You can also enter a Label and Description to show within a ② icon; page users can then click the icon in the Page Input Parameters dialog to get some context on how best to set the parameter's value.
	See Enable Variables as Input Parameters.



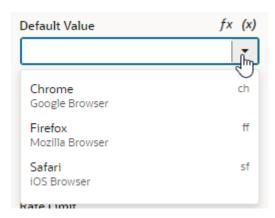
Property

Description

Default Value

Specifies the default value to initialize a variable. This property can be an expression or a static value.

For string-type variables, you can 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, Date, Date Time, Enum Values, or Time Zone 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.

Dirty Data Behavior

Tracks changes in a variable's state, marking it as "dirty" when its current value differs from its initial value. See Track Variables to Detect Unsaved Changes.

Persisted

Sets the lifespan of the variable to be longer than the page, for example, when you want to keep an authorization token for the duration of a session. This property ensures that even if the page is refreshed, the token will still be available throughout the session. Can be set to:

- Device: Stores the variable in the browser's local storage and persists it on the device where the App UI 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 App UI navigated away from this
 page.
- None: Variable is not stored (default).

Rate Limit

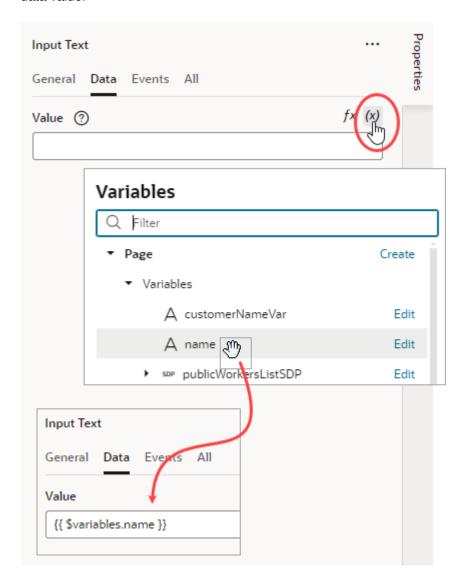
Limits how often the <code>onValueChanged</code> event is triggered, which is whenever the variable's value changes. By default, the event listener that "listens" for this event waits for the value (specified here in milliseconds) to expire after all changes stop to fire the event.

You can trigger an action chain when a variable's value changes by adding a listener for the <code>onValueChanged</code> event and selecting the action chain that the change will initiate. 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. To do this, switch to the <code>Events</code> tab in the variable's Properties pane, then specify the action chain that the variable's change will initiate. See <code>Start</code> an Action Chain When a Variable Changes.



Now that the variable is defined, you can bind it to a component to display its data. Here's a simple example: Let's say you have an Input Text component on a page to display an employee's name. If the data for this component comes from a string-type <code>name</code> variable, you bind the component to this variable.

To do this, open the page containing the Input Text component in the Page Designer and select the component. In the component's Properties pane, open the **Data** tab. Hover over the **Value** field and click (x) to open the Variables picker, then select name under Page to populate the data value:



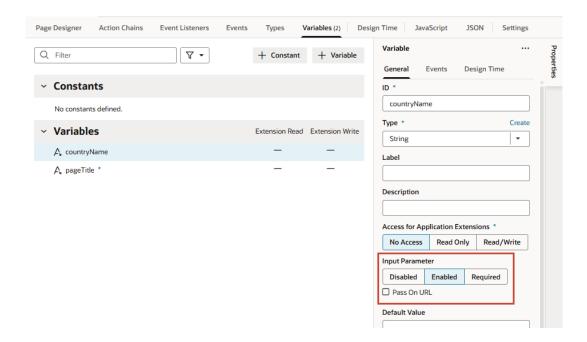
For more information, see Bind a Component to Data.

Once a variable is used in an App UI, 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 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.
- 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. To pass the parameter on the URL of the pages or flows you're 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. Optional: To make the input parameter easier to use on a page, enter a **Label** or **Description** to show within a ② icon, which page users can then click to get a more meaningful identifier and some context on how best to set the parameter's value.
 - Consider adding a Label if your users work in Express mode, where the variable's label, when defined, is used instead of the ID to display the input parameter as a page property.
- 5. Optional: Set a **Default Value** to initialize the variable or constant.
- 6. Optional: To display a list of values retrieved from a service (like a REST endpoint) for a string-type variable or constant enabled as an input parameter:
 - a. Switch to the **Design Time** tab in the variable's Properties pane.
 - Select the Subtype as LOV.



- c. Click Select Endpoint and choose the REST endpoint whose response is used to provide the list of values. The endpoint must already be set up and available to your App UI. The endpoint response must also be in JSON format.
- d. When the REST call requires input and URI parameters, click the associated Assign or Not Mapped link and map the parameters to the value they should be replaced with. The value can be a VB Studio constant.

If the path and URI parameters depend on other constants (for example, your REST service uses the result of an earlier selection as part of its own request), use expression notation in the parameter values to indicate which constant values to use. You'll need to do this in the source JSON. For example, you would use the following expression to instruct a service request to use the current value of the "dept" constant as the value for the path parameter "department":

```
"pathParameters": {
   "department": "[[ $constants.dept ]]"
}
```

When writing the expression:

- Use simple direct references. Calculated expressions such as "[[\$constants.dept + "_"]]" do not work as expected.
- Use constants; variables cannot be used.
- Make sure the referenced constants are accessible to the extension making the LOV service call.
- e. If your endpoint response is *not* an array, in the **Items Path**, specify a dot-separated path from the root of the response object to the array containing the list of values. You can leave this field empty if your response is an array.

For example, you would enter results when the response is not an array, as shown here:

```
"now": {
 "type": "string",
 "description": "wow",
 "defaultValue": "505642",
 "input": "none",
 "@dt": {
    "label": "Films Now Playing",
    "subtype": "lov",
    "valueOptions": {
      "service": {
        "request": {
          "endpoint": "site myappuiExtension:movies/getNowPlaying",
          "uriParameters": {
              "api key": "4174b7d9a7b4bf87342c98e2289c6ee6"
        },
        "response": {
          "itemsPath": "results",
          "mapping": {
            "value": "id",
            "label": "title",
            "description": "overview"
```

```
}
}
}
}
```

- f. In the **Value** field, specify the field from the response object to use as the actual value of the variable or constant, for example, id.
- g. Optional: In the Label field, enter the field from the response object to use as the primary display name of the item, for example, title. It can also be an array of field names, for example, ["firstName", "lastName"].
- **h.** Optional: In the **Description** field, enter the field from the response object to use as the item description, for example, overview.



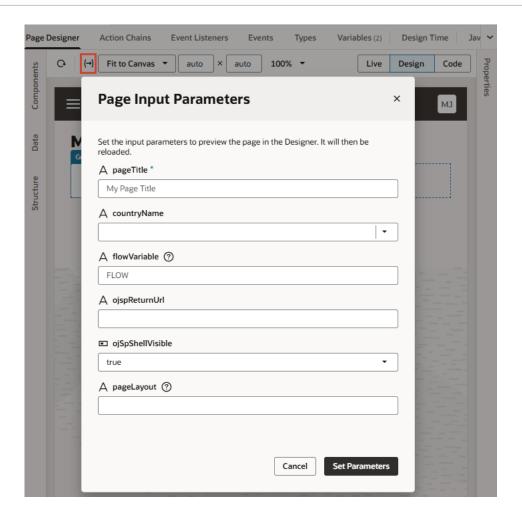
Tip:

If you have a large number of variables defined, click and select **Input**Parameters Only to filter and view only those variables used as input parameters.

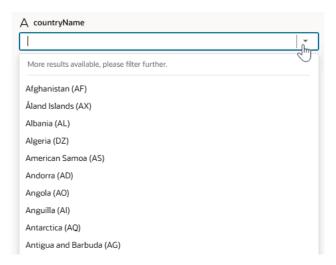
When a variable marked as an optional or required input parameter is used in your app, you can set its value on a page to see how the page displays:

- 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). If the variable is 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 and are marked with an asterisk (*).
- 3. Set the parameter value and click **Set Parameters**.





If you set up the list of values for a parameter, you should be able to select its value from a drop-down list, like this:



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 App UI. 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:

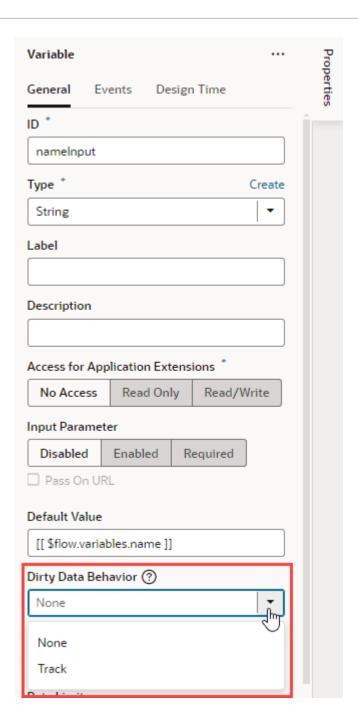
- Use the Get Dirty Data Status action to query for dirty variables in the action's current scope as well as in any containers within
- Take some action if dirty variables exist, for example, use the Fire Notification action to display a message. Note that a variable whose value was updated but changed back to its initial value is not considered dirty.

To enable a variable for tracking:

- 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
- 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.

and constants—in an app, flow, page, fragment, and layout.





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, layouts, or both, those will be checked as well. Further, if the page or its fragments and layouts are extendable, tracked variables in those extensions will also be checked.

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, layouts, or both, tracked variables in those scopes will also be reset. Further, if the page or its fragments and layouts are extendable and include tracked variables, those variables will be reset as well.

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.

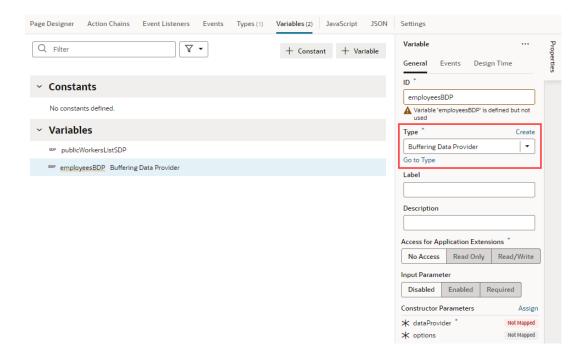
The Buffering Data Provider is a wrapper that provides buffering for an underlying data provider, so edits can be committed to the data source later on. The underlying data provider is responsible for data fetches, while the Buffering Data Provider takes care of merging any buffered edits with the underlying 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:

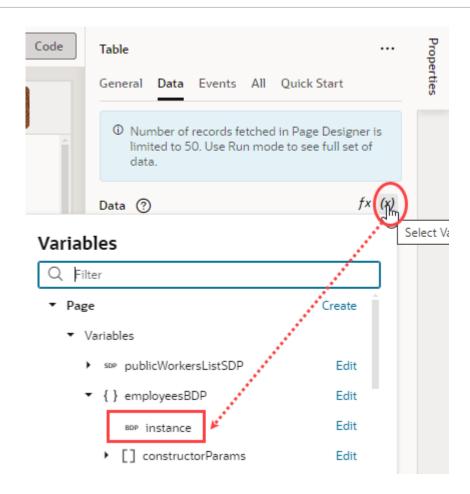


1. Open the Variables editor for your scope and create a variable of type Buffering Data Provider (for example, employeesBDP). Here's an example of a newly created Buffering Data Provider variable:



- 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 publicWorkersListSDP 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**.
- 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.publicWorkersListSDP with \$variables.employeesBDP.instance:





Create Types

Every variable in VB Studio 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 App UI's pages, VB Studio 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 App UI, 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.

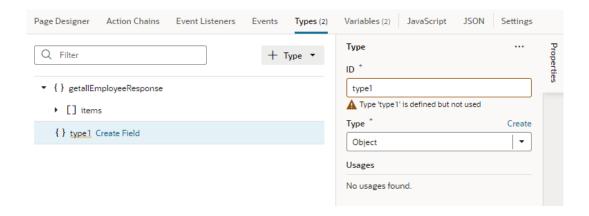
Create a Custom Object or Array Type

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:

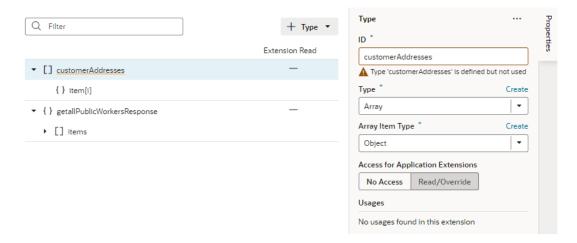
- Select your App UI, flow, or page artifact in the Navigator. You can also select a dynamic layout or a fragment.
- 2. Click the **Types** tab to open the Types editor.
- Click + Type and select Custom.



- 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 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.



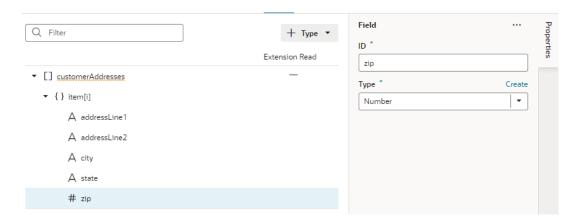
The new type is added to the list in the Types editor, with its properties displayed in the Properties pane. Here's an example of a newly created array type:



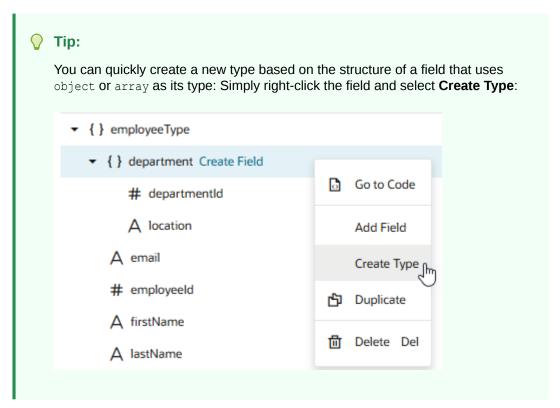
- 5. Define the type's structure by adding attributes:
 - a. Hover next to the new type (or item type), then click **Create Field** to add an attribute.
 - **b.** In the field's Properties pane, update the ID as needed and select a type for the new attribute.



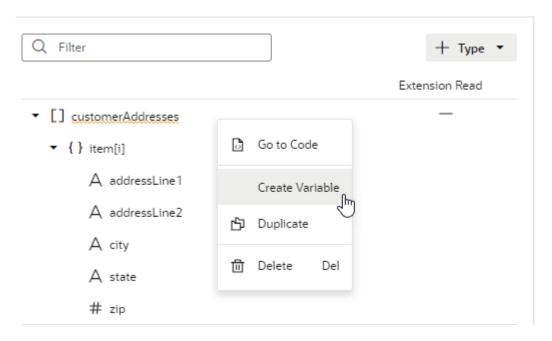
You can add as many attributes as you need to refine the type's data structure:







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 to that page.

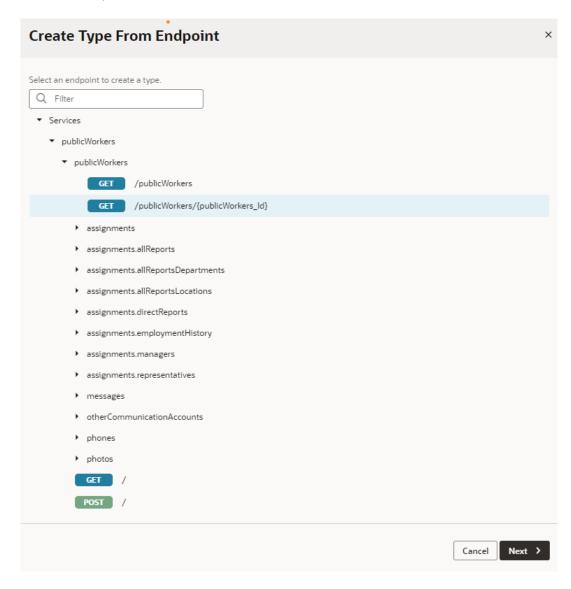
Create a Type From an Endpoint

When you create a type from an endpoint, you define a custom type structure by using the endpoint's (response) data structure.

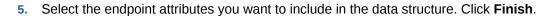
For example, when sending a request to a <code>get_publicWorkers</code> endpoint, you might want the structure of the response to be an array with the id and a few specific fields (say, a string name and email, among others). You can create a type by selecting the endpoint, then choosing the fields that you want in the response. All variables that are assigned this custom type will have the same data structure.

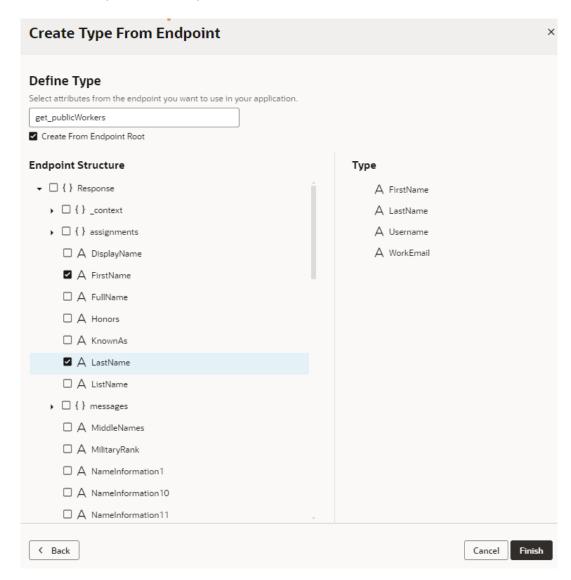
To create a type from an endpoint:

- Select your App UI, 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.
- Click + Type and select From Endpoint.
- 4. Select an endpoint from the list. Click **Next**.

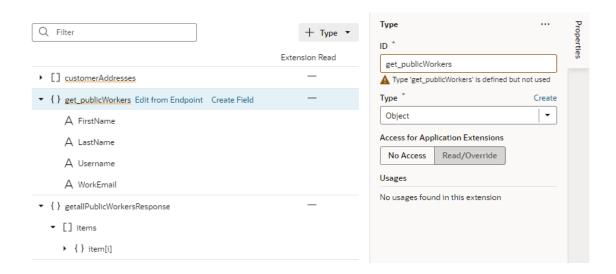




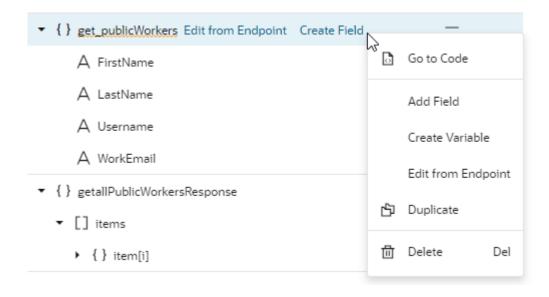




When your new type displays in the Types editor, you see it is an object type containing the endpoint fields you selected.



Use the options in the right-click menu to take further action (for example, to create a variable for this specific type or to add more fields from the endpoint).



When the type is associated with a variable, you can view its usage information under **Usages** in the Properties pane (for example, which variables are based on it and which pages use those variables). You can click a usage to readily navigate to that page.

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 VB Studio, 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 VB Studio variables. You'll only need to provide a typescript definition file (*.d.ts) or a typescript file (*.ts) that defines your type's details. VB Studio will parse your type definition, generate a JSON representation that is

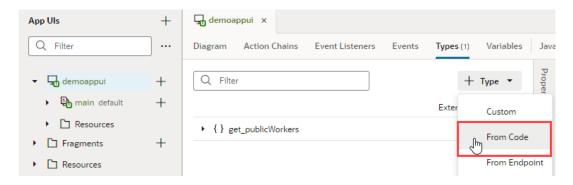
compatible with existing type schema, and create an InstanceFactory type, which you can then assign to an InstanceFactory variable (vb/InstanceFactory). The InstanceFactory variable uses the InstanceFactory type and additionally the list of constructor arguments declared by the type to define its constructor 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:

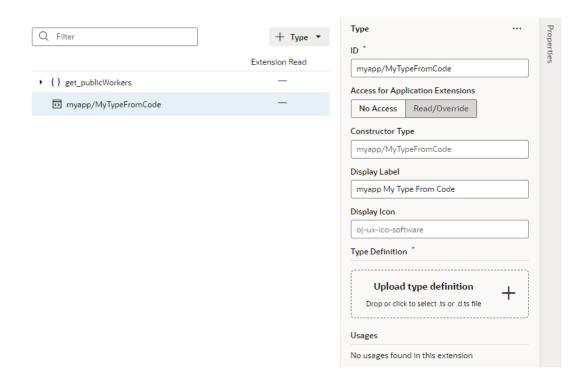
- Select your App UI (or fragment) in the Navigator.
- 2. Click the **Types** tab to open the Types editor.
- Click + Type and select From Code. This option is available only at the App UI (or fragment) level, so make sure you've selected the correct node in the Navigator.



4. Enter the type name using namespaces (for example, myapp/MyTypeFromCode) and click **Create**. Namespaces separated by a slash (/) help organize the types.

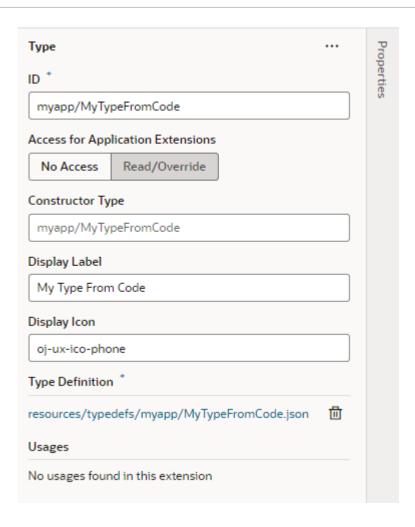
An InstanceFactory type is created, with its default display icon set to oj-ux-ico-software. A default display label is also generated based on your values, as shown here in the Properties pane:





- 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 My Type From Code and the display icon to oj-ux-ico-phone.
- 6. Drag and drop into the drop target area (or click **Upload type definition** to provide) a typescript (.ts) or typescript definition (.d.ts) file that specifies the type's details:

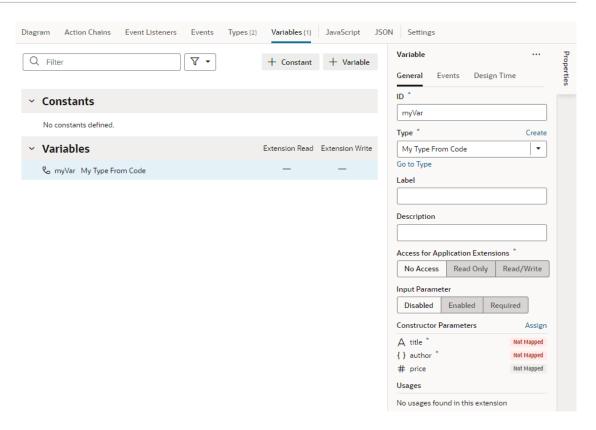




VB Studio converts your type definition to a JSON representation, then saves the JSON file under the App UI's resources/typedefs/myapp folder, where myapp is the namespace specified when you initially created the type.

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 on the Variables tab. For example, here's a myVar variable assigned to the My Type From Code 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 <code>incidentsView</code> variable references <code>incidentsSDP</code>, 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"
      ]
    }
 1
```

Any time the <code>incidentsSDP</code> variable changes (that is, a new instance is created), the <code>incidentsView</code> variable re-creates a new instance of <code>ojs/ojlistdataproviderview</code>. This also means that components bound to either variable are notified of the change.

Make Variables and Types Available to Extensions

If you want a variable, constant, or type that you've defined in your App UI to be available to App UIs in another extension, you can mark them accessible to extensions. This way, your

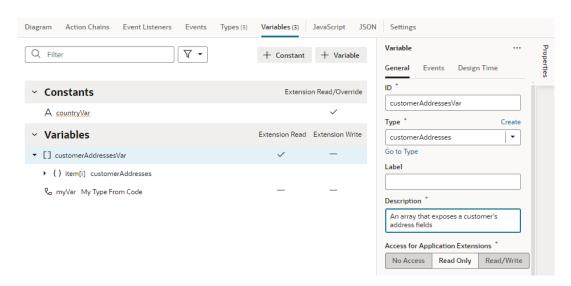
artifacts become available to App UIs in another extension when someone adds your App UI as a dependency.



Marking your variable or constant accessible to extensions means letting other extension developers change its value. Before doing this, carefully consider the consequences of allowing others to overwrite your values.

- To mark your variable or constant available to App UIs in another extension, go to the Variables editor and select the variable or the constant.
 - In the variable's Properties pane, select Read Only to allow other App UIs to read the variable's value; select Read/Write to allow other App UIs to read and modify the variable's value.
 - In the constant's Properties pane, select Read/Override to allow other App UIs to read or override the constant.

Variables that are accessible to extensions have a check mark () against them in the **Extension Read** and/or **Extension Write** columns under Variables in the Variables editor. Constants have a check mark in the **Extension Read/Override** column under Constants.



When you expose a variable or constant to extensions, make sure you enter a description so developers who extend its functionality know what it's meant for.

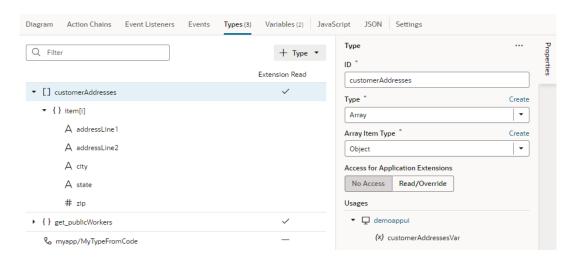


Tip:

If you have a large number of variables defined, click and select **Exposed to Extensions Only** to filter and view only those variables marked accessible to extensions.

To make your type available to App UIs in another extension, go to the Types editor and select the type, then in its Properties pane, select **Read/Override**.

Types that are accessible to extensions have a check mark (\checkmark) against them in the **Extension Read** column on the Types editor, indicating that they can be read by other App UIs.



Note:

After you've made a variable or constant accessible to extensions, you should avoid renaming its ID. Renaming an ID might break the extensions that use it.

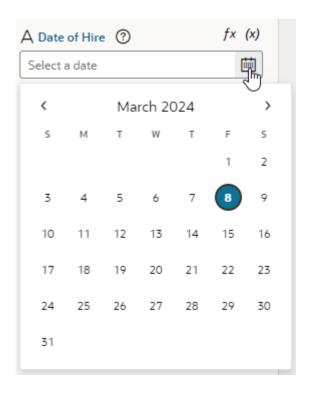
Configure How Variables are Customized in the Properties Pane

You can customize variables and constants when developing an App UI, so that users extending the page have a more intuitive experience when configuring values in the Page Designer's Properties pane.

When a page is open in the Page Designer, the page's extendable variables and constants are listed in the Constants tab in the Properties pane. Users extending the page can set the values of those variables and constants, usually displayed as text fields in the Page Designer's Properties pane. For example, here's how a constant used to store a date displays by default in the Properties pane:



In some cases, a different UI component can make editing the variable or constant more intuitive. In the case of the constant that stores a date, displaying a date picker instead of a text field can improve the experience when a user edits its value in the Page Designer:



To customize the UI component displayed for a variable or constant in the Properties pane, you use the Design Time tab in the Variables editor. You can also edit the JSON directly in the JSON editor.



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 the Customizing the JSON section below for a full list of customization options.

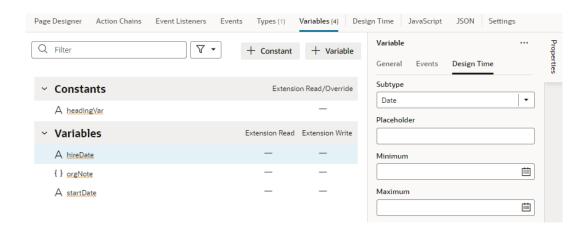
Customize Variables and Constants in the Variables Editor

To customize the UI component displayed in the Page Designer:

- Open the Variables editor.
- 2. Select the variable or constant you want to customize.
- 3. Select the **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, you'll see fields for setting Minimum and Maximum limits for the date:





Here are steps for some common customization options for string-, object-, and number-type variables and constants:

Customiz Steps in Design Time Tab Result in Page Designer ation Option To display For a string-type variable or constant: a color A textColor In the **Design Time** tab, select the **Subtype** as picker: ♡ Choose a color Optional: In the Placeholder field, specify a hint text for the variable; for example, Choose a color. 3. 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.

ation Option

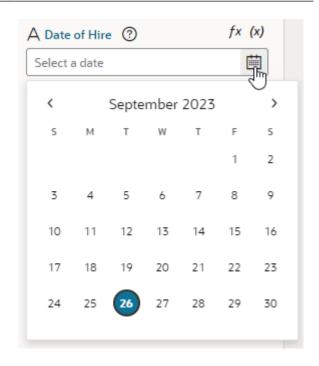
Customiz Steps in Design Time Tab

Result in Page Designer

To display a date or date-andtime picker:

For a string-type variable or constant:

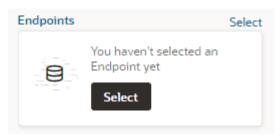
- In the **Design Time** tab, select the **Subtype** as Date or Date Time.
- Optional: In the Placeholder field, specify a hint text for the variable; for example, Select a date.
- Optional: In the **Minimum** field, set the bottom (inclusive) limit of a date or date-and-time range for the value in the Properties pane.
- 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.
- 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:

- In the **Design Time** tab, select the **Subtype** as Endpoint.
- 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.
- 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.
- Optional: Switch to the **General** tab, then in the Label field, enter a user-friendly name for the variable.

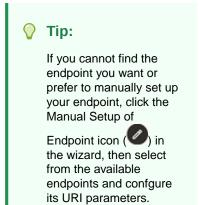


Clicking Select launches a Configure Endpoint wizard in which fragment users can select a suitable endpoint and choose its URI parameters.



Note:

The Placeholder field does not take effect in the Properties pane when you use the Endpoint subtype.



Customiz Steps in Design Time Tab ation Option

Result in Page Designer

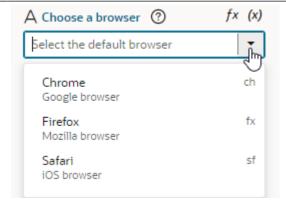
To display a dropdown menu containing an array of possible values:

For a string-type variable or constant:

- In the Design Time tab, select the Subtype as Enum Values.
- 2. Optional: In the **Placeholder** field, specify a hint text for the variable; for example, Select the default browser.
- 3. 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.

- 4. Repeat step c to create your entire list of values.
- Optional: Switch to the **General** tab, then in the **Label** field, enter a user-friendly name for the variable.





ation Option

Customiz Steps in Design Time Tab

Result in Page Designer

To display a dropdown list of values retrieved from a service, such as a REST endpoint:

For a string-type variable or constant:

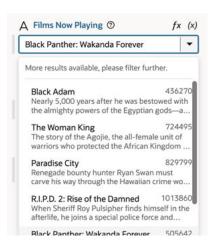
- In the **Design Time** tab, select the **Subtype** as LOV.
- 2. Optional: In the Placeholder field, specify a hint text for the variable; for example, Select an option from the list.
- Click Select Endpoint and choose the endpoint whose response is used to provide the list of values. The REST endpoint must already be set up and available to your App UI. The endpoint response must also be in JSON format.
- When the REST call requires input and URI parameters, click the associated Assign or Not Mapped link and map the parameters to the value they should be replaced with. The value can be a VB Studio constant.

If the path and URI parameters depend on other constants (for example, your REST service uses the result of an earlier selection as part of its own request), use expression notation in the parameter values to indicate which constant values to use. You'll need to do this in the source JSON. For example, you would use the following expression to instruct a service request to use the current value of the "dept" constant as the value for the path parameter "department":

```
"pathParameters": {
  "department":
"[[ $constants.dept ]]"
```

When writing the expression:

- Use simple direct references. Calculated expressions such as "[[\$constants.dept + " "]]" do not work as expected.
- Use constants; variables cannot be used.
- Make sure the referenced constants are accessible to the extension making the LOV service call.
- If your endpoint response is not an array, in the Items Path, specify a dot-separated path from the root of the response object to the array containing the list of values. If your response is an array, you can leave this field empty.





Customiz Steps in Design Time Tab ation Option

Result in Page Designer

For example, you would enter results when the response is not an array, as shown here:

```
"now": {
     "type": "string",
     "description": "wow",
     "defaultValue": "505642",
     "input": "none",
     "@dt": {
       "label": "Films Now Playing",
       "subtype": "lov",
       "valueOptions": {
         "service": {
           "request": {
             "endpoint":
"site myappuiExtension:movies/
getNowPlaying",
              "uriParameters": {
                  "api key":
"4174b7d9a7b4bf87342c98e2289c6ee6"
      }
           },
           "response": {
              "itemsPath": "results",
              "mapping": {
                "value": "id",
                "label": "title",
                "description":
"overview"
         }
       }
}
```

- 6. In the **Value** field, specify the field from the response object to use as the actual value of the variable or constant, for example, id.
- 7. Optional: In the Label field, enter the field from the response object to use as the primary display name of the item, for example, title. It can also be an array of field names, for example, ["firstName", "lastName"].
- 8. Optional: In the **Description** field, enter the field from the response object to use as the item description, for example, overview.



Customiz Steps in Design Time Tab ation Option

Result in Page Designer

To display a time picker:

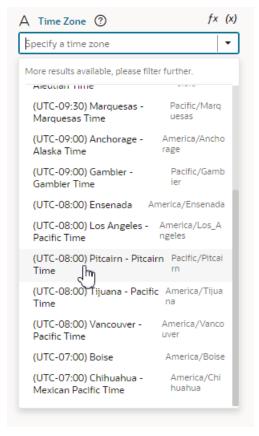
For a string-type variable or constant:

- In the Design Time tab, select the Subtype as Time.
- Optional: In the Placeholder field, specify a hint text for the variable; for example, Select a time.
- Optional: In the **Minimum** property, set the bottom (inclusive) limit of a time range for the value in the Properties pane.
- Optional: In the **Maximum** property, set the top (inclusive) limit of a time range for the value in the Properties pane.
- Optional: Switch to the **General** tab, then in the Label field, enter a user-friendly name for the variable.



To display a dropdown menu with a list of time zones: For a string-type variable or constant:

- In the Design Time tab, select the Subtype as Time Zone.
- Optional: In the Placeholder field, specify a hint text for the variable; for example, Select a time zone.
- Optional: Switch to the General tab, then in the Label field, enter a user-friendly name for the variable.

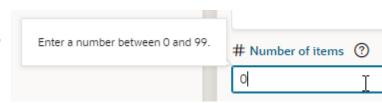


Customiz Steps in Design Time Tab ation Option

Result in Page Designer

To limit the For a number-type variable or constant: input values to a number in a range:

- In the **Design Time** tab, specify a hint text for the variable, for example, Enter Quantity, in the Placeholder field.
- 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.
- 3. 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 JSON file is automatically updated. You can open the JSON editor to view the metadata. For example, here's what you might see for a constant that is customized to use a date picker:

```
"constants": {
    "hireDate": {
      "type": "string",
   "@dt": {
        "subtype": "date",
        "label": "Date of Hire",
        "valueOptions": {
          "placeholder": "Select a date"
      }
   }
},
```

Customize the JSON with Metadata

While you can use a constant's Design Time tab for some customization options, you'll need to edit the JSON directly for advanced options. To do this:

- Open the 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:

```
"constants": {
   "relatedObject": {
      "type": "string",
      "@dt": {
         "subtype": "businessObject",
         "label": "Related Object"
       },
```



```
"description": "Description of related object"
}
```

Use the label property to change the constant's display name in the Properties pane.

Property Options for Variable Metadata

The following table describes the metadata properties that can be used to customize how variables are displayed in the Properties pane.

Property	Туре	Description	
label	string	Use this property to specify a user-friendly name for the variable.	
subtype	Available subtypes: "businessObject" "color" "date" "date-time" "endpoint" "enum"	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.	
	"lov""time""timezone"	A Date of Hire ②	fx
		Select a date	
		March 2024	
		S M T W T	F
			1
		3 4 5 6 7	8
		10 11 12 13 14	15
		17 18 19 20 21	22
		24 25 26 27 28	29
		31	
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.

16

23

30

Properties for valueOptions

Depending on the subtype you select for the variable, you can use the following properties to further customize the components used for editing fields in the Page Designer.



Property

Description

Usage

fields

Use this 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. Property type: array

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.



This property is only supported for displaying the first level of object fields.

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:

The fields property can be used to customize how the object is displayed:

```
"variables": {
    "employee": {
        "type": "person",
        "input": "fromCaller",
        "defaultValue": {
```



Property Description Usage

```
"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 employeed active?",
                 "label": "Active"
            ]
          }
        }
```

The customized object would look similar to this in the Properties pane:



Property	Description	Usage
maximum	Use this property to specify the inclusive top limit of a range when specifying a value in the Properties pane.	Property type: number string This property is suitable for numbers and date/time formats.
		An example of values for the maximum property:
		<pre>{ "maximum": "10", "minimum": "0" }</pre>
		For further details about the JET 'maximum' metadata property, see JET component metadata properties in the Oracle JET JavaScript Extension Toolkit documentation.
		(Recommended to use the Design Time tab for configuration.)
minimum	Use this property to specify the	Property type: number string
	inclusive bottom limit of a range when specifying a value in the	This property is suitable for numbers and date/time formats.
	Properties pane.	An example of values for the minimum property:
		{ "maximum": "10", "minimum": "0"
		}
		For further details about the JET 'minimum' metadata property, see JET component metadata properties in the Oracle JET JavaScript Extension Toolkit documentation.
		(Recommended to use the Design Time tab for configuration.)
placeholder	Use this property to specify a user-friendly hint text.	Property type: string
		This property can be used in Properties pane customizers that have placeholder support, for example, JET Input type customizers.
		An example of configuring a value for the placeholder property:
		<pre>"placeholder": "Search" </pre>
		If a default value is supplied by the constant, 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.

Property	Description	Usage
service	Use this property to fetch possible values from a service for an List of Values (LOV) picker.	Property type: object When using the lov subtype to display a dropdown list of values, you use service to retrieve the values from a service, such as a REST endpoint, and configure additional properties beyond those available in the Design Time tab.
		With the service endpoint already set up in VB Studio and available to the App UI, 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.
		(Recommended to use the Design Time tab for configuration.)
translatabl e	Use this to specify if translation helpers should be present for the property.	Property type: boolean Translation is only available for string types. An example of configuring a value for the translatable property: } "translatable": true }



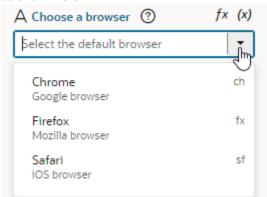
Property	Description	Usage
values	Use this property to specify an array of possible values for the field.	Property type: object
		This property can be used with the enum subtype.
		Values must include value, and can optionally include label and description.
		An example of using the values property:
		"subtype": "enum",
		"valueOptions": {
		"values": [
		{
		"value": "ch",
		"label": "Chrome",
		"description": "Google Browser"
		},{
		"value": "fx",
		"label": "Firefox",
		"description": "Mozilla Browser"
		},{
		"id": "sf",

}]

The example above displays in the Properties pane as shown here:

"description": "Apple Browser"

"label": "Safari",



For further details about the JET 'enumValues' metadata property, see JET component metadata properties in the Oracle JET JavaScript Extension Toolkit documentation.

Organize How Constants Are Listed in the Properties Pane

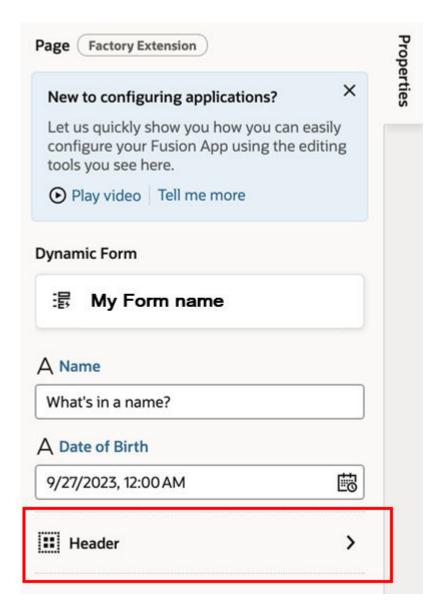
To help users who will be extending pages, you can customize how constants and components are organized and displayed in the Page Designer's Properties pane.

Typically, when extending a page in the Page Designer, the Properties pane contains two tabs—the Constants tab and the Components tab—where the page's extendable items are listed. Extendable items, which includes constants, dynamic components, and extendable items in

embedded fragments, are listed under the appropriate tab. To provide a better design experience for users extending the page, you can organize how items are listed in the Properties pane by organizing items into folders, and by setting the order that they are listed.

You can also choose to hide any items you don't want listed in the Properties pane (they'll still be listed in the Variables and Rule Sets editors). This is particularly helpful when there are many extendable components and constants. If you decide not to organize any of the items, the items are listed in the Constants and Components tabs.

Here's an example of the Properties pane for a page with an extendable dynamic form and constants. The Properties pane is optimized so that the constants that are typically configured are visible directly in the pane. The page constants for configuring the header, which are used less frequently, are grouped under the Header section to be less distracting:



To customize how items are displayed in the Properties pane:

Open the page's Design Time editor.

The page's extendable items (including dynamic components from embedded fragments) are listed in the Components pane. By default, the items in the General pane are listed in

the order they appear in the page, but you can change the order to organize how items are grouped and listed.

2. In the General pane, change the order items and sections are listed by grabbing the handle to the left of the item and moving it into the position you want:



- 3. Create a section and add items to it:
 - Click + Section, enter a section label in the pop-up (for example, Header), and click Create.
 - b. Optionally, in the newly created section's 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**.

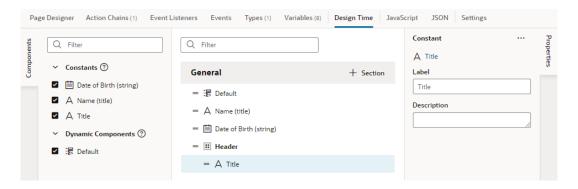
You can also associate the section with a page component. When the user selects the component in the Page Designer, the Properties pane displays the items in the associated section.

To associate a component with a section:

- Open the Page Designer and select the component on the page.
- ii. In the Properties pane, select the section in the Design Time Section dropdown list:



c. Now drag the items you want to add to the section and drop them onto the section header (for example, drag the Title constant and drop it onto the Header section).



To remove an item from a section, right-click the item 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**.

4. Under General, select each item to edit its properties in the Properties pane.

Modify the Label and Description properties to customize the label and descrption the user will see in the Properties pane when configuring the page.



Now when the user configures the page in an extension, they'll see the items in the Properties pane displayed in the order you want them to appear, and grouped how you would like.

You can also customize how fields for editing constants are rendered in the Properties pane. See Configure How Variables are Customized in the Properties Pane.

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

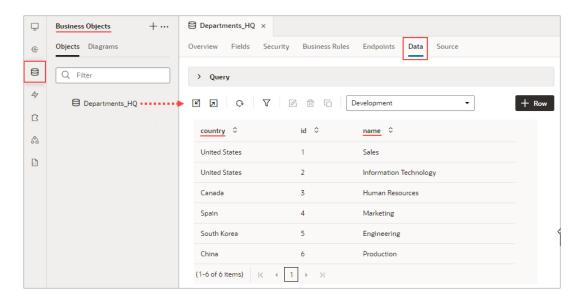
Let's go through an example of creating a custom fetch action chain for an SDP. In this example, we'll add a Single Select component to a page and bind it to an SDP in order to list the department headquarters from a business object. We'll then configure the Single Select component to show additional 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:

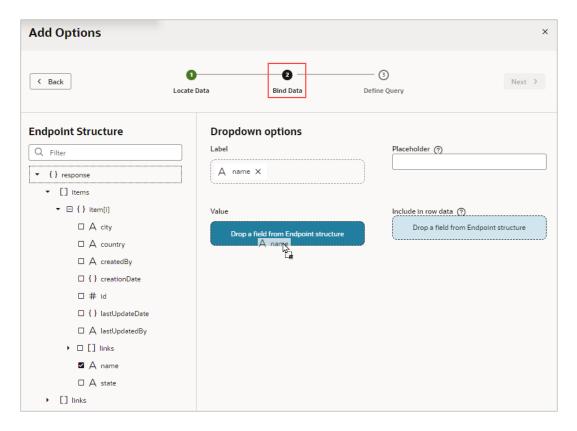
- Select the Business Object ≡ tab, then click + at the top right to create a new business object called Departments HQ.
- On the Departments HQ tab, select the Fields tab and add a name (String) and a country (String) field.
- 3. On the **Data** tab, add a few rows for the Select Single component to list:





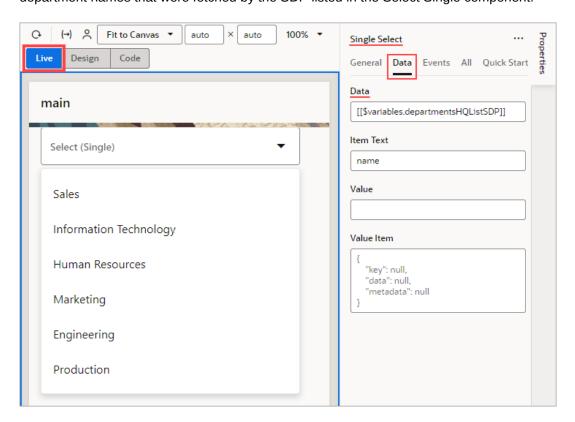
Next, we'll add a Select Single component to a page, and then use its Add Options Quick Start to map it to the Departments HQ business object.

- Create a new page, then in the Page Designer tab, add a Select (Single) component to the page.
- To map the component to the Department HQ business object, in 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 in the Properties pane's Data tab. If you switch the Page Designer to Live view, 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:

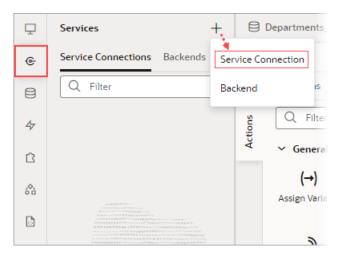
```
{"headerText":"", "field": "countryFlag", "template": "flagTemplate",
"id":"countryFlag"}
          ] '
          class="oj-select-results"
          data="[[collection.data]]"
          selected.row="[[collection.selected]]"
          on-oj-row-action="[[collection.handleRowAction]]">
          <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">
              <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>
```



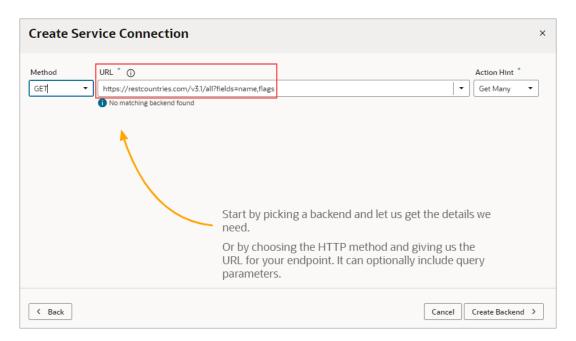
```
Page Designer
            Action Chains Event Listeners (1) Events Types (1) Variables (2) JavaScript JSON Settings
                                                                                                 Live Design
                                                                                                                Code
              <!-- Add Page specific actions etc -->
     37
            </div>
    38
          </div>
     39
          <div class="oj-flex">
            <oj-select-single label hint="Select (Single)" class="oj-flex-item oj-sm-12 oj-md-6"
   data="[[$variables.departmentsHQListSDP]]" item-text="name">
    40
    41
     42
     43
                 <template slot="collectionTemplate" data-oj-as="collection">
                 →≺oj-table
  Add Table
                     accessibility.row-header="[[['department', 'country']]]"
   component
                     horizontal-grid-visible="disabled"
    47
                     vertical-grid-visible="disabled"
    48
                     selection-mode='{"row": "single"}
     49
                     columns-default='{"resizable": "disabled",
     50
                                      "sortable": "disabled"}'
     51
                    columns='[
                     class="oj-select-results"
                    data="[[collection.data]]"
                     selected.row="[[collection.selected]]"
     58
                    on-oj-row-action="[[collection.handleRowAction]]">
     60
     61
                     <template slot="departmentTemplate" data-oj-as="cell">
    62
                      <span>
     63
                          <oj-bind-text value='[[cell.data]]'></oj-bind-text>,
    64
                      </span>
                                                                                   Specify components for displaying columns
    65
                     </template>
                    <template slot="countryTemplate" data-oj-as="cell">
    66
    67
    68
                        <oj-bind-text value='[[cell.data]]'></oj-bind-text>
    69
                      </span>
     70
                    </template>
                    <template slot="flagTemplate" data-oj-as="cell"</pre>
    71
     72
                     <oj-avatar src='[[cell.data]]'></oj-avatar>*
     73
                    </template>
     74
     75
                  </oi-table>
     76
                 </template>
     77
               </oi-select-single>
          </div>
     78
```

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**:



 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:



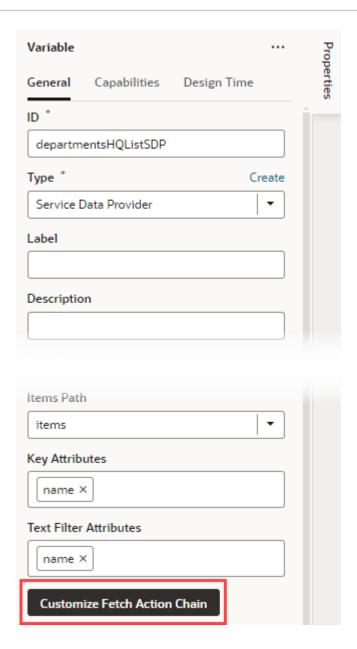
A backend is created for this service connection, which stores the server details. You can use this backend to create related service connections, and to apply endpoint requests and response transform functions to them all.

- 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:

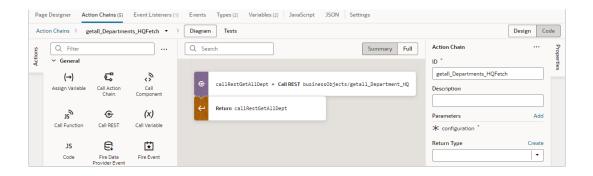
- 1. Go to the page's **Variables** tab and select the SDP that's bound to the Select Single component.
- In the Properties pane, scroll down to the bottom and click Customize Fetch Action Chain:



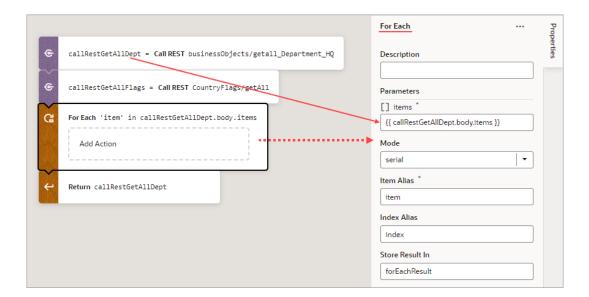
You're taken to the Action Chains editor, where the SDP's fetch action chain is loaded, which has an auto-generated name and a preconfigured Call REST action.



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 }}:

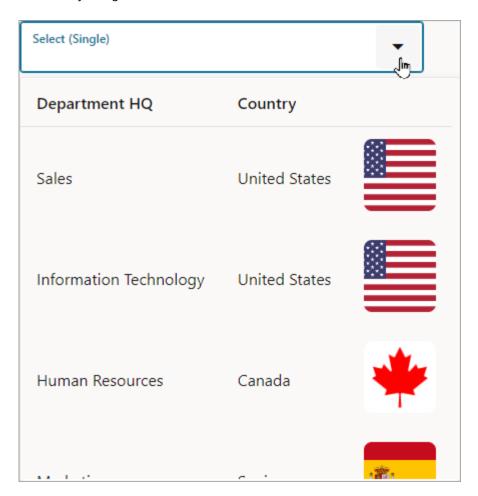


- 7. Add a JS Code action to the **Add Action** area of the For Each action.
- 8. In the Properties pane, replace the text in the Code box with this code to add the flag image to the data that's to be returned by the action chain: item.countryFlag = callRestGetAllFlags.body.find(country => country.name.common === 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.

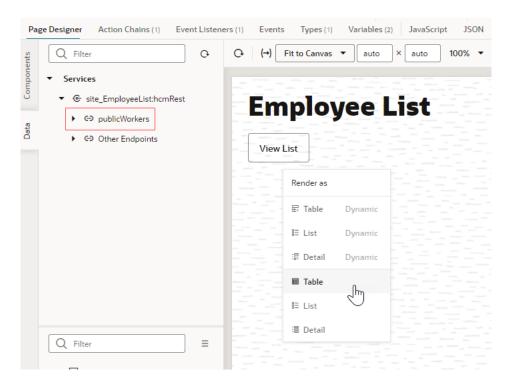
An SDP automatically executes the REST point that it's bound to when the associated UI component is first shown on the page, so if we want to delay the execution of the SDP call, we need to hide the UI component. For example, you can delay display of table data on a page until the user clicks a button.

In this procedure, we use an <code>oj-if</code> component to hide a table, then we add an <code>ojAction</code> event to a button on the page. When the button is clicked, the variable controlling the <code>oj-if</code> component is updated, and the REST call is executed to fetch and display the data in the table. For more information on the <code>oj-if</code> 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.

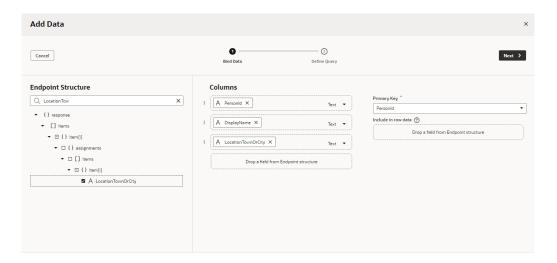
In this example, we've set up a service connection using the **Create Service Connection** wizard to create a Human Capital Management service connection from the catalog and chose the **publicWorkers** object. For more information, see Create Service Connections from the Oracle Cloud Applications or 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.

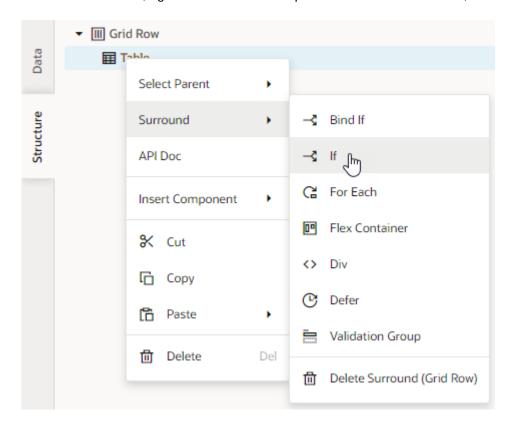




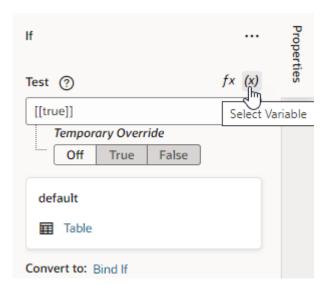
b. In the Bind Data page, search for and select the endpoints that you want to add as table columns (for example PersonID, DisplayName, and LocationTownOrCity). Click Next.



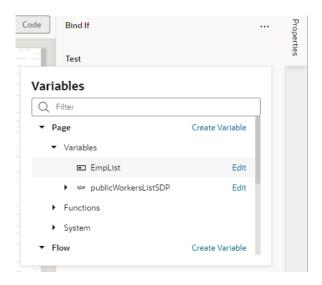
- c. In the **Define Query** page, click **Finish**.
- 3. Create a boolean variable to control the table display.
 - a. In the Variables tab, click + Variable.
 - b. Update the ID to (for example) EmpList and choose Boolean in the Type field.
 - c. Select false as the Default Value.
- 4. Use the oj-if component to hide the table.
 - a. In the Structure view, right-click the Table component and select Surround, then If.



Select the If component in the Structure view. In the Properties pane, hover over the Test field and click (x) to open the Variables picker.

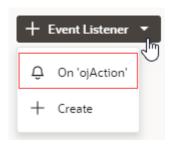


c. Select **EmpList** from the Variables list.



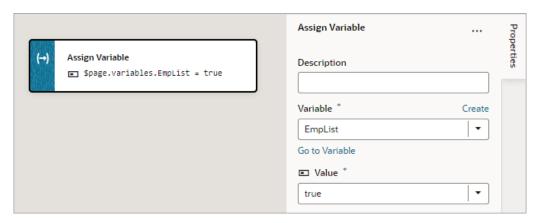
The table is hidden.

- 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.

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.

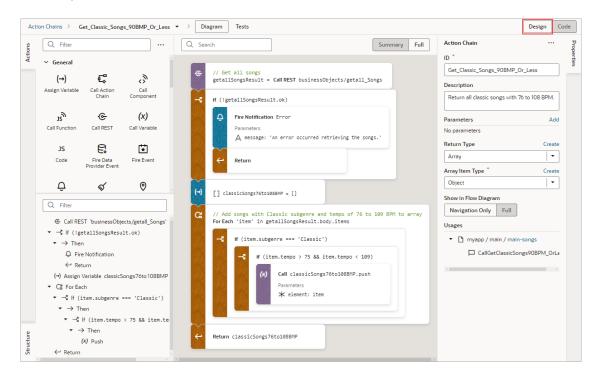


Now when users reach the page, they will need to click the button to view the table.

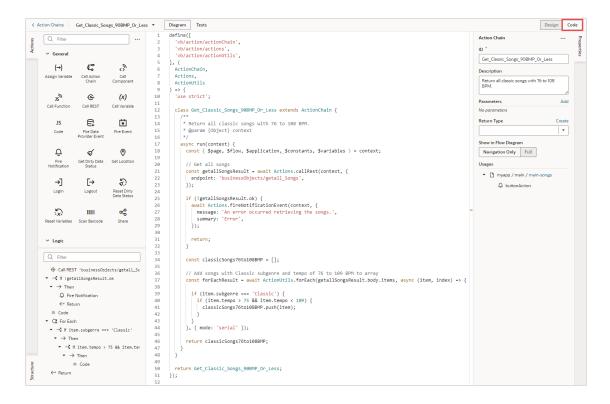
Work With JavaScript Action Chains

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.

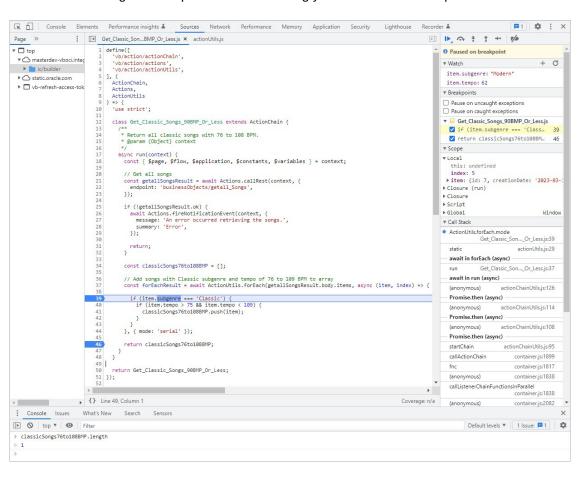
You implement action chains using either the visual Action Chain editor or through code, using JavaScript. Here's an example of an action chain built with the Action Chain editor. The action chain first calls a REST endpoint to get a full set of songs, then checks to see if any errors occurred. As long 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:

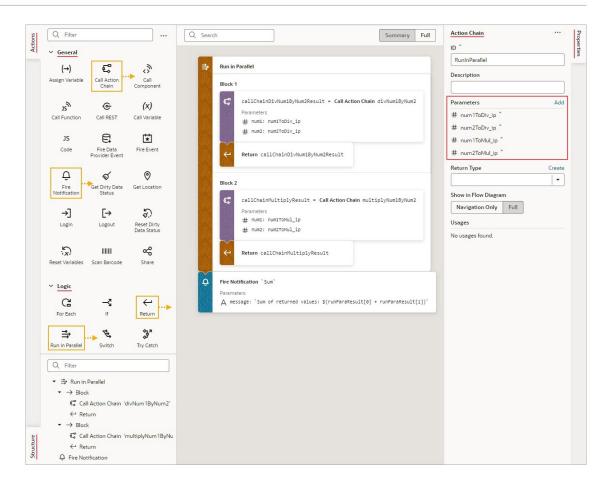
- vbEnter: triggered when a page starts and can be used to fetch data
- ojAction: triggered when a button component is clicked
- onValueChange: triggered when the value stored in a variable changes

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 VB Studio 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 $\mathtt{asyn}()$ method is used to call an action chain that returns the quotient of two numbers, and another $\mathtt{asyn}()$ 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 $\mathtt{asyn}()$ method and the second element containing the value from the second $\mathtt{asyn}()$ 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`,
});
```

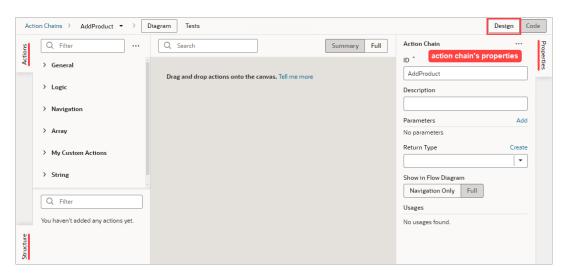
When creating action chains, keep in mind that each action chain has a scope that depends on where it's defined: at the App UI, flow, page, or fragment level. An action chain defined at the App UI level can be called from any flow or page, but a page-level action chain can only be called from that page— however, the chain itself can access variables defined on the page, parent flow, or App UI. 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 view is used to visually create an action chain:



Code view is used to create an action chains with code:



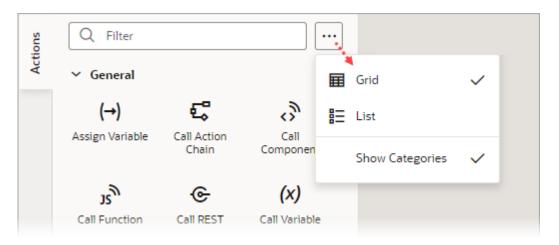
Both views have an Actions palette, a Structure pane, and a Properties pane:

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:

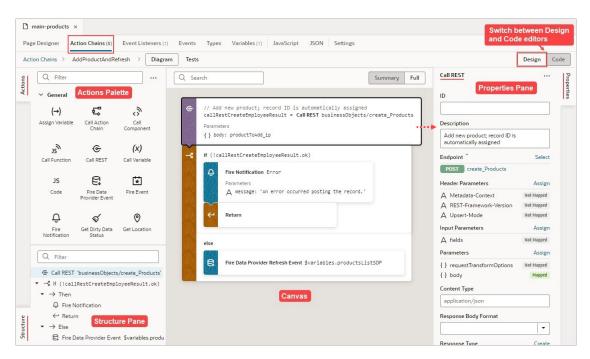
The Structure pane provides a compact view of the actions, and has a filter to quickly find and select an action. Also, syntax errors detected by the JavaScript parser are shown, if they can be handled, otherwise a message states that the Structure pane can't be displayed due to errors.

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 View

When you use the Actions palette, Properties pane, and Structure pane in Design view, VB Studio writes the corresponding JavaScript behind the scenes. You can edit this code directly at any time by switching to Code view, which opens a code editor.

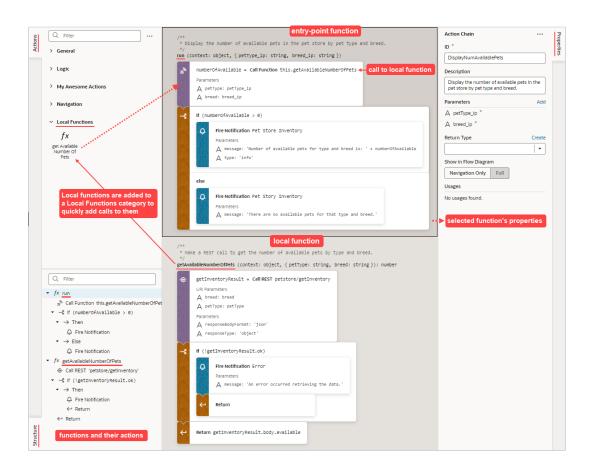


Right-click an action to display a context menu with the following options:

- 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:

- Use the search box at the top of the canvas to find text within your action chain.
- 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:



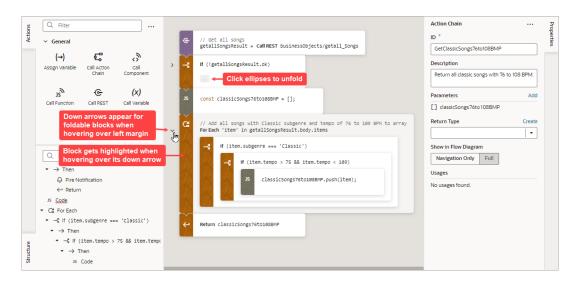
- To convert a local function into an action chain for use in other action chains, right-click the local function and select Convert Function to Chain. The local function and any local functions it uses are converted to an action chain. In the original action chain, the Call Function action that called the local function will be 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 is replaced with a Call Function action that calls the new local function:



 To visually simplify large action chains, you can fold (collapse) 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:





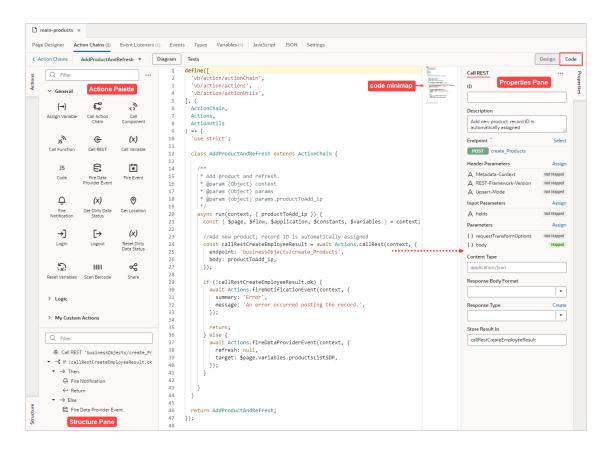
If you prefer to construct your action chain using code, click the **Code** button at the topright of the screen to open a code editor. Typically, one uses Design view to visually add
and configure an action, then switches back to Code view to work directly with the code, as
needed.

Create Action Chains in Code View

You use the Code view 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:

Drop action here to create an Else statement with that action

```
if (language == 'eng') {
    const navToEnglishFlowResult = await Actions.navigateToFlow(context, {
        target: 'parent',
        flow: 'english_flow',
    });
    Prop action here to add it
    to this Else statement
    await Actions.navigateToFlow(context, {
        target: 'parent'
    });
}
```

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:

```
← Action Chains

                   GetClassicSongs76to108BMP ▼
                                                        1 ∨ define([
     Q Filter
                                                                'vb/action/actions'

✓ General

                                                                 'vb/action/actionUtils',
                        Ç
                                       3
         (→)
                                                                ActionChain,
                                                               Actions,
                                    Call
Component
                                                               ActionUtils
                        ©
                                                                'use strict':
         J2
                                      (x)
                                    Call Variable
                                                               class GetClassicSongs76to108BMP extends ActionChain {
                                                                 /**
| * Return all classic songs with 76 to 108 BPM.
                        €
                                      *
                                                      14
                                                                  * @param {Object} context
* @param {Object} params
                    Fire Data
Provider Eve
                                    Fire Event
                                                      17
                                                                   * @param {object[]} params.classicSongs76to108BMP
                                       0
          Ô
                        Q
                                                      19
                                                                  async run(context, { classicSongs76to108BMP = []
                                                       20 🗸 }) {
                                                      21
                                                                    const { Spage, $flow, $application, $constants, $variables } = context;
                                                      22
23
                                      8
         \rightarrow
                        [→
                                                                     // Get all songs
                       Logout
                                                                     const getallSongsResult = await Actions.callRest(context, {
                                                      25
                                                                      endpoint: 'businessObjects/getall_Songs',
                                                      26
27
28 >
     Q Filter
                                                                    if (!getallSongsResult.ok) \{\cdots \leftarrow Click ellipses to un
        @ Call REST 'businessObjects/getall Sc
                                                      36
37
38 ∨
                                                                    // Add all songs with Classic subgenre and tempo of 76 to 109 BPM to classicSongs76to108BMP array const forEachResult = await ActionUtils.forEach(getallSongsResult.body.items, async (item, index) => {

▼ 

If !getallSongsResult.ok
       ▼ → Then
           G Fire Notification
                                                                         f (item.subgenre === 'Classic') {
  if (item.tempo > 75 && item.tempo < 109) {
           ← Return
                                                      41 🗸
      ▼ Ca For Each
                                                                            classicSongs76to108BMP.push(item);
        ▼ 📑 If item.subgenre === 'Classic'
                                                      43
                                                      45
                                                                    }, { mode: 'serial' });
           ▼ - If item.tempo > 75 && item.ter
             	ilde{ullet} 
ightarrow Then
                                                                    return classicSongs76to108BMP;
                                                      48
                 JS Code
                                                      51
52
                                                               return GetClassicSongs76to108BMP;
                                                             });
```

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 <code>settings.json</code> file where you can customize VB Studio's code editors. To disable the code minimap, add this entry to the <code>settings.json</code> 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, VB Studio 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:

```
define([
                  2
                         'vb/action/actionChain',
                  3
                        'vb/action/actions'
                  4
                       ], (
                  5
                        ActionChain,
                  6
                        Actions
                  8
                       'use strict';
                  9
Action chain's
                 10
                        class addNumbers extends ActionChain {
class declaration
                 11
                            * @param {Object} context
                 12
Define input
                 13
                            * @param {Object} params
parameters and
                 14
                            * @param {number} params.num1_inparam
return type
                            * @param {number} params.num2 inparam
                 15
                            * @return {number}
Run() method
                 17
to override
                           async run(context, { num1_inparam, num2_inparam }) {
                 19
                           const { $page, $flow, $application, $constants, $variables } = context;
Runtime context
                 20
                 21
                             return num1_inparam + num2_inparam;
Payload
                 22
returned by
                 23
action chain
                 24
                 25
                        return addNumbers;
                 26
                       });
```

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 <code>module</code> 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></actionname>	Name of action.
Context	The runtime context.
parameters	Action-specific parameters object.
<custom-action-id></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
                                                 entry-point function
  * @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 });
                                                      call to local function
   if (numberOfAvailable > 0) {
     await Actions.fireNotificationEvent(context, {
       summary: 'Pet Store Inventory',
       message: 'The number of available pets for this type and breed is: ' + numberOfAvailable,
     });
   } 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
                                                  local function
  * @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

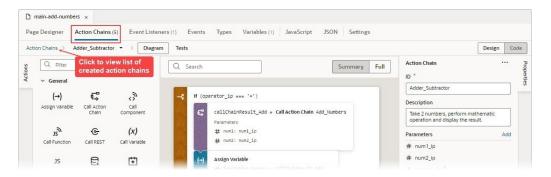
Here, we'll use the Action Chain editor's Design view to assemble built-in actions into a sequence that performs a task. Each action performs a specific function and returns results that can serve as inputs for subsequent actions.

To visually create an action chain:

 Navigate to where you want to initiate the creation of the action chain, depending on your preference and how you want it triggered:

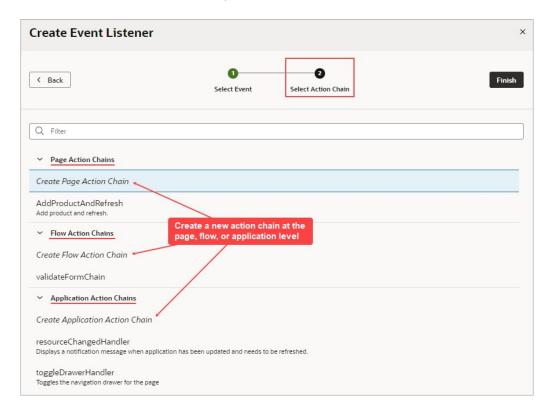
Actions Chains Tab:

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 App UI, 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 App UI) 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:

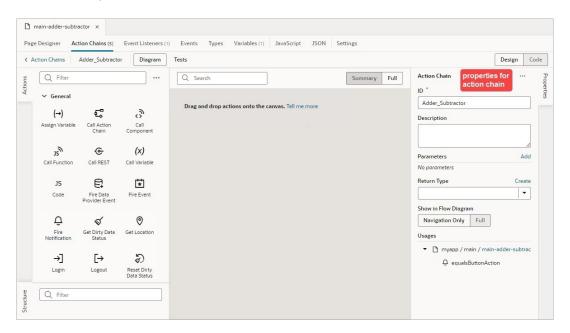
To have your new action chain started by a component event, select the component on the Page Designer's canvas, and in the Properties pane use the **Events** tab to create a new event, event listener and action chain for the component. For further details, see Start an Action Chain From a Component and Start an Action Chain By Firing a Custom Event.

Variable:

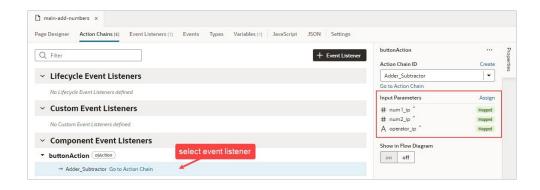
To have an action chain started when a variable's value changes, open the relevant **Variables** tab, at the App UI, flow, or page level, and select the variable. In the Properties pane, select the **Events** tab and click **+ Event Listener**. A new onValueChanged event is automatically created for 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.

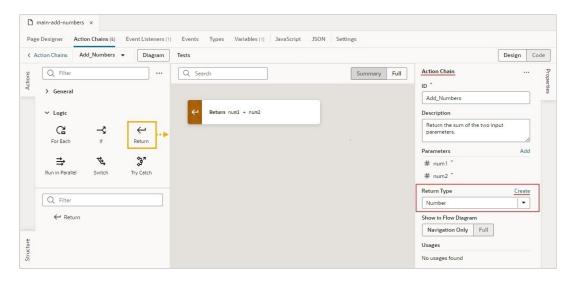
2. Enter a name for the action chain in the **ID** field, and if you like, a description. The new action chain opens in the Action Chain editor:



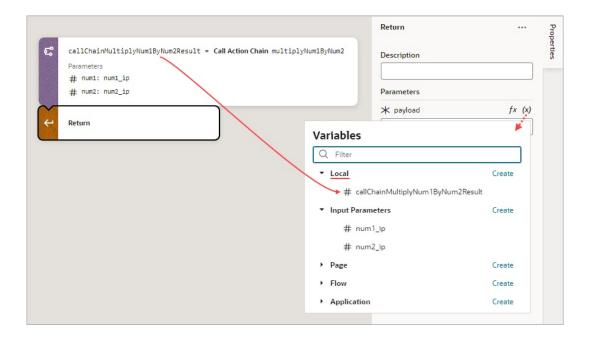
- 3. If your new action chain needs input parameters:
 - a. Define the input parameters using the **Add** link for the Parameters property in the Properties pane.
 - **b.** Pass the input parameter values to the action chain:
 - If your action chain is to be started by another action chain, the input 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 (App UI, 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 (App UI, 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:

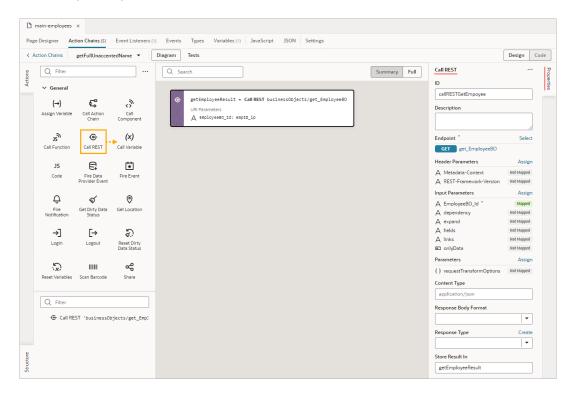


As shown in this example, in which the variable for the Return action to return is selected, the result returned by an action chain is available in the Variables picker, under the **Local** node, :



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:

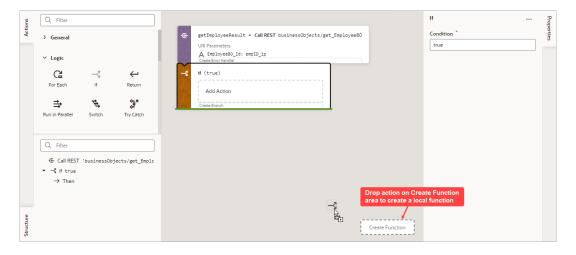


7. Double click or drag and drop the next action from the Actions palette onto the bottom edge of the action that it follows. Configure the action in the Properties pane or through code. To add an action before another action, drop it on the top edge of the action it is to precede.

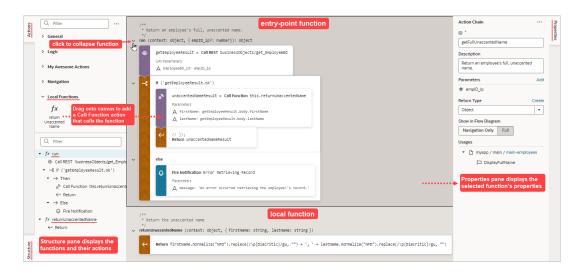


As shown above, the Structure pane displays a compact view of the actions and it provides a filter to quickly find and select an action.

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:

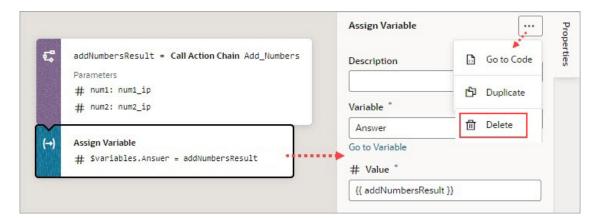


When a local function is created, it gets its own space on the canvas, the Property pane displays its properties, and the Structure pane displays its actions. Also, the local function is added to the Actions palette, under the Local Functions category, for you to quickly add a call to it.

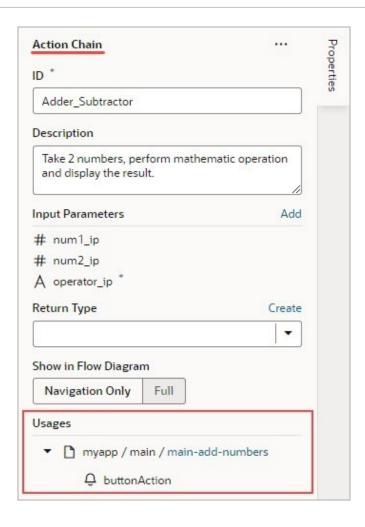


9. Continue to add and configure actions until your action chain is complete. The action chain is saved automatically as you make changes.

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:

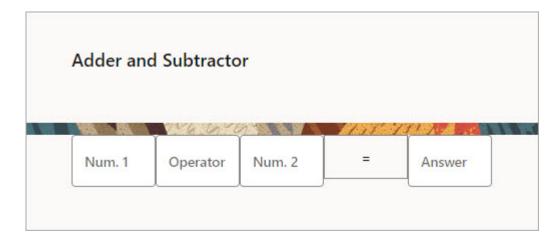


To view usage details for your action chain, such as which pages use it, click an empty space on the canvas to select the action chain and look under Usages in the Properties pane. Click a usage to navigate there. The event listener tied to the event that calls the action chain is also listed, as shown 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:

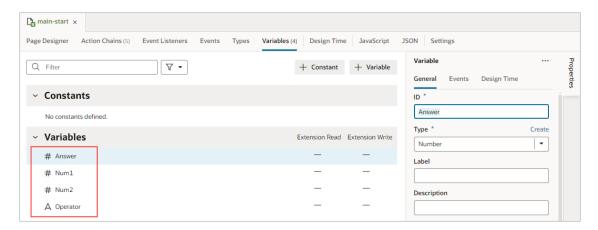


The interface has:

- Four text components:
 - Two for entering the numbers to add or subtract (Num. 1, Num. 2)

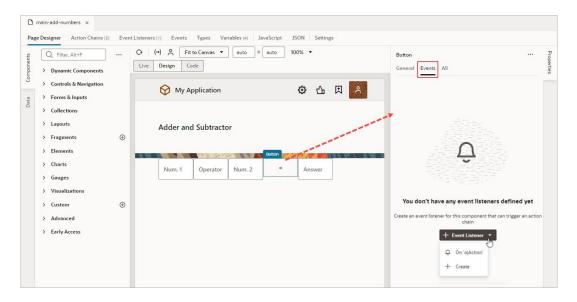
- One for entering either a plus or minus sign (Operator)
- One to display the result (Answer)
- One button (=) that triggers the action chain that performs the operation and displays the result

Each of the four text components is bound to a page variable:

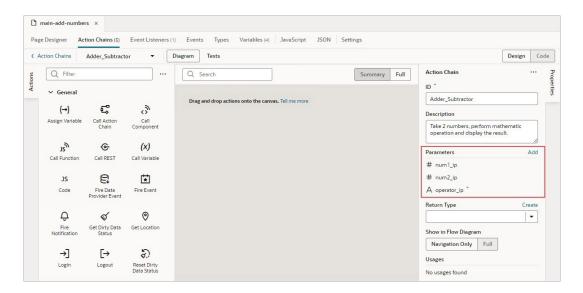


Here, we create the action chain that either adds or subtracts two numbers and displays the result:

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 in the Properties pane:

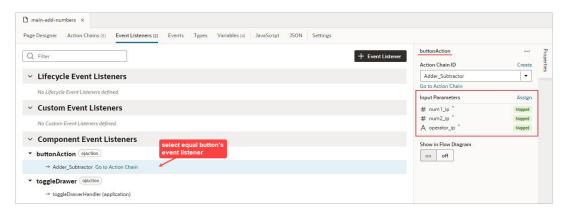


- Click + Event Listener, then On 'ojAction' to create an event that's triggered by clicking the button, as shown above.
 - The new action chain opens in the Action Chain editor.
- 3. Using the Properties pane, enter Adder_Subtractor for the action chain's ID field, and optionally a description.
- 4. 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:

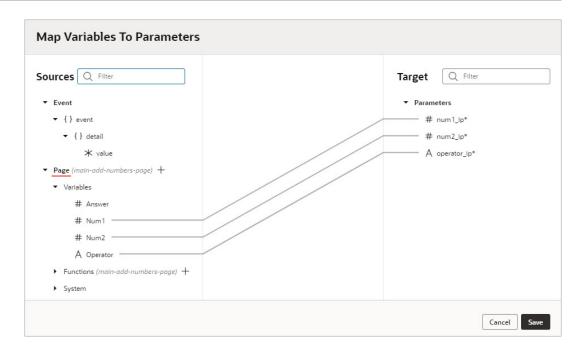


Nothing needs to be returned by this action chain, so we don't need to define a return type.

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:



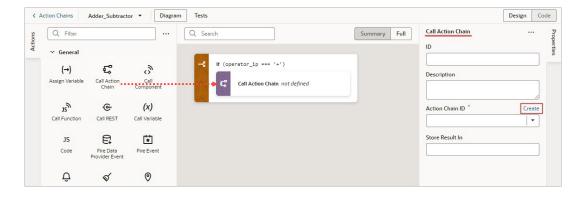
6. In the mapper, map the page variables that were bound to the text components for the numbers and the operator to the action chain's input parameters:



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.

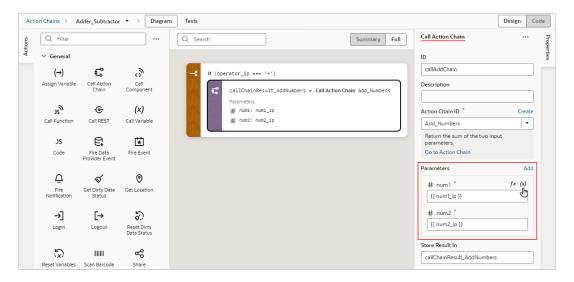


- 10. To create the action chain that adds two numbers, in the Properties pane, click the Create link for the Action Chain ID property (shown above). In the dialog box, enter Add_Numbers in the ID field, and optionally, enter a description. Click the Create button. The action chain has been created and 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.

- 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 ${\tt Else}\ {\tt If}\ condition\ to\ handle\ the\ case\ when\ a\ user\ enters\ a\ minus\ sign$ for the operator.

16. Drag and drop the **Call Action Chain** action onto the **Create branch** area at the bottom of the If condition:



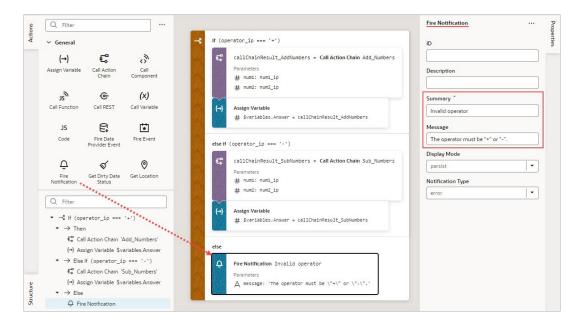
- 17. Change the Else condition into an Else If by entering operator_ip === '-' for the Condition field, in the Properties pane.
- **18.** Complete this Else If condition by following the previous instructions on how to handle the plus sign case.

Here's the completed ${\tt Else}\ {\tt If}\ condition:$



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.

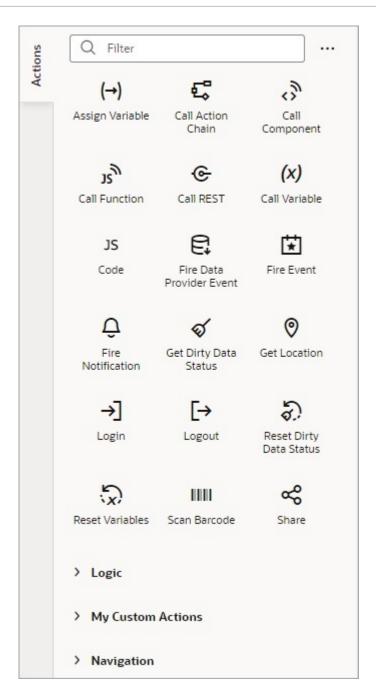
Here's the completed code for the action chain:

```
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,
          },
```

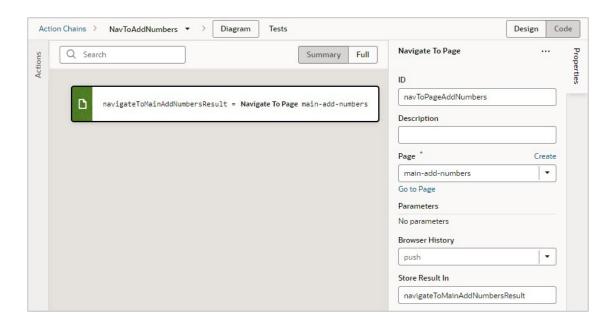
```
}, { id: 'callAddChain' });
        $variables.Answer = callChainResult AddNumbers;
      } else if (operator_ip === '-') {
        const callChainResult SubNumbers = await Actions.callChain(context, {
         chain: 'Sub Numbers',
         params: {
            num1: num1_ip,
            num2: 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

VB Studio 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.

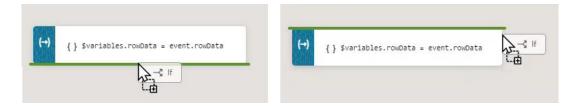


Each action performs a specific function and requires you to set different properties. For example, when you add the Call REST action to your action chain, you need to specify the endpoint and other details about the response to the Call REST action. Similarly, when you add a Navigate To Page action, you'll need to select a page 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.

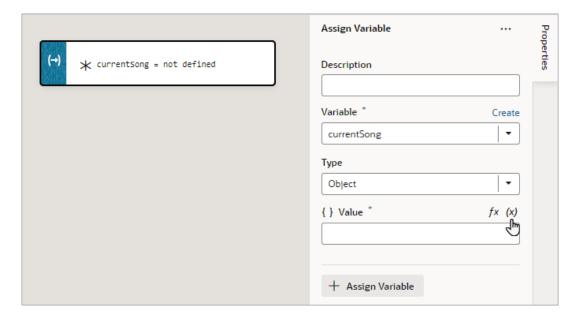
Add an Assign Variable Action

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:

- 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.





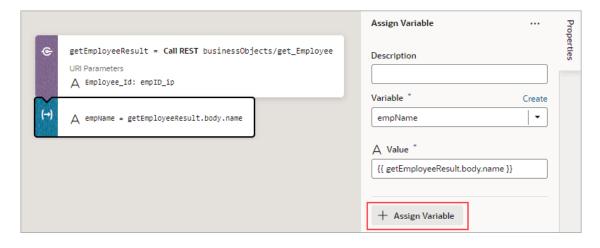
To use an Assign Variable action for a value assignment:

- 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 (x) to choose the variable that holds the value, or click fx to create an expression for the value.

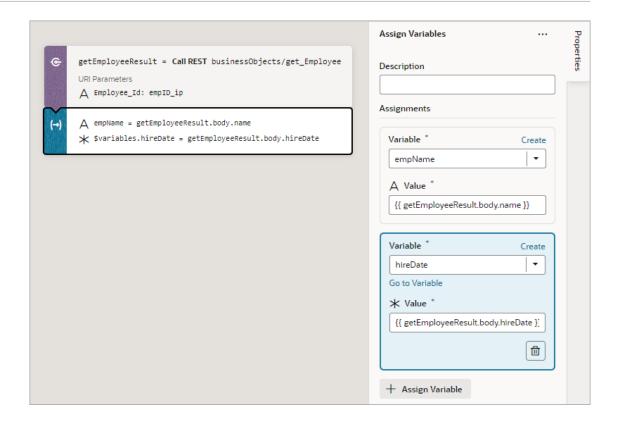
In this example, the page variable fullName is assigned the result from a module function:



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:



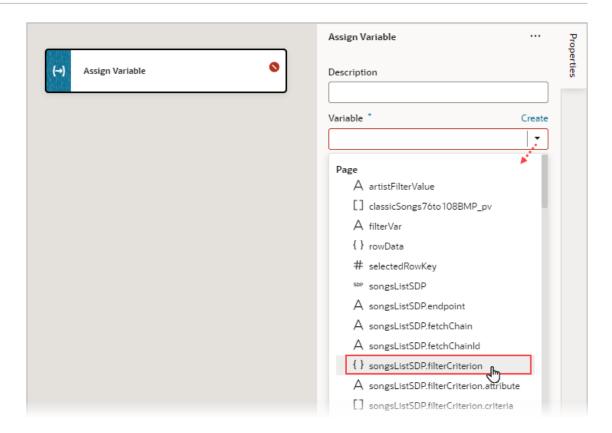
If you'd like to move a variable assignment to a different position, in the Properties pane, click and hold an assignment, then move it to its new position.

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.

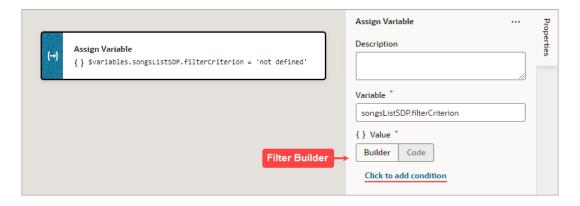
When the Assign Variable action's Variable property is set to an SDP's filterCriterion property, the Filter Builder appears under the Variable property for you to create the filter criterion. To directly work with the code, click the **Code** button. For details, see Filter Builder's Code Editor.





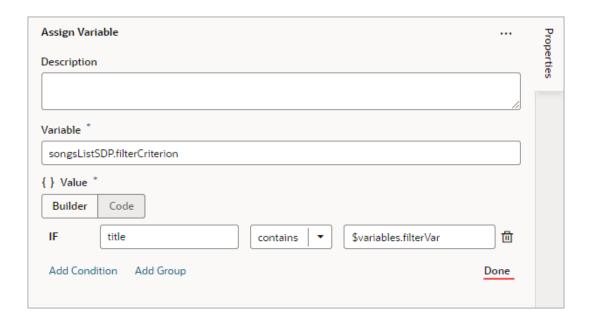
To use the Assign Variable action's Filter Builder to create the filter criterion for an SDP:

1. Click the Filter Builder's Click to add condition link:

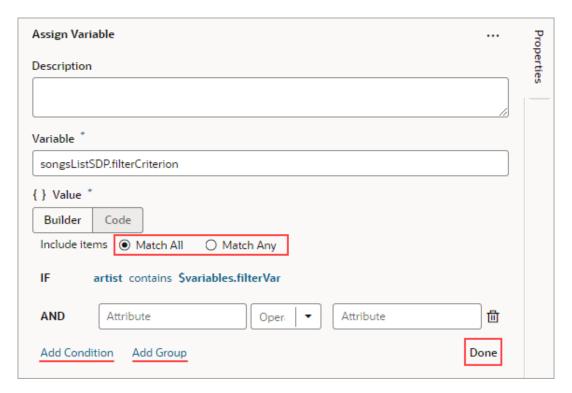


- 2. For the first Attribute textbox, enter the name of the column (field in record, like "city") that you want compared against a specific value (like "Tokyo").
- **3.** For the Operator list, select the operator for the criterion.
- 4. For the second Attribute textbox, enter the specific value to compare 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",
            "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

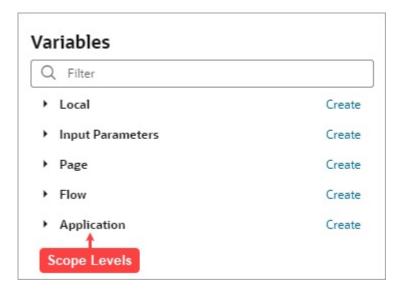
You add a Call Action Chain action to start an action chain. This action can call action chains defined in the same page, flow, or App UI.

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:

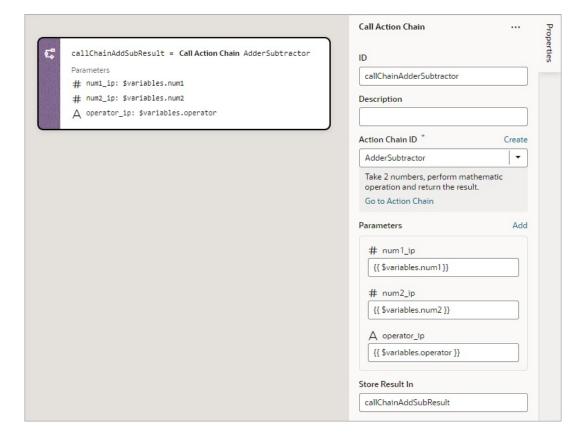
- 1. Add the action in one of three ways, as explained at the end of Built-In Actions.
- 2. Select an existing action chain from the Action Chain ID list, or click Create to create a new action chain. You'll only be able to select or create an action chain within your current scope or any higher scope. For example, if you're at the page level, you can select or create an action chain at the page, flow, or App UI level. However, if you're at the App UI level, you can only do so at the App UI level.
- 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 (x) 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.

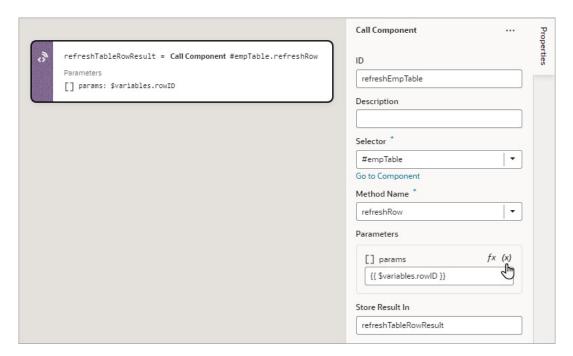
For API information about this action, see Call Component Method in the *Oracle Visual Builder Page Model Reference*.

To use a Call Component action:

- 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:



 If the method requires input parameters, hover over the far-right side of the parameters under the Parameters section and click (x) 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

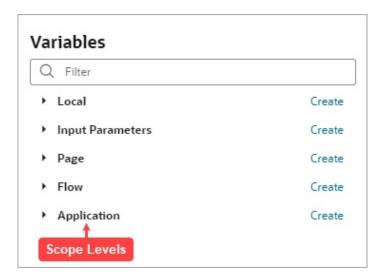
You add a Call Function action to call a function defined for the current page, flow, or App UI, and you can also call a global function. For more about global functions, see Add JavaScript Modules As Global Functions. Functions for a page, flow and App UI are created using their JavaScript editor.

For API information about this action, see Call Function in the *Oracle Visual Builder Page Model Reference*.

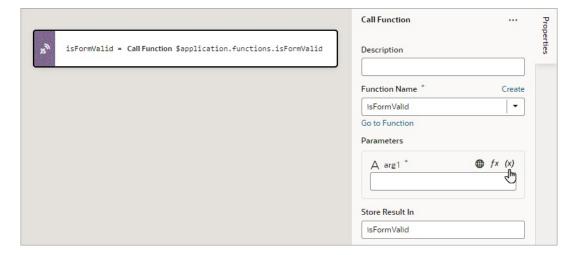


To use a Call Function action:

- 1. Add the action in one of three ways, as explained at the end of Built-In Actions.
- 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 App UI.
- 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 (x) 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*.

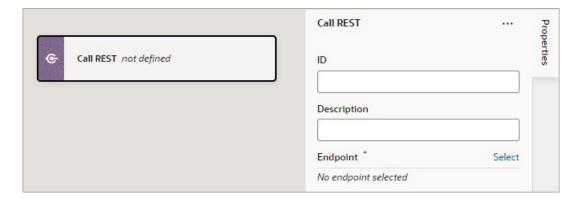
After you add a Call REST action, you need to specify the endpoint for the request. Depending on the endpoint, you might also need to provide input parameters, such as an ID to identify a record.

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:

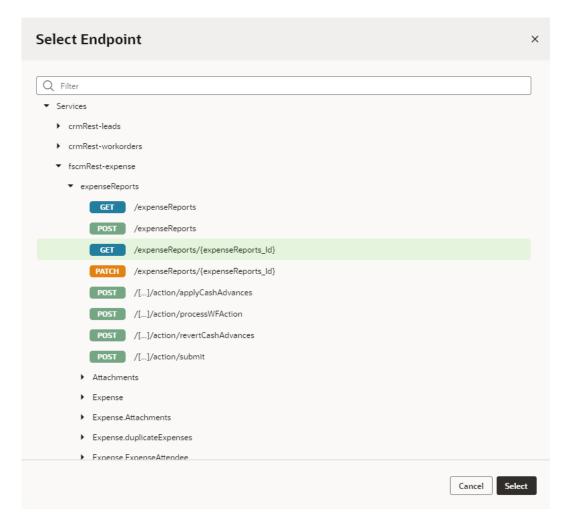
1. Add the action in one of three ways, as explained at the end of Built-In Actions.





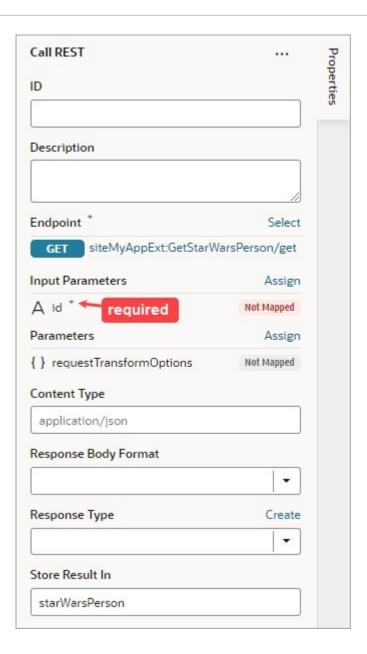
Click Select beside Endpoint in the Properties pane.

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.



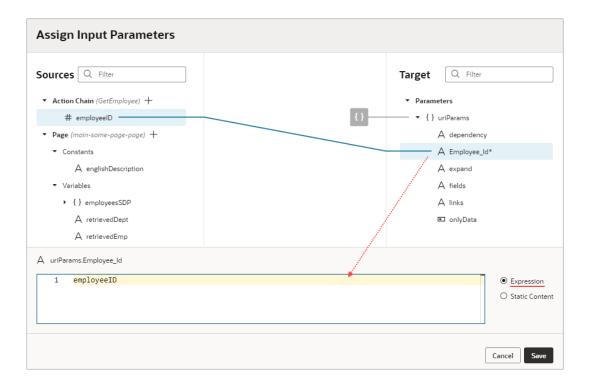
Select an endpoint from the list and click Select.

The properties for the REST call are displayed in the Properties pane, with required properties marked with an asterisk (*):



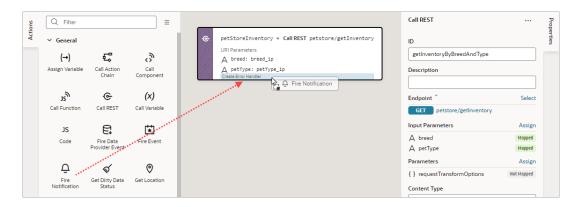
 If the REST call requires header or input parameters, click the associated Assign or Not Mapped link and use the assignment window to specify the value for the property. Click Save.

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.

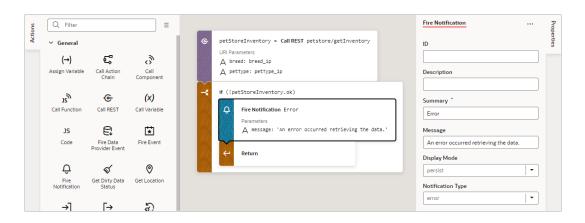


If a suitable variable does not exist, use the + icon beside the relevant node (Action Chain, Page, and so on) to create a new variable.

- 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 App UI). 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.

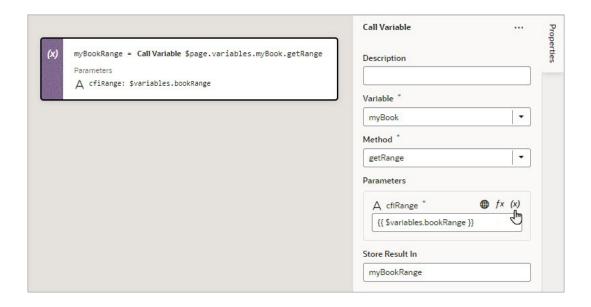
Before you use a Call Variable action, make sure an InstanceFactory type variable is already defined for the App UI. See Create a Type From Code.

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 App UI.
- 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 (x) 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:

```
Code

** tenPercentDiff = not defined

** tenPercentDiff = not defined

** // Compare elements [i] and [i+1] to check if difference is 10% or greater for (let i = 0; i < (tempesAscending.length - 1); i++) {

// Check if temperature difference is 10% or greater if (tempesAscending.length - 1); i++) {

// Check if temperature difference is 10% or greater if (tempesAscending.length - 1); i++) {

// Add the two objects to an array tenPercentDiff.push([tempesAscending[i], tempesAscending.length - 1); i++) {

// Add the two objects to an array cities10percentDiff.push([tempesAscending.length]); is cities10percentDiff.push([tempesAscending.length]); is cities10percentDiff.push([tempesAscending.length]); is cities10percentDiff.push([tempesAscending.length]]); is cities10percentDiff.push([tempesAscending.length]
```

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 specifies which changes to show.



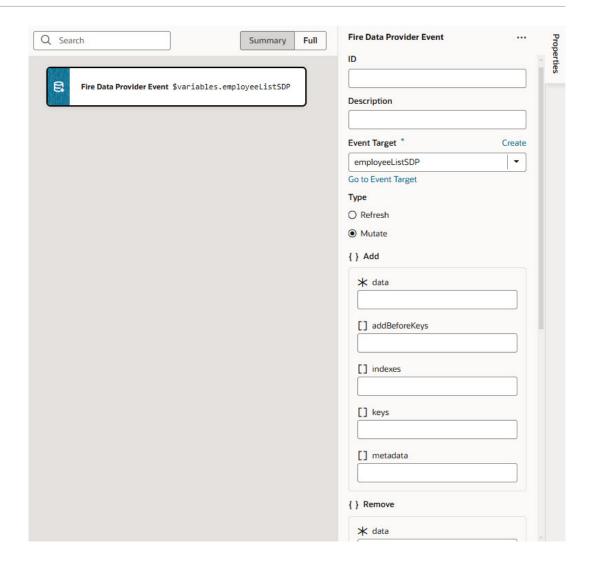
This action is not necessary for an Array Data Provider (ADP) variable, since the data array of an ADP variable, exposed via the data property, can be updated directly using the Assign Variable action. Assigning the data array is automatically detected by VB Studio, and all listeners are notified of this change. Users will be warned of this when the fireDataProviderEvent is used with an ADP, before 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. From the **Event Target** list, select an existing variable based on a data provider (like ServiceDataProvider) as the target of the event. If necessary, click **Create** to create a new data provider variable as the event's target.





- 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. For the mutate event to perform optimally, here's what's required for each section:
 - Add: 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>>.
For example:

```
data: {items: [callRestCreateEmployeeResult.body],},
keys: [callRestCreateEmployeeResult.body.id],
metadata: [{key: callRestCreateEmployeeResult.body.id,}],
```

- Remove: Use the keys parameter to pass the key values of the deleted records with the Set<*> format.
- Update: Same as add.

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 App UI, 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.

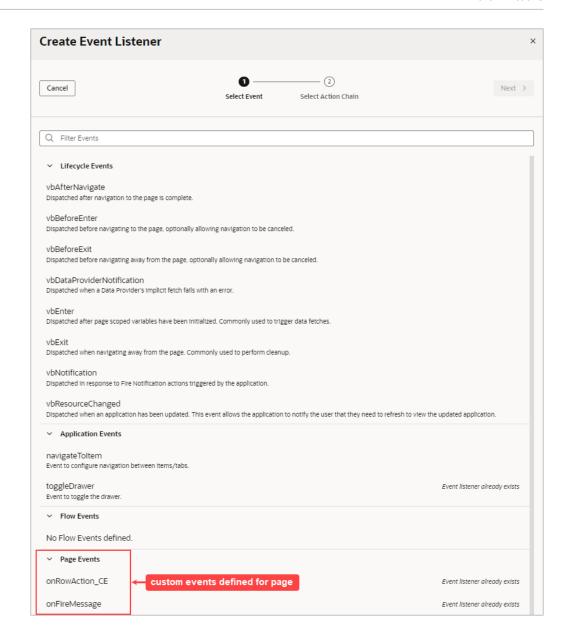
You can trigger a custom event by using a Fire Event action in an action chain, which can be started in several ways (see Start an Action Chain). You could also trigger a custom event by using an event helper's fireCustomEvent() method (see Module Function Event Helper) in a module function (JavaScript 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:

- Create a custom event, defining parameters if required.
- 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 App UI). 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 (x) to choose the

Fire Event onRowAction_CE

Description

Event Name * Create

onRowAction_CE

Go to Custom Event

Parameters

rowKey * fx (x)

Store Result In

No return value

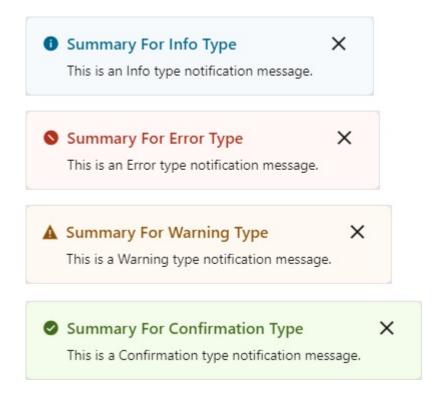
variable. If you need to create a variable, use a **Create** link in the Variables dialogue to create it at the appropriate scope level.

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.

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 }}).

- 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.

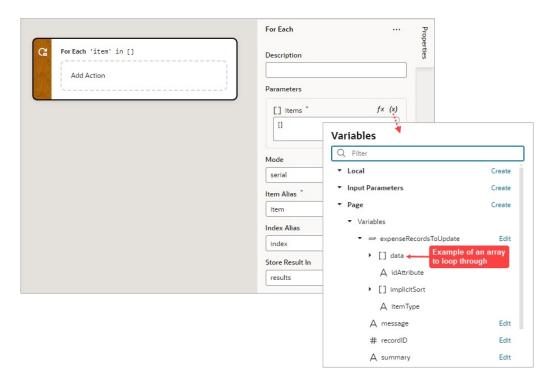
Add a For Each Action

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:

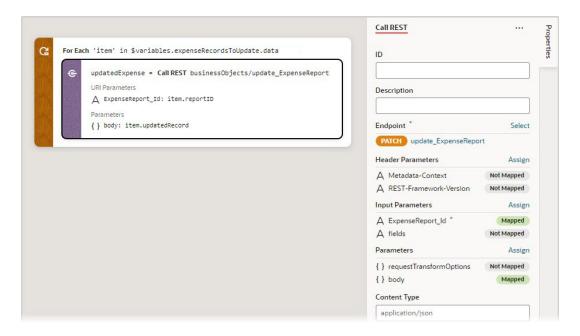
- 1. Add the action in one of three ways, as explained at the end of Built-In Actions.
- 2. Configure the action's properties in the Properties pane:
 - a. For Parameters, hover over the property's far-right side and click (x) to choose the array to loop through, such as this array, \$page.variables.expenseRecordsToUpdate.data:



- 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:

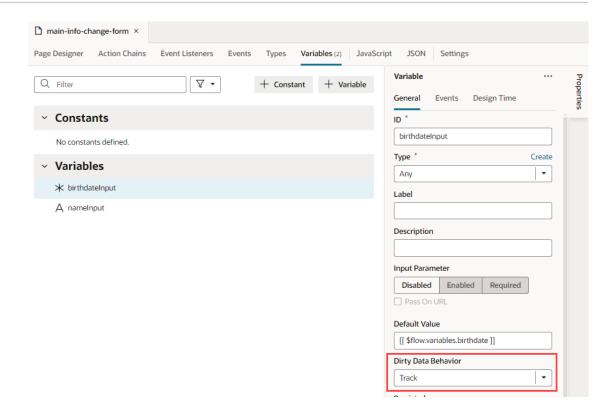


An array is returned with each element containing the return value from the last action in the loop, from each iteration. For instance, if the loop contains two actions that return results, action $A \rightarrow A$ action and the loop iterates 5 times, the returned array will have 5 elements, each corresponding to an iteration and containing action B'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.

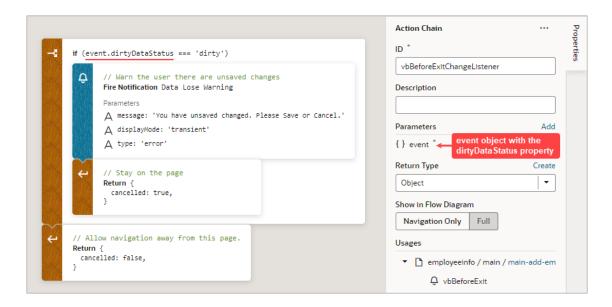
This functionality works with all data types except Service Data Providers (SDPs). You'll have to handle tracking value changes for SDPs manually, if needed.

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 <code>vbBeforeExit</code> event listener. This event listener is triggered when navigating away from the page and starts an associated action chain that receives an <code>event</code> parameter with a <code>dirtyDataStatus</code> property. Here's an example of an action chain started by a page's <code>vbBeforeExit</code> event listener, which prevents navigating away from the page if the tracked variables will lose their data:



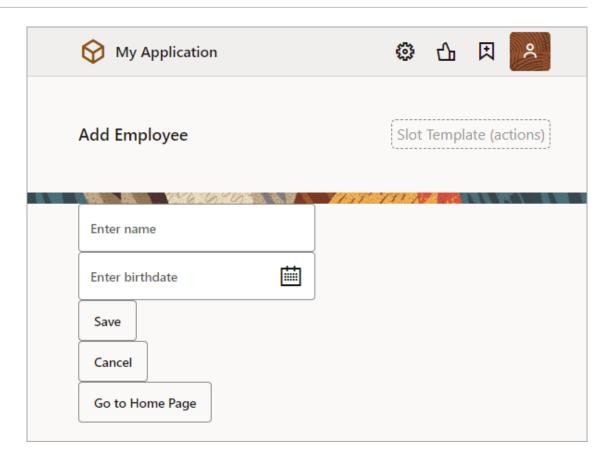


The event.dirtyDataStatus property, unlike the Get Dirty Data Status action, considers the effect that navigating away from the page would have on the tracked variables. The property is set to 'dirty' if navigating away from the page will cause the tracked variables to lose their data; otherwise, it's 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.

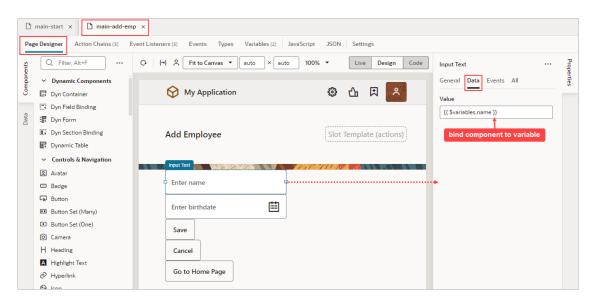
Example:

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:



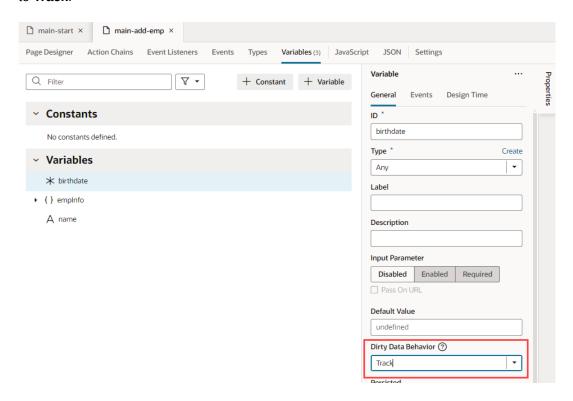
To add dirty data functionality to this example:

- 1. Set the page variables' Dirty Data Behavior property to "Track".
- 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.

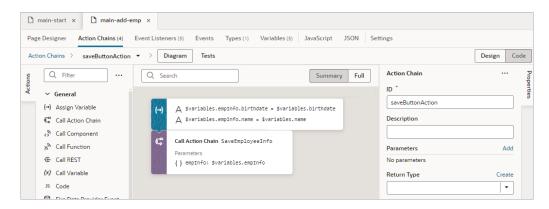
- 3. To warn users of unsaved changes due to navigating away from the page, add a <code>vbBeforeExit</code> event listener and use the <code>event</code> parameter's <code>dirtyDataStatus</code> property to check if navigation away from the page will result in the tracked page variables losing their 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**:



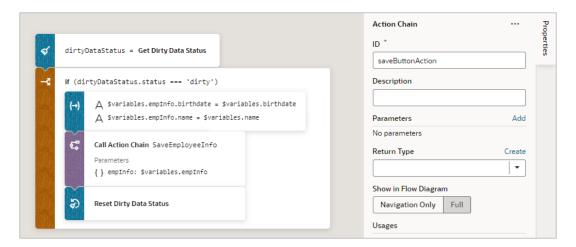
- To add the dirty data functionality to the Save button's action chain, to check if there's actually dirty data before posting:
 - a. Go the Save button's action chain. Here's an example, which passes a new employee's information to an action chain that posts the data to storage:





- **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'.

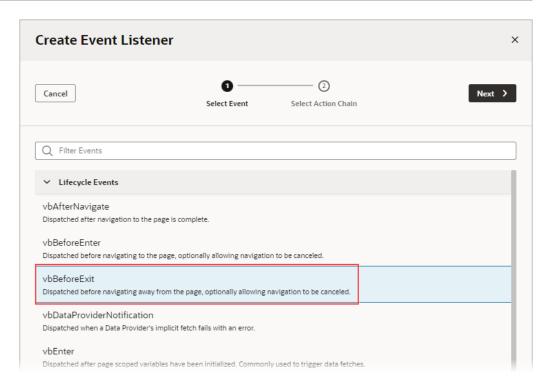
Here's the action chain with the added dirty data functionality:



- 3. Next, we need to create a <code>vbBeforeExit</code> 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 <code>event</code> object with a <code>dirtyDataStatus</code> property to check if the navigation away from the page will result in the tracked page variables losing their data.

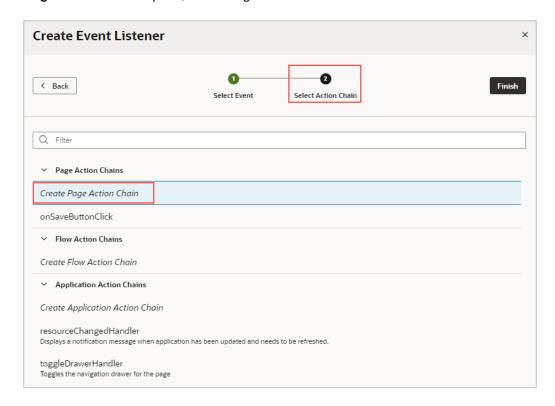
 To begin:
 - a. Open the Event Listeners tab and click the + Event Listener button to create a new event listener.
 - **b.** Select **vbBeforeExit**, which starts its associated action chain whenever a user tries to navigate away from the page. Click **Next**:



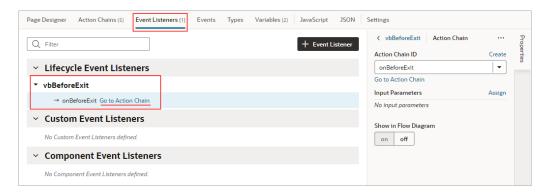


We now need to create the action chain that's started by this event listener.

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.

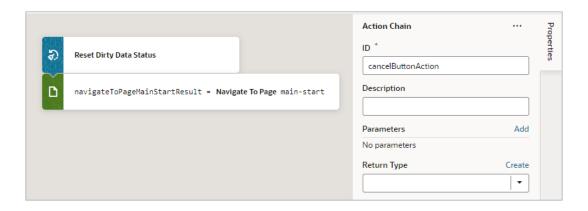
- e. Add an If action to check if the navigation away from the page results in the tracked variables losing their data.
 - Use the <code>event</code> parameter that was passed to the action chain, which has a <code>dirtyDataStatus</code> property. The property is set to 'dirty' if there will be lost 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

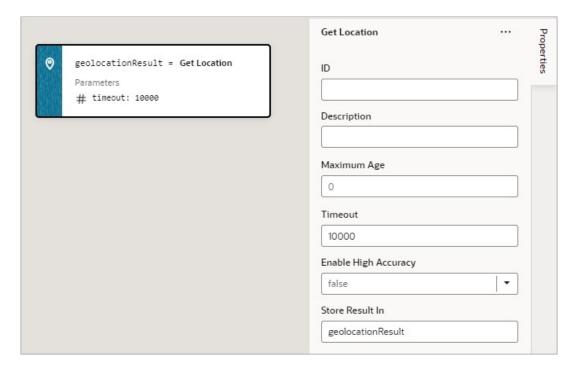
You add a Get Location action to get a user's live location.

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.

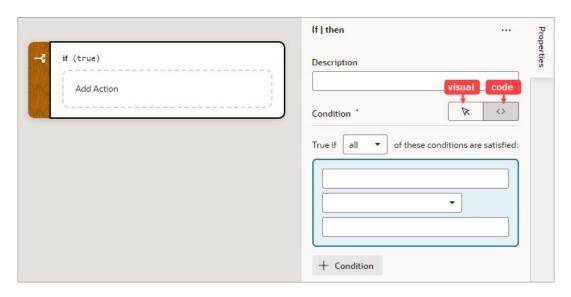
If a value is returned by the action, it is assigned to the auto-generated variable shown by the Store Result In property.

Add an If Action

You use this action to add If, Else and Else If conditions.

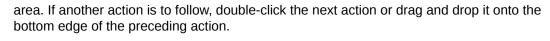
To use an If action:

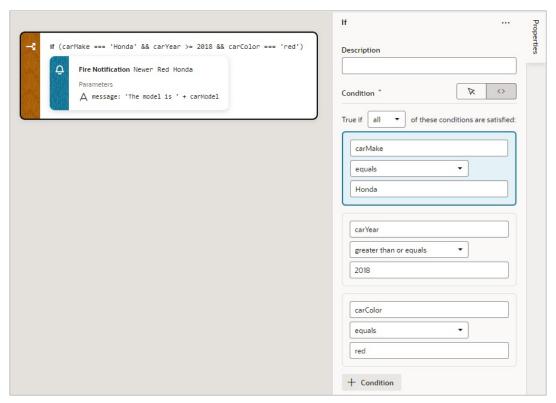
- 1. Add the action in one of three ways, as explained at the end of Built-In Actions.
- 2. With the If block selected on the canvas, use either the visual condition builder or the code condition builder to create the condition. To visually create the condition, click the first field in the blue box 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.



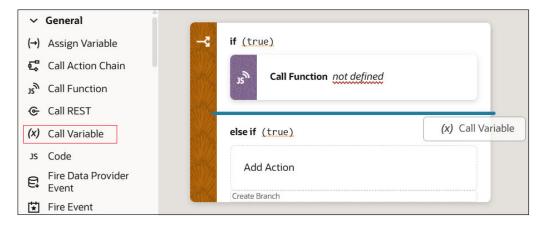
- To add another condition, click the + Condition button and create the condition as explained in the previous step.
- 4. To reorder, delete, or add a condition before or after an existing condition, right-click the condition to bring up its context menu.
- 5. To combine the conditions with an AND operator, 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**

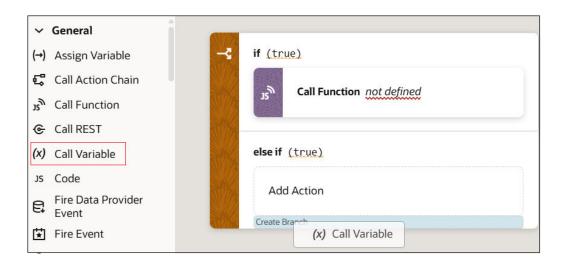




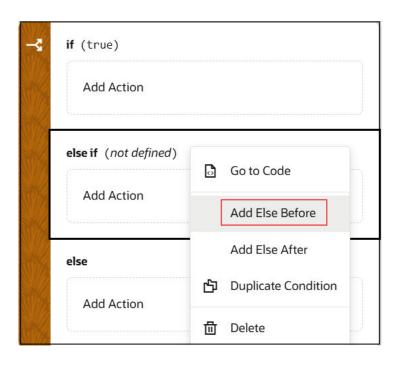


- 7. To add an Else or Else If condition, you have a few options:
 - To add a block with a specific action, from the Action palette, drag an action onto the
 Create Branch area (for the last node) or onto the Create Function drop line (if it's
 not the last node).





- To add an empty Else block at the bottom, right-click the If block and click **Add Else**.
 - If the If block is the last block, VB Studio adds an Else block.
 - If there's already an Else-If or Else block after it, VB Studio adds an Else-If block instead.
- To add an empty block at a specific position, right-click an Else or Else-If block and then:
 - Click Add Else Before to get an Else-If block above.
 - Click Add Else After to get an Else block if there's nothing after, or an Else-If block if other blocks already follow.



To duplicate an If or Else If block, right-click the block and click **Duplicate**. This saves time because you can modify a copied block instead of recreating it from scratch.
 By default, an Else condition is created. To turn it into an Else If, enter a condition for it in the Properties pane.

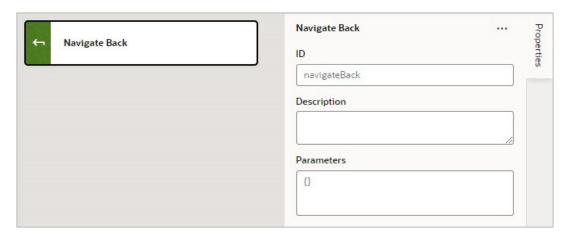
Add a Navigate Back Action

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 Application Action

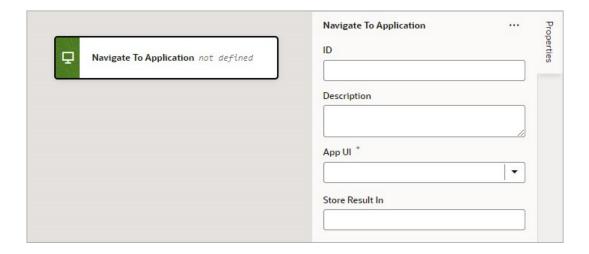
You use this action to navigate to an App UI. By default this action navigates to the specified App UI's default page, but you can also specify which flow and page to navigate to, as long as that flow or page has been set as navigable. For details on how to use the navigable setting, see steps 3 and 4 below.

For API information about this action, see Navigate To Application in the *Oracle Visual Builder Page Model Reference*.

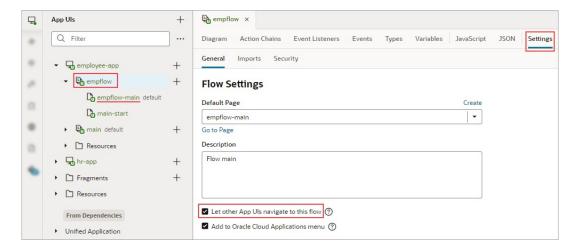
To use a Navigate To Application action:

1. Add the action in one of three ways, as explained at the end of Built-In Actions.

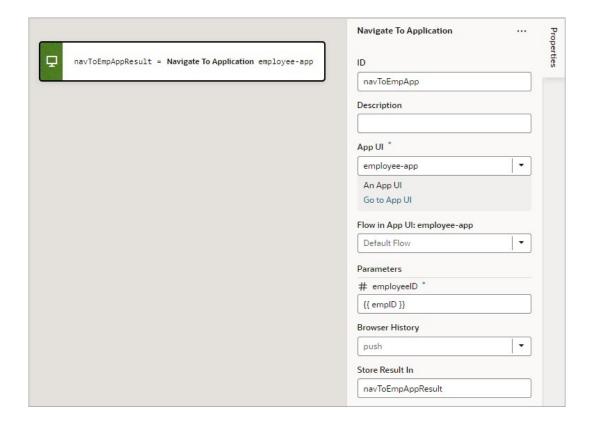




- 2. For the **App UI** property, select the App UI to navigate to. To find it more quickly, start to type the name of the App UI in the text area and select it when it appears.
- 3. For the **Flow in App UI** property, which appears after selecting an App UI, select the flow within the selected App UI to navigate to. Only default flows and flows that have their "Let other App UIs navigate to this flow" setting enabled can be navigated to and are available in the dropdown list:



- 4. For the **Page in Flow** property, select the page within the selected flow to navigate to. Only default pages and pages that have their "Let other App UIs navigate to this page" setting enabled on their Settings tab can be navigated to and are available in the dropdown list:
- 5. If the selected App UI has input parameters, enter them under Parameters, which appears after selecting the App UI.



6. For the Browser History property, 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 App UI, it is assigned to the auto-generated variable shown by the **Store Results In** property.

Add a Navigate To Flow Action

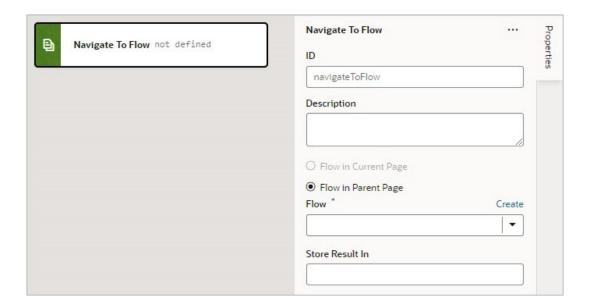
You use this action to navigate to a flow in the current App UI, and if necessary, to pass parameters to the flow. To navigate to a flow in a different App UI, use the Navigate to Application action.

For API information about this action, see Navigate To Flow in the *Oracle Visual Builder Page Model Reference*.

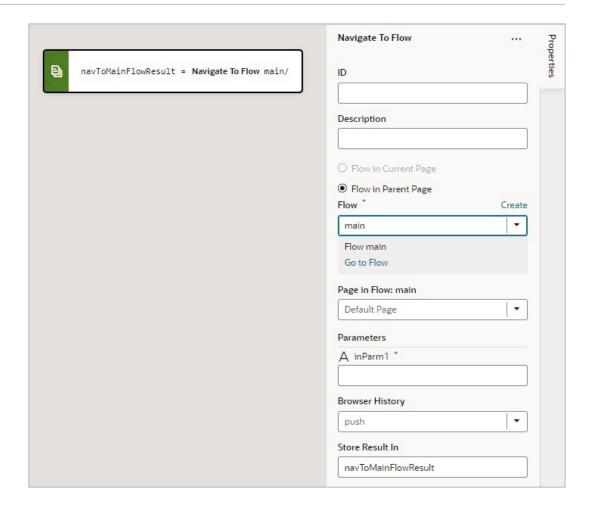
To use a Navigate To Flow action:

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.
- 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.

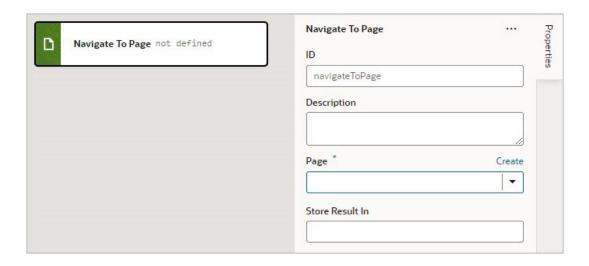
Add a Navigate To Page Action

You use this action to navigate to a page in the current App UI, and if necessary, to pass parameters to the page. To navigate to a page in a different App UI, use the Navigate to Application action.

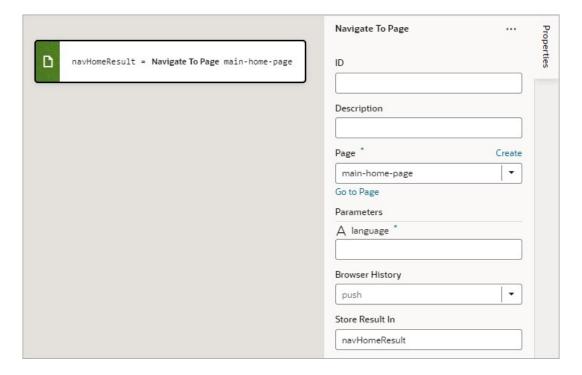
For API information about this action, see Navigate To Page in the *Oracle Visual Builder Page Model Reference*.

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.



- 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.

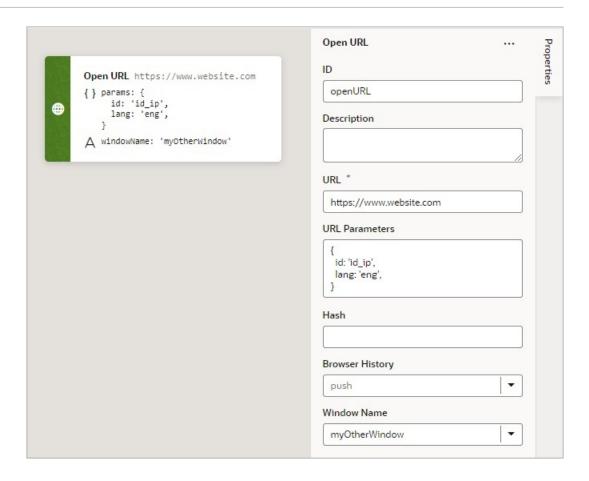
For API information about this action, see Open URL in the *Oracle Visual Builder Page Model Reference*.

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.



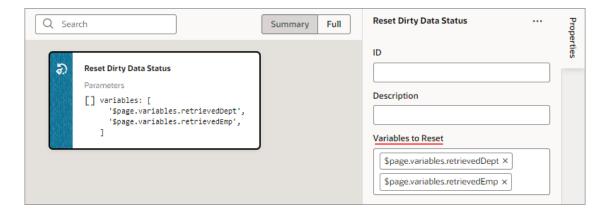


Add a Reset Dirty Data Status Action

You use a Reset Dirty Data Status action to reset the Dirty Data status of the tracked variables in the scope where the action is used (application, page, fragment, layout, flow), and in any subscopes (contained pages, fragments, layouts, flows, and any extensions of them). The Dirty Data status of the tracked variables gets set to 'notDirty'.

This action is used with the Get Dirty Data Status action, which returns 'dirty' if any tracked variable in the current scope or its subscopes has a Dirty Data status of 'dirty'.

If you'd like to reset the Dirty Data status of specific variables so they don't cause the Get Dirty Data Status action to return 'dirty', enter or select them using the Variables to Reset property.



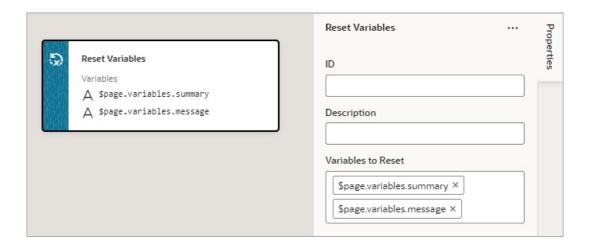
Add a Reset Variables Action

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:

- 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.

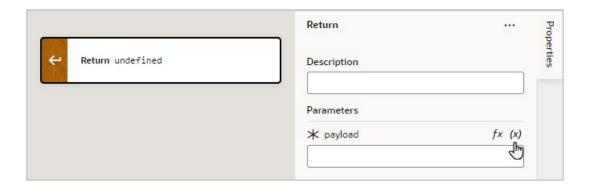
The Return action can also be used to exit an action chain early due to an exception, such as an invalid value, or some other condition. If no value is returned by the Return action, the value of undefined is returned by default.

For the Run In Parallel action, which uses <code>aysc()</code> 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.
- For the Payload property, hover over the far-right side of the property and click (x) to choose the variable to return.





Add a Run In Parallel Action

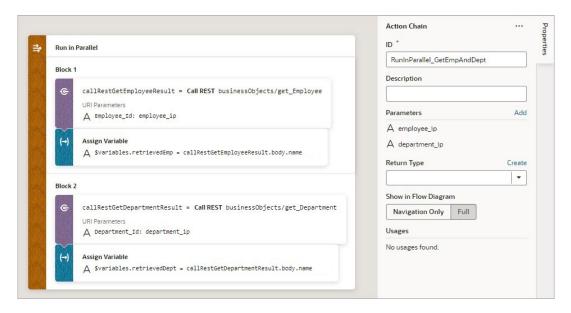
Use the Run In Parallel action to run multiple code blocks in parallel or to wait for results in order to produce a combined result.

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:

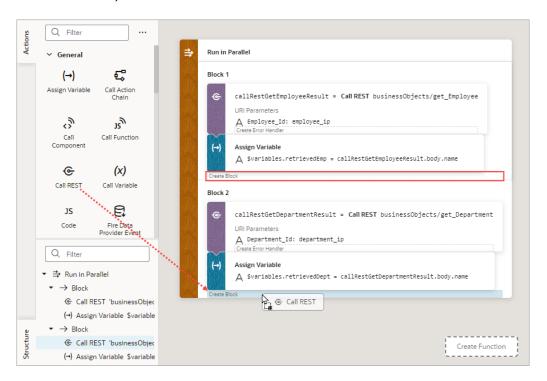
- 1. Add the action in one of three ways, as explained at the end of Built-In Actions.
- 2. Drop the actions to run for each block in the **Add Actions** area of the Run in Parallel action. For example, you could make two REST calls and assignments in parallel:

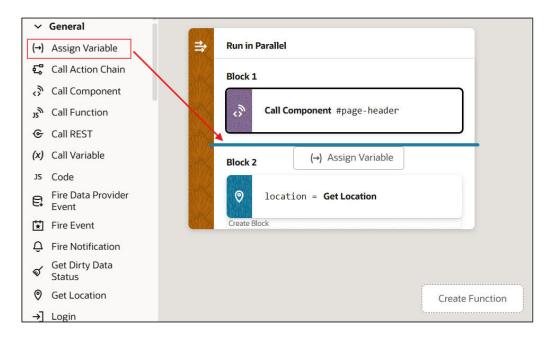


- 3. To add another block of code to run in parallel, you have a few options:
 - To add an empty block at the bottom, right-click the Run in Parallel node and click Add Block.
 - To add an empty block at a specific position, right-click a block and click **Add Block Before** or **Add Block After**.



To add a block with a specific action, from the Action palette, drag an action onto the
 Create Branch area (for the last node) or onto the Create Function drop line (if it's
 not the last node).



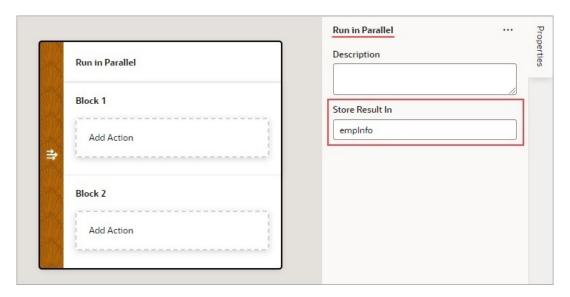


• To duplicate a block, right-click the block and click **Duplicate**. This saves time because you can modify a copied block instead of recreating it from scratch.

Use 2: Run Multiple Action Chains in Parallel to Produce a Combined Result

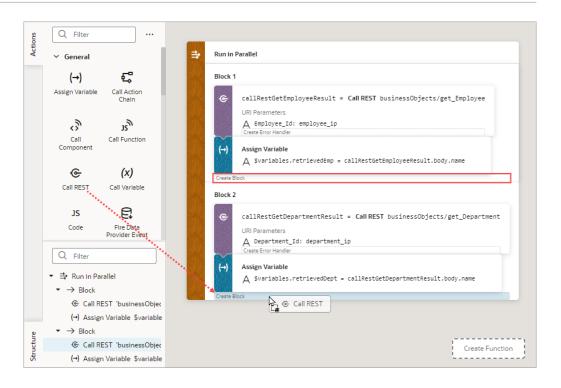
To use a Run In Parallel action to produce a combined outcome from the results of multiple action chains:

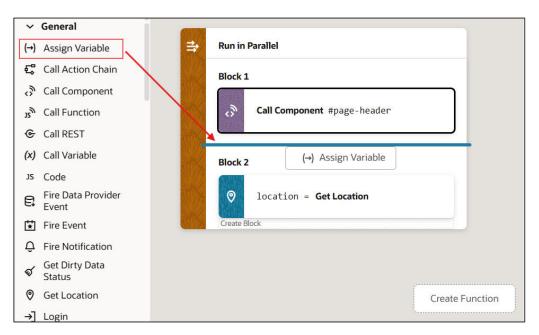
- Add the Run In Parallel action in one of three ways, as explained at the end of Built-In Actions.
- 2. For the **Store Result In** property, provide a name for the array that will hold the result from each block. The first block's result is stored at index 0, the second block's result is stored at index 1, and so on.



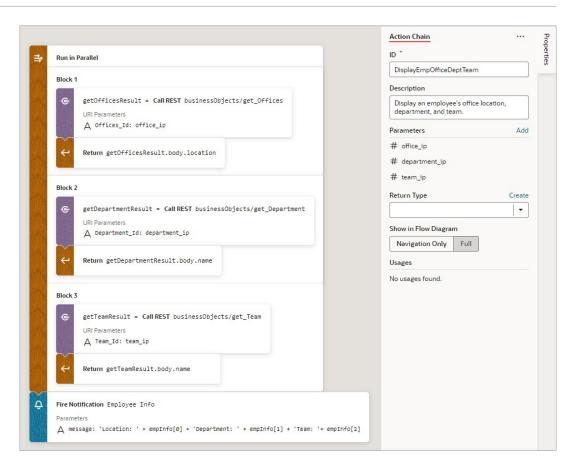
- 3. Drop the actions to run for each block in the Add Action area of the Run in Parallel action.
- 4. To add another block of code to run in parallel, you have a few options:
 - To add an empty block at the bottom, right-click the Run in Parallel node and click Add Case.
 - To add an empty block at a specific position, right-click a block and click Add Block Before or Add Block After.
 - To add a block with a specific action, from the Action palette, drag an action onto the **Create Branch** area (for the last node) or onto the **Create Function** drop line (if it's not the last node).







- To duplicate a block, right-click the block and click **Duplicate**. This saves time because you can modify a copied block instead of recreating it from scratch.
- 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.

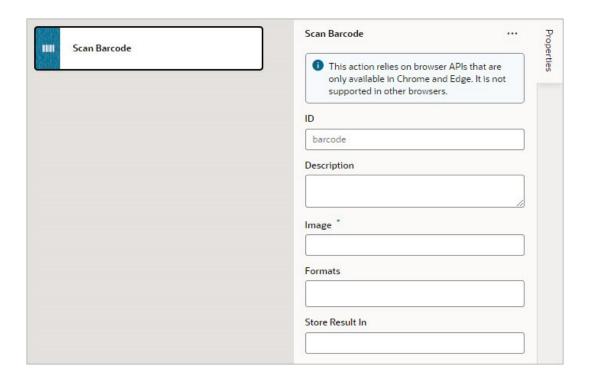
For API information about this action, see Scan Barcode in the *Oracle Visual Builder Page Model Reference*.



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:

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 element) to decode.

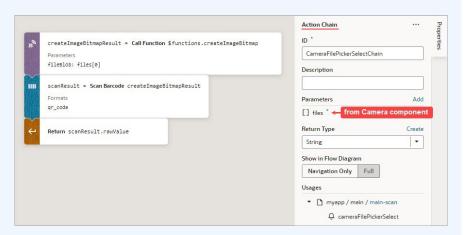
Note:

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:



- Pass the converted ImageBitmap as the Image property for the Scan Barcode action.
- 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*.



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.
- 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.

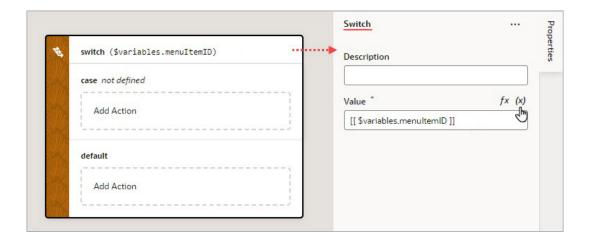
For API information about this action, see Switch in the *Oracle Visual Builder Page Model Reference*.

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 (X) 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.

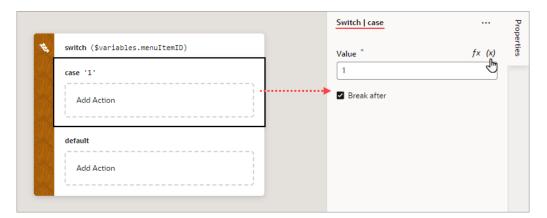
For example, when a menu defines a set of options, you can use the Switch action to determine which option was selected and to perform actions for that option. In this case, the page variable menuItemId holds the selected menu option, and it's selected using the Switch action's **Value** property:





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 (x).

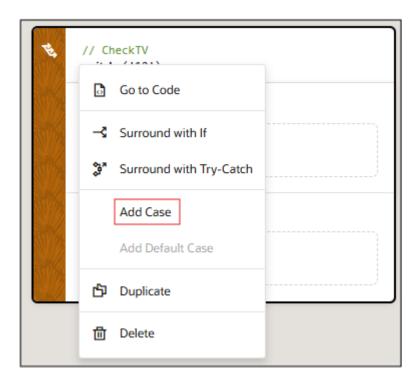


• Drop the actions to execute for the case in the Add Action area of the case block and configure their properties in the Properties pane:

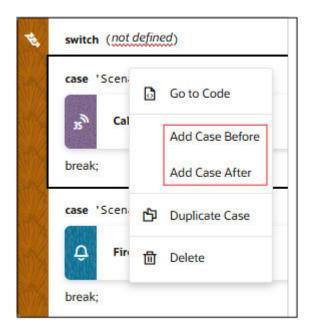




- If the rest of the cases are to be ignored, select **Break after**. If not selected, the rest of the cases get executed, regardless of their values, until running into a break statement (if at all).
- **4.** To add more cases to the switch action, you have a few options:
 - To add an empty case at the bottom (just before the default case, if it exists), right-click the Switch node and click Add Case.

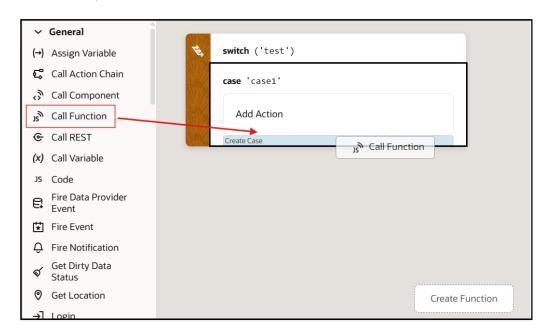


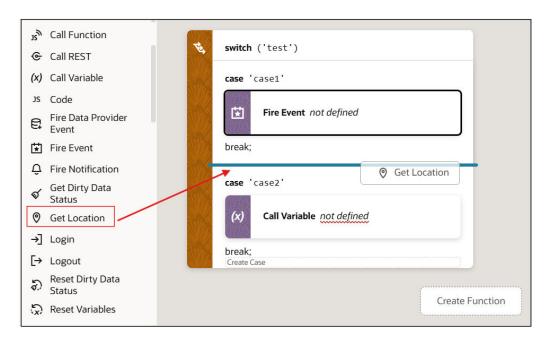
 To add an empty case at a specific position, right-click an existing case and click Add Case Before or Add Case After.





 To add a case with a specific action, from the Action palette, drag an action onto the Create Case area (for the last node) or onto the Create Function drop line (if it's not the last node).





To duplicate a case, right-click a case and click **Duplicate**. This can be useful if you
want to modify an existing case instead of recreating one from scratch.

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:

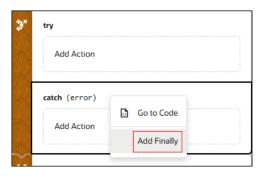


3. To add an action to a Try-Catch block, select the block and double-click the action in the Actions palette, or drag an action from the Action's palette onto the desired block:

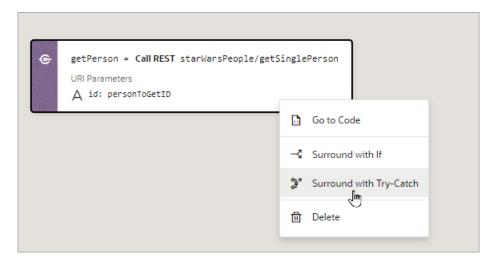


When you add a Try-Catch action to an action chain, you can't delete the Catch or Finally block in *Design* view. However, you can delete these blocks in *Code* view (by deleting the code that forms these nodes). If you've deleted the Catch or Finally blocks and want to add them back, here's what you can do:

- To add a Catch node, in Design view, right-click the try or finally block and click Add Catch.
- 2. To add a Finally node, in Design view, right-click the catch block and click Add Finally.

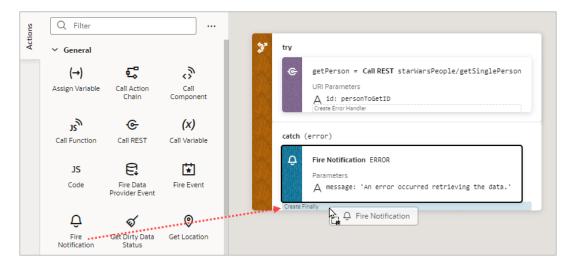


You can also surround an action with a Try-Catch action by right-clicking it and selecting **Surround with Try-Catch**:



If you need a Finally block, you have a few option:

• To add a Finally block with a specific action, drag the action that you want to the Try-Catch action and drop it into the **Create Finally** area that appears:



To add an empty Finally block, right-click the try or catch block and click Add Finally.

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 in the global Resources folder, which means they can be available to all the App UIs within your extension. For details about how to make a custom action available to other App UIs, see the end of this section, Add the Metadata.

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:

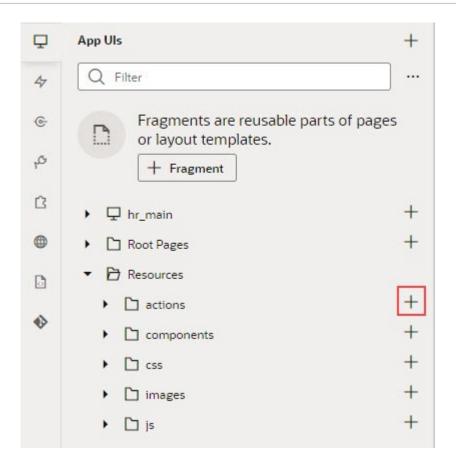
- 1. Create the Action Files (action.json and action.js):
 - action.json: Contains the metadata for the custom action. Used to define input
 parameters and to define an object for returning values. This file is also used by the
 Designer to add the action to the Actions palette, and to display the action's 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:

 Select the App UIs tab, expand the global Resources node, then click the Create Custom Action icon (+) next to the actions node:



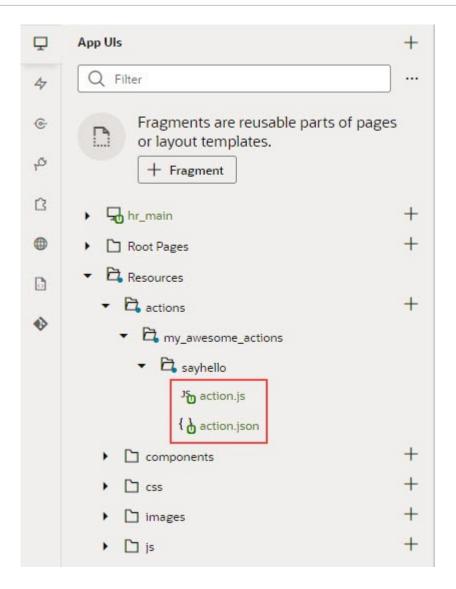


2. In the ID field, enter the name of the action group folder for your new custom action, followed by a forward slash and the name of your new action, as in: <action-group>/ <action-name>

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:





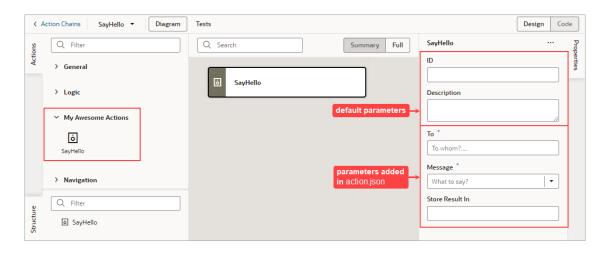
Add the Metadata

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:



```
1
               2
                     "id": "demo/sayHelloAction",
                     "idPrefix": "sayHello",
               3
                     "category": "My Awesome Actions",
    basic details
                      "defaultParameters": {},
                     "description": "Say hello to with a message",
               6
               7
                     "displayName": "Say Hello",
               8
                     "helpDescription": "Blah",
               9
                      "iconClass": "oj-ux-ico-hospitality",
                      "resultShape": {
              10
   return object
                       "result": "string"
              13
                      "propertyInspector": [
              14
                          "name": "to",
              15
                          "label": "To",
              16
                         "type": "string",
input parameters
                          "component": "inputText",
                          "placeholder": "To whom",
              19
                         "required": true
              20
              21
                       },
              22
              23
                          "name": "message",
                          "label": "Message",
              24
                          "type": "string",
              25
                          "component": "comboboxOne",
              26
                          "placeholder": "Hello, World!",
              27
              28
                          "required": true,
                          "options": [
              29
                           { "label": "Hello World",
              30
                              "value": "Hello, World!"
              31
              32
                           },
                            { "label": "Bye World",
              33
                              "value": "Bye, World!"
              34
              35
                           },
                            {"label": "Greetings World",
              36
                             "value": "Greetings, World!"
              37
              38
              39
                          "supportExpression": true
              40
              41
              42
              43
```

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.

If you want your action to be available to other App UIs, add the referenceable property to the action.json file and set its value to extension. For any new extension based on your application, the custom action will automatically appear in the dependent App UI's Action Chain editor.

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, <i>Custom</i> .
"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 (namevalue 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	Require d	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.

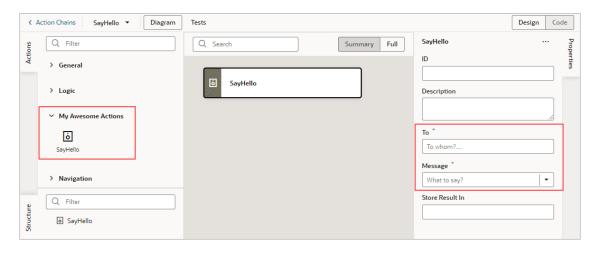
To provide the code for your action, use the perform() method, which receives the input parameter, parameters. The input parameter contains the values for the action's input parameters, as entered in the Properties pane.



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:

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 VB Studio where the custom action's code file is stored, by adding a requirejs property to the app.json file. Here are the steps:

- 1. In the Navigator, on the **App UIs** tab, select the App UI, then select the **JSON** tab to open the app.json file.
- 2. Specify the path to the custom action's code file by adding a requirejs property in app.json and naming the path, using this format:

```
""<custom-action-ID>": "ui/self/resources/actions/<action_group>/<action_name>/
action"
```

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": "ui/self/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 <code>vbEnter</code> is fired to load a page. Or, use the <code>onValueChanged</code> 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, layout, or fragment, you'll need to create a component event, with a component event listener, if you want it to trigger some behavior (for example, to open a URL).

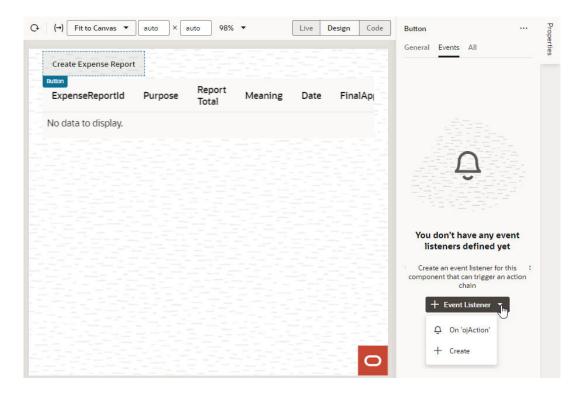
You can apply several predefined events to a component, but the available events are usually determined by the component. For example, the <code>ojAction</code> event is triggered when a button is clicked, so you would typically apply it to a button component (you couldn't apply it to a text field component). Each button will have a unique event and an event listener listening for the button's <code>ojAction</code> 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.



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.

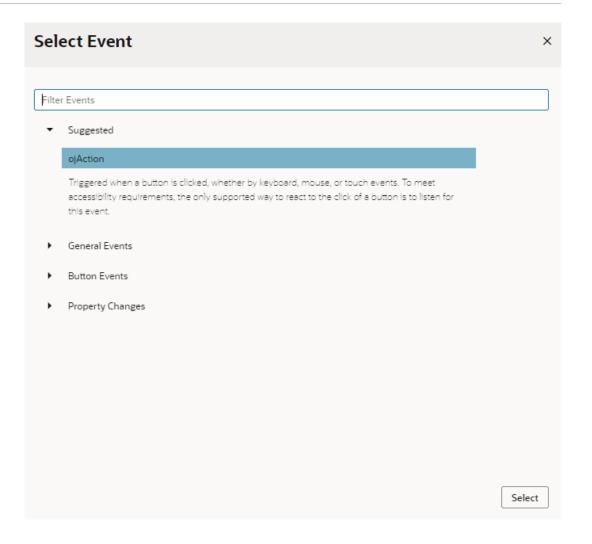
To start an action with a component:

- 1. Select the component in a page, layout, or fragment. Typically, you assign events to elements such as buttons, menus, and fields in form components.
- 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.

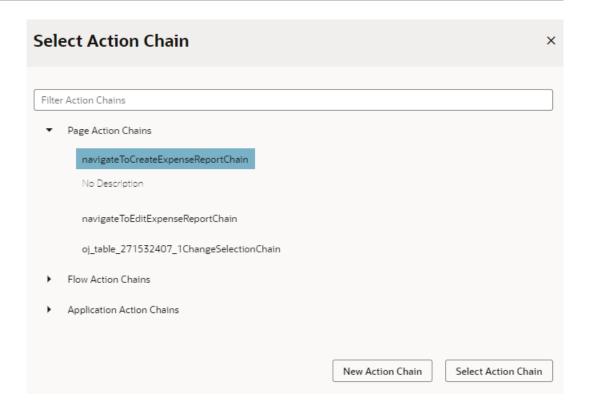


When you add the new event using the quick start option, an action chain is created for you and the Action Chain editor opens automatically. When you add the new event using the custom option, you'll need to select an event.

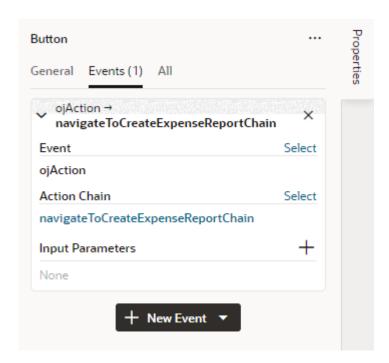
3. For a custom event, select the event you want to use to trigger an action chain. Click **Select**.



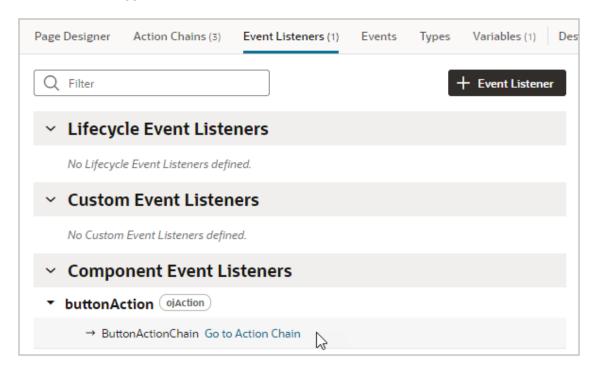
4. Select the action chain you want the event to trigger and click **Select Action Chain**. Alternatively, click **New Action Chain** to create a new action chain.



The Events tab in the Properties pane shows events on the component that VB Studio 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 <code>onValueChanged</code> event to the variable.

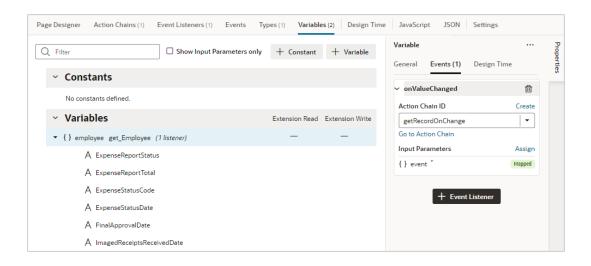
When you use an onValueChanged event to trigger an action chain, the trigger has the payload of the variable's old and new values. For example, let's say you changed an employee's Name property and reset the Employee; the framework sends an event that the Employee has changed, and as part of the payload, indicates that the name has changed.

To start an action chain when the value of a variable changes:

- Open an artifact's Variables editor.
- 2. Select the variable in the list, then click **Events** in the Properties pane.
- 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 <code>onValueChanged</code> 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 occurrences that take place 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 <code>vbEnter</code> 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 App UI 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 <code>vbEnter</code> 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 in a page:



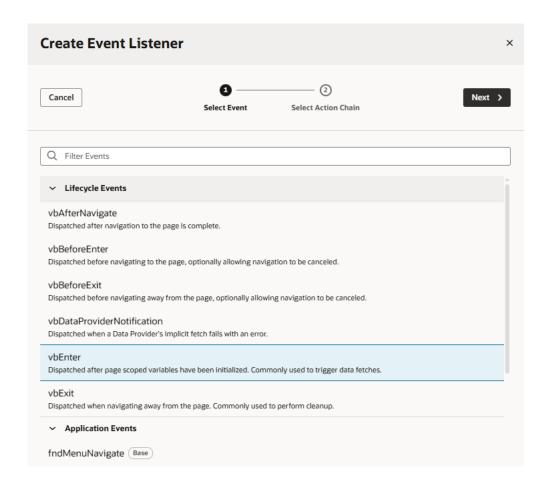
Lifecycle Event	Description
vbBeforeEnter	Triggered before navigating to a page. Commonly used when a user does not have permission to access a page and to redirect the user to another page (for example, a login screen). Because this event is dispatched to a page before navigating to it, you can cancel 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 App UI 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 App UI. Commonly used to fetch data. For this event, you can use these variable scopes to get data: \$application: All App UI 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 App UI. 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</object></object></string></string>
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.
vbDataProviderNotif ication	Triggered when a Data Provider's implicit fetch fails with an error.

Note:

Lifecycle events are also supported in fragments and work similar to those used by a page; however, not all events are available when you're working with a fragment. See Define Actions and Events in Fragments for the lifecycle events supported within a fragment.

To start an action from a lifecycle event:

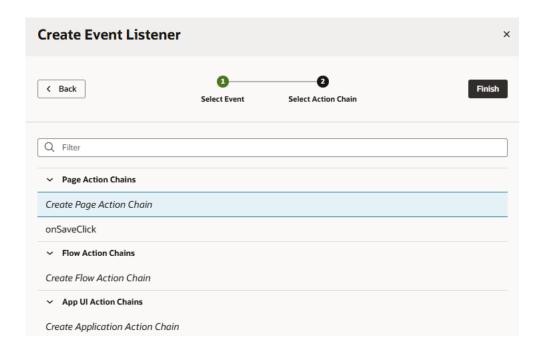
- Open the Event Listeners tab for the page (or fragment) containing the event you want to trigger an action chain for.
- Click + Event Listener.
- 3. In the Create Event Listener wizard, under Lifecycle Events, select the event you want to trigger an action chain for. Click **Next**.



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 App UI.

If you want to create a new action chain, select the create action chain option at the appropriate level (page, flow, or App UI), then click **Finish**. Keep in mind that in a fragment, you create action chains only within the scope of the fragment.





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 another action chain to display a notification, transform data, and so on. You can also trigger a custom event by using an event helper's fireCustomEvent() method in a JavaScript module function.

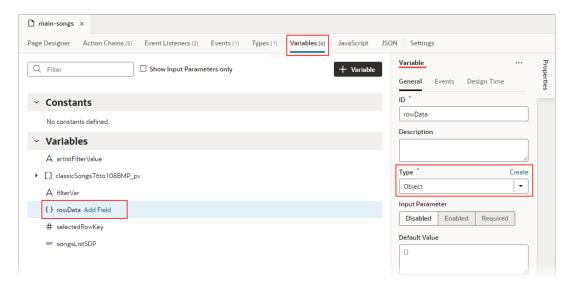
See Add a Fire Event Action to learn how you can use the Fire Event action to trigger a custom event.

In this example, we'll use a module function to subscribe to a table's <code>ojRowAction</code> 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.

```
employeeinfo-main ×
Page Designer Action Chains (2)
                             Event Listeners (2) Events (1) Types (1) Variables (2) Design Time
                                                                                             JavaScript
                                                                                                        JSON Settings
 1
      define([], () \Rightarrow {
        'use strict':
 4
        class PageModule {
 6
          constructor(context) {
                                                              Called by the page's vbEnter event,
            this.eventHelper = context.getEventHelper();
                                                               which is triggered by a page's entry
 8
 9
          subscribeToTableRowActionEvent(table)
10
11
             table.addEventListener("ojRowAction", (event) => {
               this.eventHelper.fireCustomEvent("onRowAction_CE", {rowKey: event.detail.context.key});
12
13
14
15
16
           Triggers custom event, onRowAction_CE > Triggers event listener > Starts action chain
17
18
        return PageModule;
 19
      });
```

(For more information about the module function event helper, see Module Function Event Helper.)

1. Create a page variable of type Object to hold the selected row's data:



2. Create a JavaScript function (module function) to subscribe to the table's <code>ojRowAction</code> event, which is triggered when a user clicks a table's row. You'll use this <code>ojRowAction</code> event to trigger your custom event, which will start the action chain that saves the row's data to the <code>rowData</code> page variable.

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 the required payload. The event parameter contains the row's data (event.detail.context.data).

```
nain-songs ×
Page Designer Action Chains (5) Event Listeners (2) Events (1) Types (1) Variables (6)
                                                                                            JSON Settings
                                                                                JavaScript
      define([], () \Rightarrow {
 2
       'use strict';
 4
        class PageModule {
          constructor(context) {
 6
            this.eventHelper = context.getEventHelper();
 8
          subscribeToTableRowActionEvent(table) {
10
11
            table.addEventListener("ojRowAction", (event) => {
12
              this.eventHelper.fireCustomEvent("onRowAction_CustomEvent", {rowData: event.detail.context.data});
13
            });
14
          };
15
16
17
        return PageModule;
18
      });
```

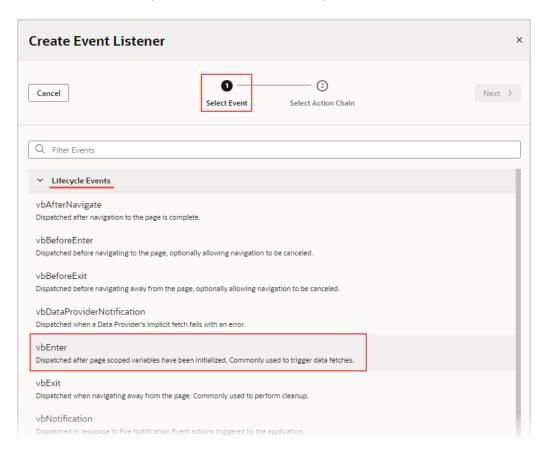
Here's the example code:

```
constructor(context) {
  this.eventHelper = context.getEventHelper();
}
```



```
subscribeToTableRowActionEvent(table) {
   table.addEventListener("ojRowAction", (event) => {
      this.eventHelper.fireCustomEvent("onRowAction_CustomEvent",
{rowData: event.detail.context.data});
   });
};
```

- 3. Create the event listener for the page's <code>vbEnter</code> 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 <code>ojRowAction</code> event.
 - a. Select the page's **Event Listeners** tab, then click **+ Event Listener**.
 - b. In the Select Event step, select vbEnter under Lifecycle Events and click Next.

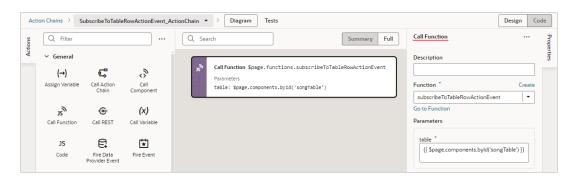


c. In the Select Action Chain step, click the Add icon next to Page Action Chains.

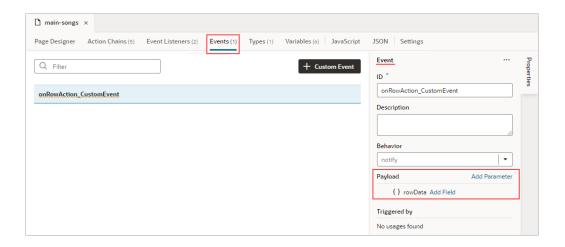


If the custom event for this listener has input parameters (which this one doesn't), the action chain would be created with an event input parameter that contains the custom event's input parameters.

4. Create the action chain to call the function that subscribes to the table's <code>ojRowAction</code> event. Add the Call Function action to the canvas. Set its <code>Function Name</code> property to the JavaScript function, and pass the table to the function using the table parameter:



- 5. Now 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.
 - a. On the page's **Events** tab, click **+ Custom Event** 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.



- b. 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.
- 6. 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).
 - a. On the page's **Event Listeners** tab, click **+ Event Listener**.
 - **b.** In the Select Event step, scroll down to Page Events and select the custom event that you created. Click **Next**.

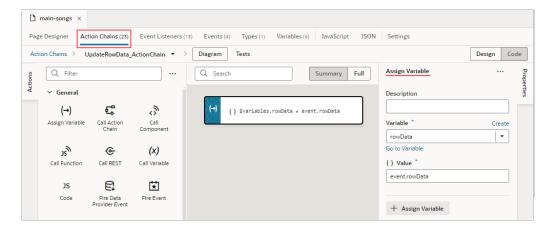


c. In the Select Action Chain step, click the Add icon next to Page Action Chains.



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.

d. 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, set its Variable property to the page variable that will contain the row's data, then set its Value 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.

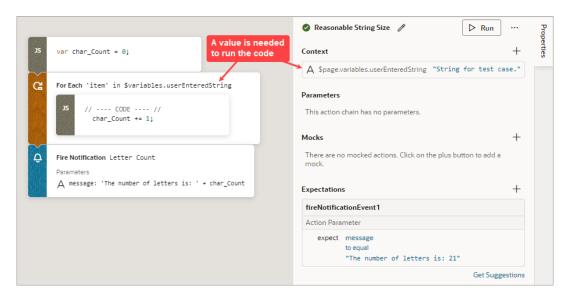
The Tests editor removes the need to manually code a test for each code path by:

- Displaying the action chain's input parameters, context variables, and constants, so you
 can easily enter their values for testing.
- Displaying the actions that need their results provided (mocked) for the code path being tested.
- 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, 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, the code in the following image 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.

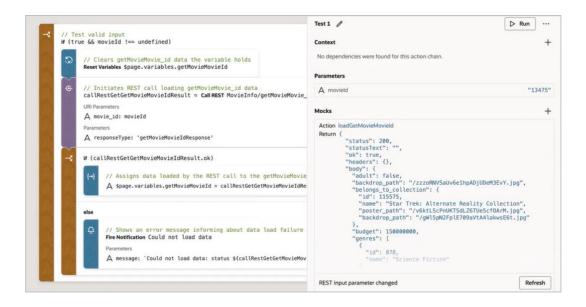


 For actions that can't automatically return a value during testing (such as a Call REST action), provide the action's return value as a mock.

You'll see the Mocks section only when you have to provide the action's return value.

For example, in this Action Chain test, a Call REST action needs to get a movie using a parameter called movieId:





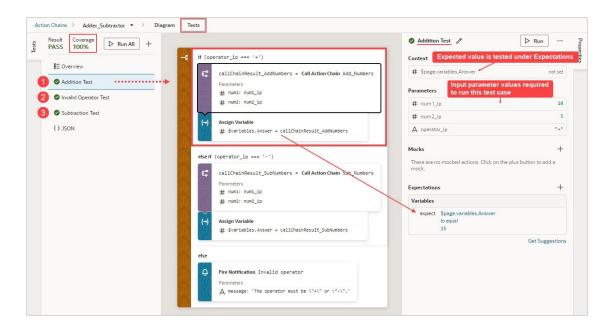
After values are provided for the test case, either by you (for actions that can't
automatically return a value) or the system (for actions that do automatically return a
value), VB Studio automatically generates expectations. Select the expected results, such
as a variable's final value, to test against.

For example, after you provide the initial values and mocks for a test, if VB Studio 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.

For the Addition Test, which tests the if operator_ip === '+' code block, you can see the three input parameter values in the **Parameters** section of the Properties pane. The expected results are in the **Expectations** section.



The source code for all of your tests is stored in a separate JSON file, actionchainnametests.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.

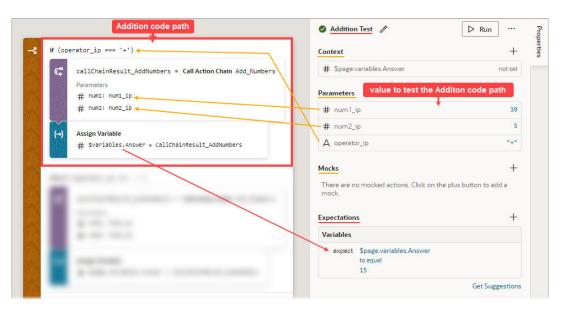
To create a test for a test case:

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.

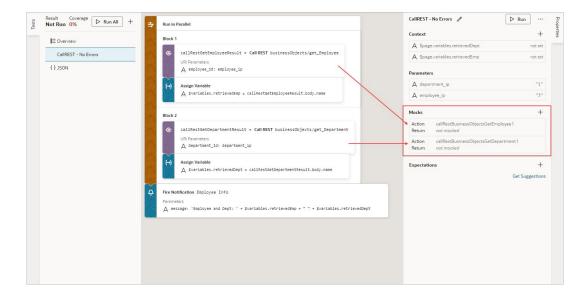
Note:

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, provide the results for any listed actions, ensuring that the values are appropriate for the test case.



These actions always require mocked results:

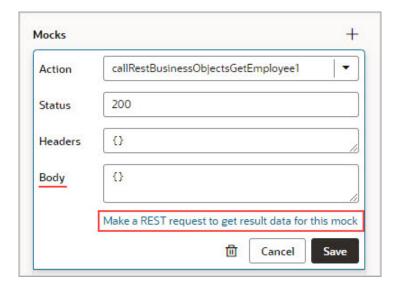
- Call Component
- Call REST
- Call Variable
- Get Location
- Scan Barcode

To provide a mock for these actions (except for Call REST), click the action in the Mocks section. In the resulting window, provide a value that is appropriate for the test case.

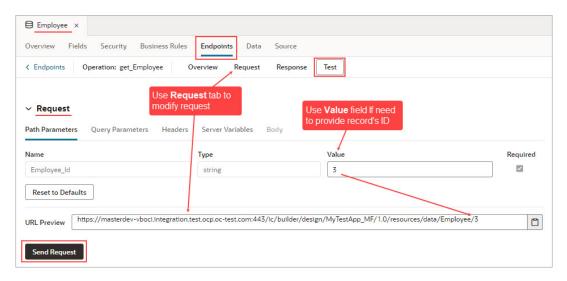
For the Call REST action, all you need to do is provide the required parameters. VB Studio will make a REST request to get data and automatically populate the mock with the response. After you click **Accept**, VB Studio generates expectations based on this mock.

```
Mocks
                                                                                       +
                                                                   Accept
 Generated REST response
                                                                                Cancel
  Action loadGetMovieMovield
  Return {
            "status": 200,
            "statusText": ""
            "ok": true,
             "headers": {},
             "body": {
               "adult": false,
"backdrop_path": "/zzzoRNVSaUv6elhpADjUDeM3EvY.jpg",
               "belongs_to_collection": {
                 "id": 115575,
"name": "Star Trek: Alternate Reality Collection",
                 "poster_path": "/v6ktLScPnUKTSdLZ6TUeScf0ArM.jpg", 
"backdrop_path": "/gWl5pN2FplE709aVtA4lakwsE6t.jpg"
                'budget": 150000000,
               "genres": [
                    "id": 878,
                    "name": "Science Fiction"
```

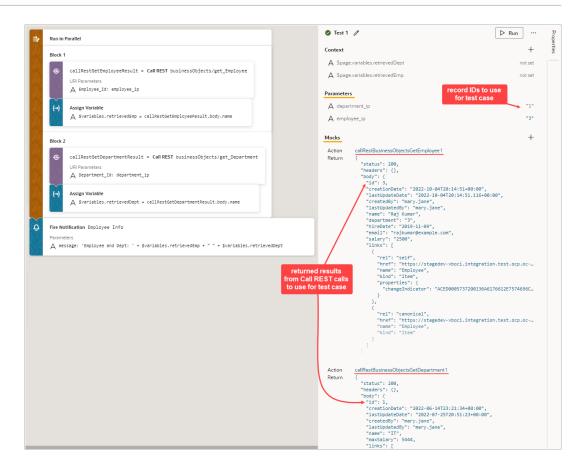
If you don't want to use the automatically generated mocks, you can always manually make a REST call and add mocks. Just click anywhere in the Mocks section, then click Make a REST request to get result data for this mock.



You'll 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:

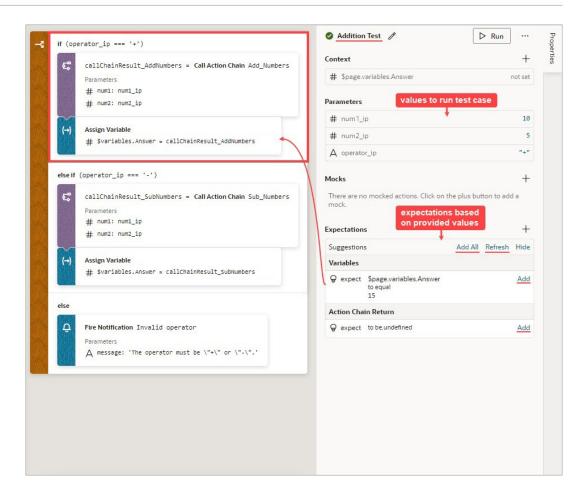


Shown here are the copied responses from the Call REST action requests, provided as mocks:

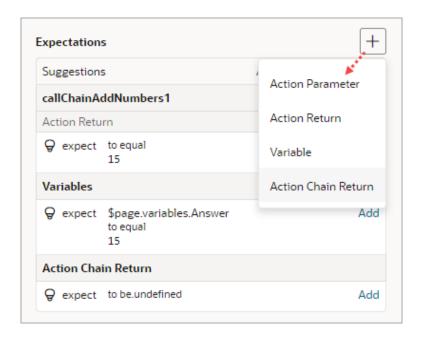


To add a mocked result for an action that isn't shown in the Mocks section, click the Add icon (+) for the Mocks section. Select the action from the **Action** drop-down list, then provide the value for the test case in the **Return** field.

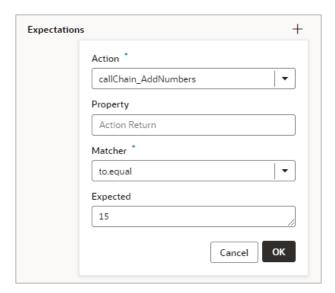
- 4. Now that you've provided the values for the test case, you can get suggested expectations based on those values by clicking the **Get Suggestions** button in the Expectations section.
 - Shown here are the suggested expectations that are based on the values entered for the Addition Test. In the 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 \mathbf{OK} :

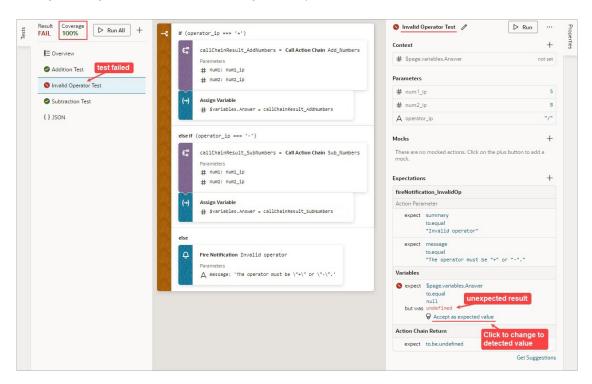


Run the Tests

Once you've defined the tests, you can run them individually or all at once using the **Run** or **Run All** button.

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:

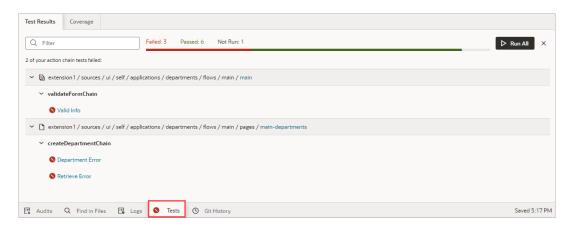


Use the Tests Footer in Your App UI

The Tests footer lets you simultaneously monitor multiple action chain tests that have been written for a App UI.

The Tests footer includes two tabs, Test Results and Coverage:

 At the top of the Test Results tab, you'll find the number of tests that have passed, failed, and are yet to be run in your App UI. Failed test for each App UI, flow, and page are shown below. For each failed test, click the link to access the editor, where you can identify and resolve any code-breaking issues.



While you can run all tests in your App UI using the Run All button, it's not required. Whenever you make changes to an action chain, the corresponding tests automatically run after ten seconds if you haven't yet run them.

• The Coverage tab provides an overview of how thoroughly the tests have covered the action chain code paths. At the top, it displays the average coverage of all tests, along with the number of failed and not-run tests. Below, the average coverage for each App UI, flow, and page is shown, along with a summary of coverage, failed tests, and not-run tests for each of their action chains.





Note:

Starting in 25.01, test coverage calculations are more accurate than they've been in the past. Specifically, the Coverage field now shows the percentage of all action chains with tests that have passed successfully.



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. To do this, 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 Employee page.

Note:

You must create all new action chains in JavaScript, which offers significant advantages over JSON. You can still edit existing JSON action chains, however. If you're not clear on the benefits of using JavaScript over JSON, see the next topic.

JavaScript and JSON Action Chains

We recommend that you use JavaScript action chains (rather than JSON), as they provide a number of benefits, including:

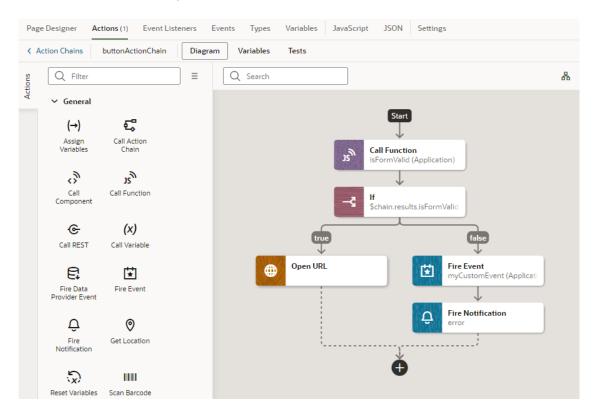
- The JavaScript Action Chain visual editor offers a helpful Structure pane.
- The JavaScript code editor includes an Actions palette, a Structure pane, and a Properties pane to assist with writing code.
- JavaScript action chains can use local functions to improve modularization and keep related logic 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.

What is an Action Chain?

An action chain drives a series of actions in response to an event from the user interface. It can be short, like an action that makes a REST call, followed by another that takes the result of that call and stores it in a variable. An action can contain both actions as well as logic that determines what happens in the sequence, such as an action that calls a common function, followed by an if action that conditionally evaluates an expression and provides alternate paths based on the function call's result.

Action chains are always triggered by events. There are many types of events, such as the <code>vbEnter</code> lifecycle event, which occurs when a page starts and can be used to fetch data; the <code>ojAction</code> event that occurs when a button component is clicked; and the <code>onValueChange</code> event that's 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 VB Studio suggests (say, the <code>onValue</code> 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 assembling predefined (*built-in*) actions in the Action Chains editor, or using JSON code to create your own. Here's an example of an action chain that uses several built-in actions to open a URL:



While actions within a particular chain run serially, you can run multiple action chains concurrently by configuring the event listener to start multiple chains.

When creating action chains, keep in mind that each action chain has a scope that depends on where it's defined: at the App UI, flow, page, or fragment level. An action chain defined at the App UI level can be called from any flow or page, but a page-level action chain can only be

called from that page—though the chain itself can access variables defined on the page, parent flow, or App UI. 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.

An action chain also 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 action chain.

Create an Action Chain

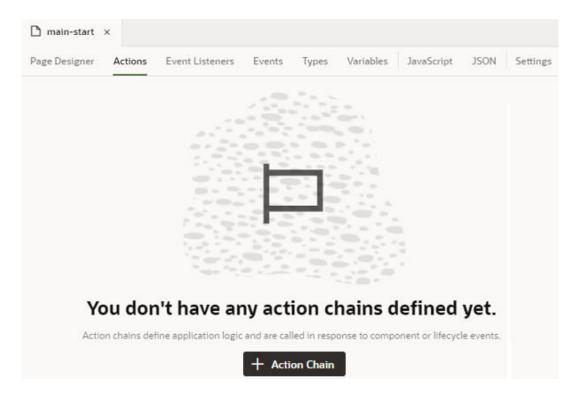
Create action chains by assembling several, individual actions into a sequence in the Action Chains editor. The Actions palette contains a list of built-in actions that you can drag on to the canvas to create your sequence.

Each action performs a specific function and may potentially have multiple outcomes (such as "success" or "failure", or a branch). It can also return results. You chain actions by connecting one action to the previous action's outcome. Keep in mind that an action can't stand on its own, it can only exist within an action chain.

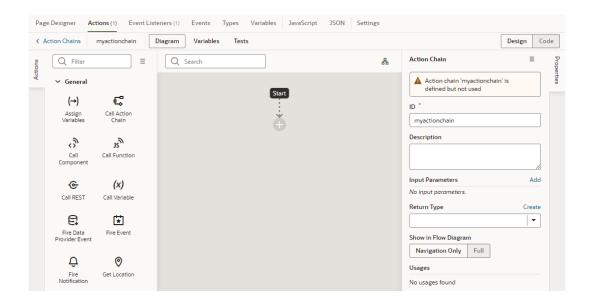
To create an action chain:

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 Studio Page Model Reference*.

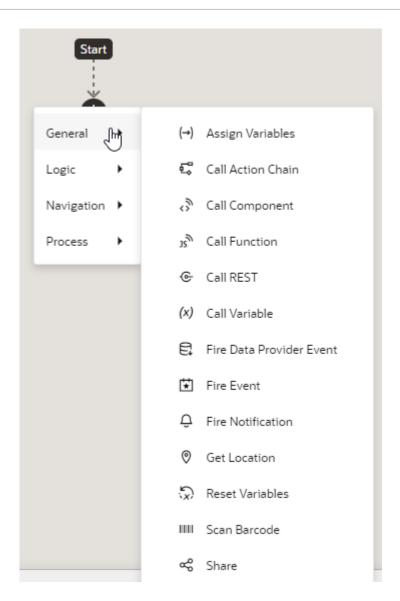


Tip:

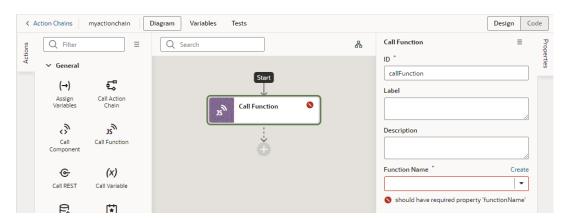
It's possible to declare local variables that are only available within the scope of your action chain. To do this, click the **Variables** tab in the Action Chains editor and create your variable. These variables are internal to the action chain and can be used internally by actions in the chain. You 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.





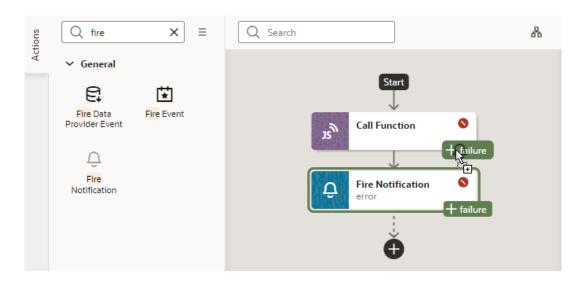
The new action is added to the chain and is selected by default. The Properties pane displays the properties that you can specify for the action. For example, here's what the editor looks like when you add the Call Function action in the Design view:



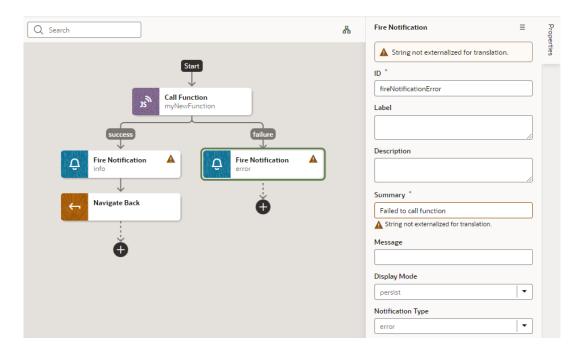
The action is usually flagged with a warning icon when a required field isn't set (in this case, because a JavaScript function hasn't been selected yet). Specify the action's

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.

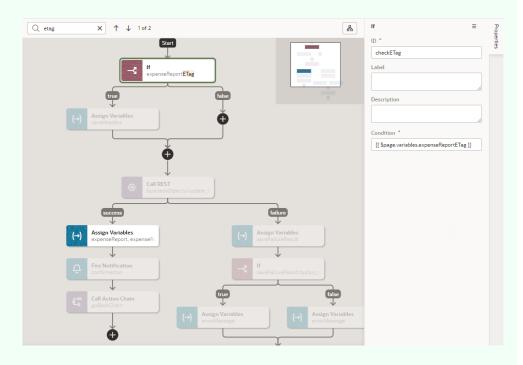




Tip:

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.

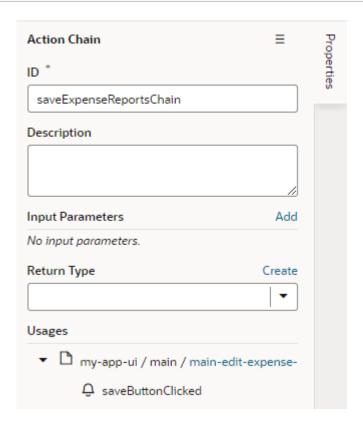
Here's an example of using "etag" to find the expenseReportEtag variable:



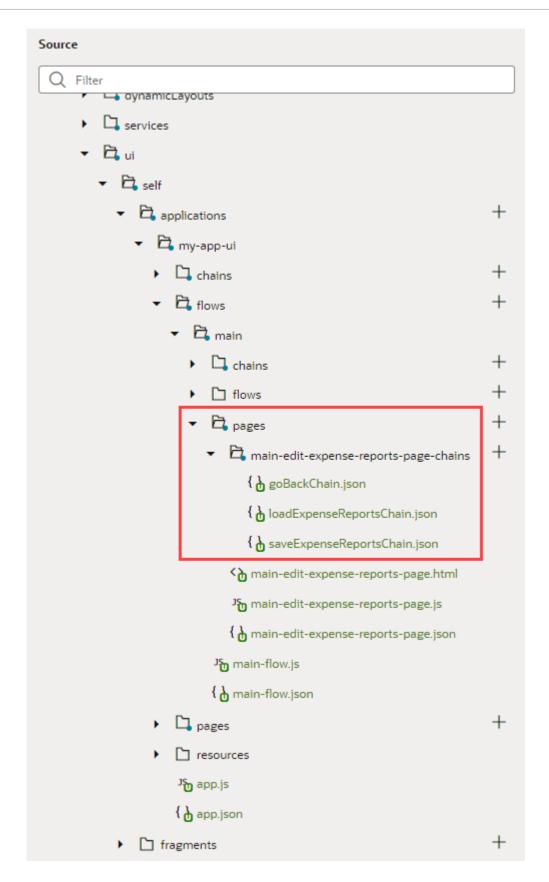
You can also click the Show Overview icon () to view a visual representation of the action chain's flow. In combination with search, the overview diagram can help you know where the highlighted action is within the overall flow. Toggle the icon to show or hide the diagram as required.

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:

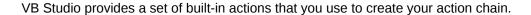


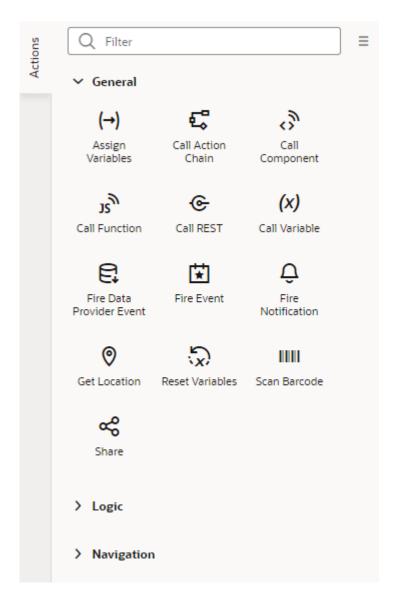
An action chain's source code is stored in its own JSON file. This helps to optimize performance by reducing the size of the artifact JSON and to reduce the potential for merge conflicts when multiple action chains are edited. To view and edit an action chain's JSON file, it's simplest to use the **Code** editor in the Diagram view, though you can always view files using the Navigator's **Source** view. For action chains in App UIs, 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:



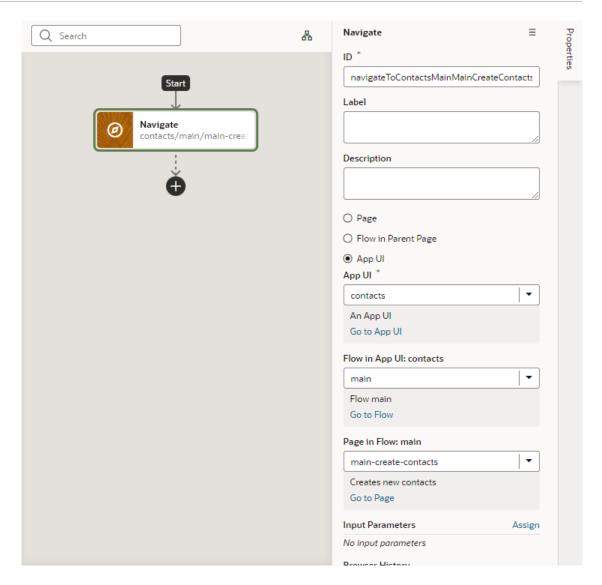
If you create tests for your action chain, those will be stored in another JSON file, distinct from the action chain's file.

Built-in Actions





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 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 VB Studio, 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.

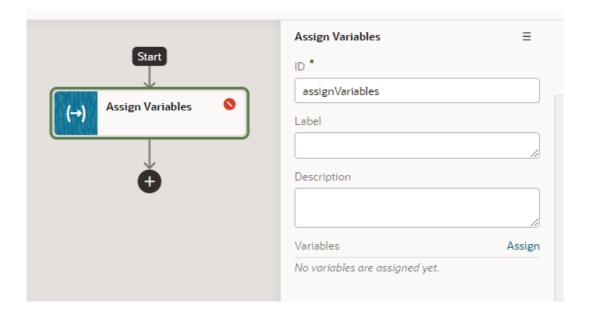
Add an Assign Variables Action

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.



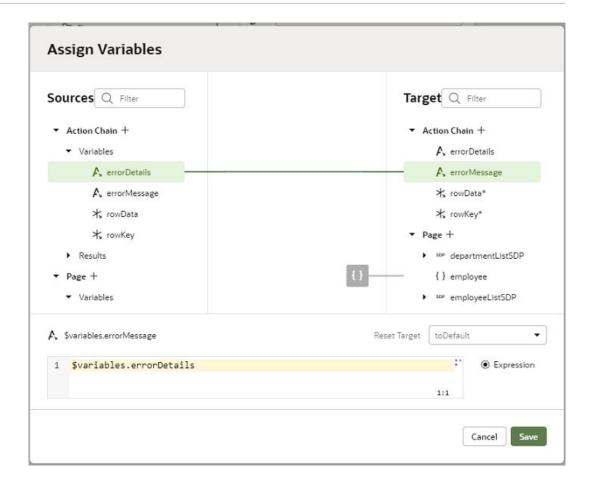
The Assign Variables action is badged with a warning icon when no variables have been assigned.

- 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.
- 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).





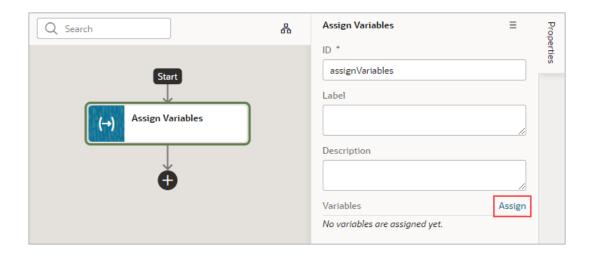
Use Filter Builder to Create Filter Criteria for an SDP

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:

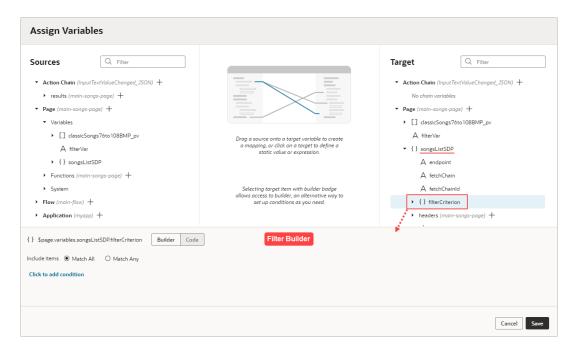
1. In the Properties pane, click **Assign** to open the Assign Variables window:



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.

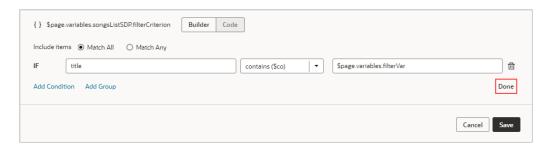


To use the Filter Builder to create the filter criterion for the SDP:

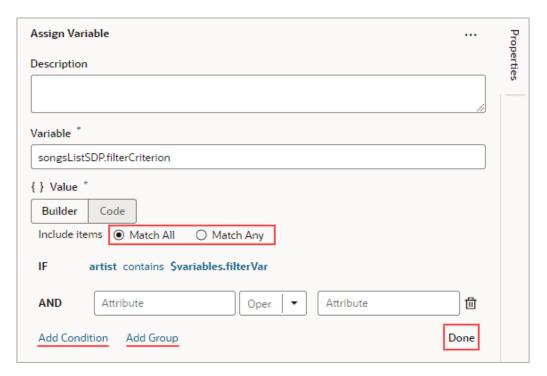
- Click the Filter Builder's Click to add condition link. To create the filter criterion:
 - a. For the first Attribute textbox, enter the name of the column (record field) that you want to compare its values against the user's inputted value.
 - **b.** For the Operator drop-down list, select the operator 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

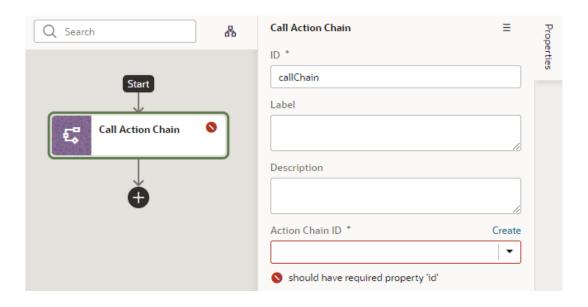
You add a Call Action Chain action to an action chain to start a different action chain, which can be either JSON or JavaScript. This action can call an action chain within your current scope or any higher scope. For example, if this action is used at the page level, it can call an action chain at the page, flow, or App UI level. However, if it's used at the App UI level, it can only call an action chain at the App UI level.

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.

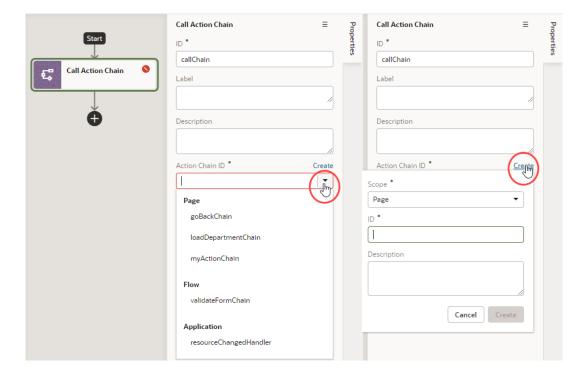
You can drag the action onto the Add icon () in the action chain, or between existing actions in the chain. The properties pane opens in the editor when you add the action to the chain.



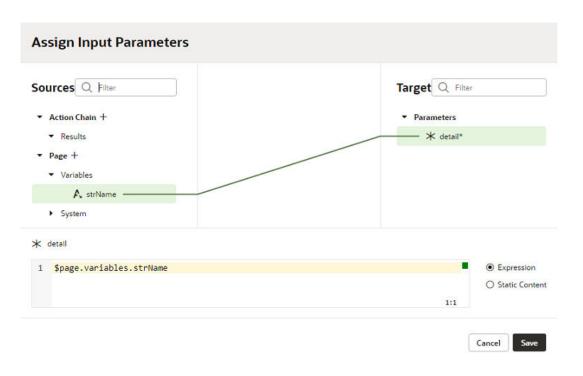


4. Select an existing action chain from the drop-down list of available action chains, or click **Create** to create a new action chain.

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** for App UI). 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 App UI. 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 App UI artifact, and you will not see an entry for Page action chains.



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**.



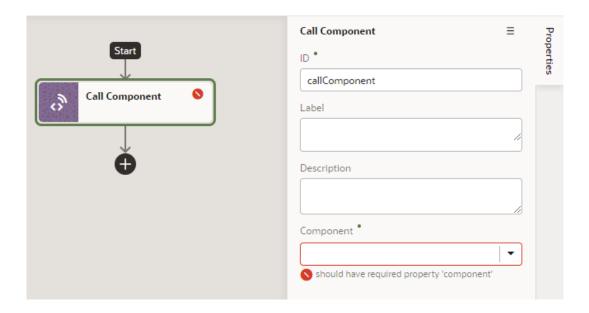
Add a Call Component Action

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:

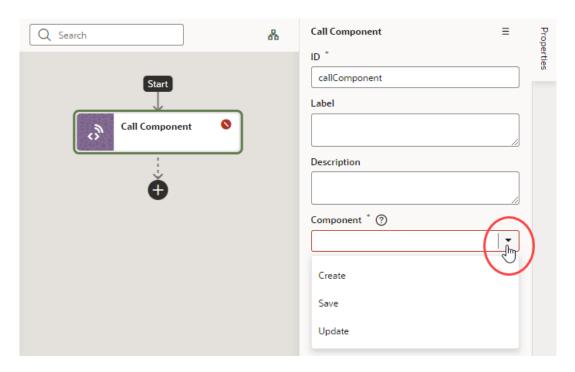
- 1. Open the Actions editor for the page or application.
- 2. Create an action chain, or open an existing action chain to add the action in the editor.
- 3. Drag Call Component from the Actions palette into the action chain.

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.

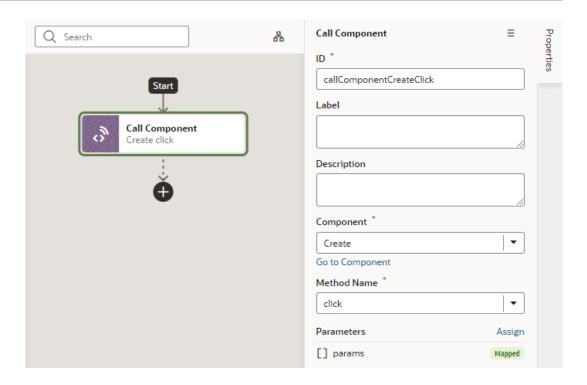


4. In the Properties pane, select the component name in the **Component** drop-down list.

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.

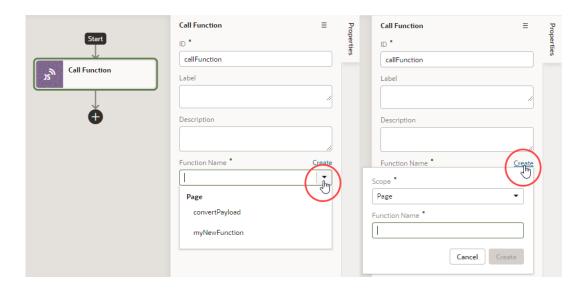
To add a Call Function action to an action chain:

- 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. 	

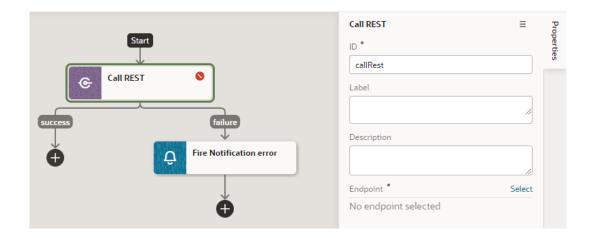


Type of Endpoint	Typical Requirements	
GET	When you call a GET endpoint, you will typically need:	
	 Input parameter: The ID of the record you want to retrieve should be passed as an input variable. 	
	 The payload of the GET call needs to be assigned to a variable using the Assign Variable action. 	
	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.

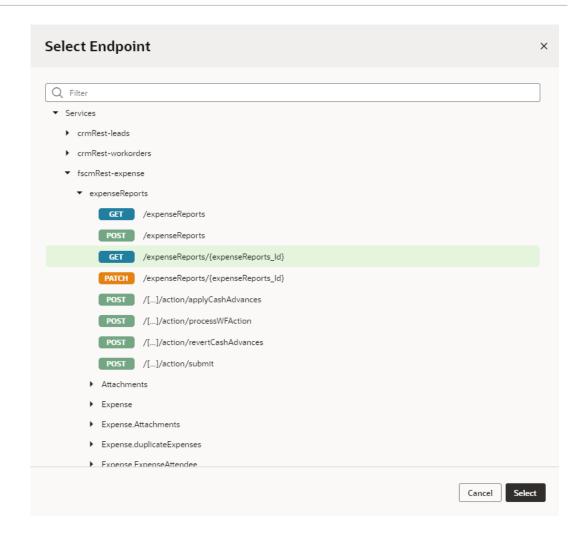
You can drag the action onto the Add icon () in the action chain, or between existing actions in the chain. The properties pane opens when you add the Call REST endpoint action to the action chain.



4. Click **Select** beside the Endpoint property in the properties pane.

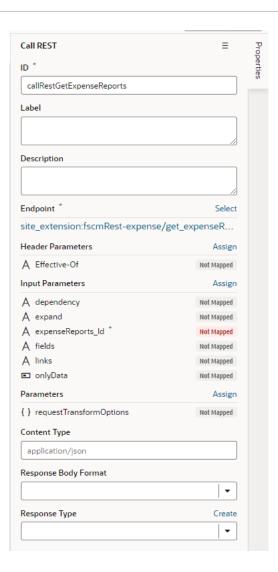
The Select Endpoint window displays a list of the endpoints that are available in your application. Each business object and service usually exposes multiple endpoints. The endpoint that you select will depend upon the function of the action chain. The endpoint that you select will also determine the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





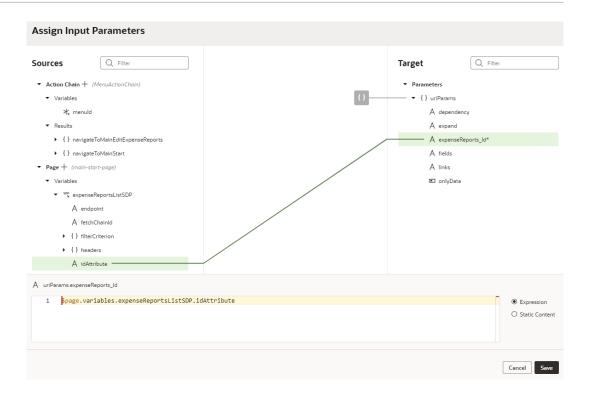
- 5. Select an endpoint from the list. Click **Select**.
- **6.** Edit the action's properties in the properties pane.

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.

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.



Because actions are by design synchronous, it will wait for the asynchronous call to resolve before proceeding to the next action in the chain.

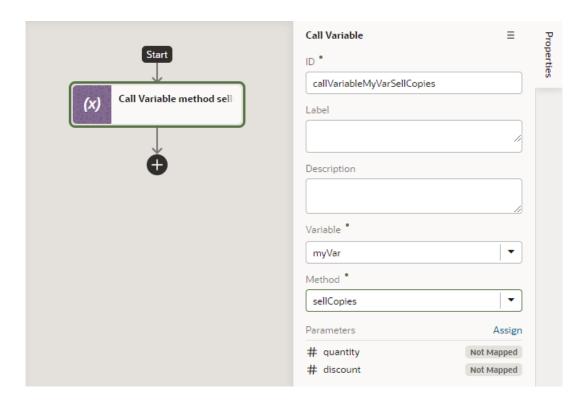
Before you use a Call Variable action in an action chain, make sure an InstanceFactory type variable is already defined for the application. See Create a Type From Code.

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.

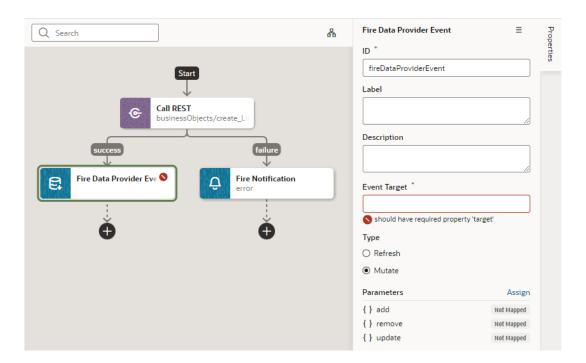
To add a Fire Data Provider Event action to an action chain:

1. Open the Actions editor, for example, at the page level.



- 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.

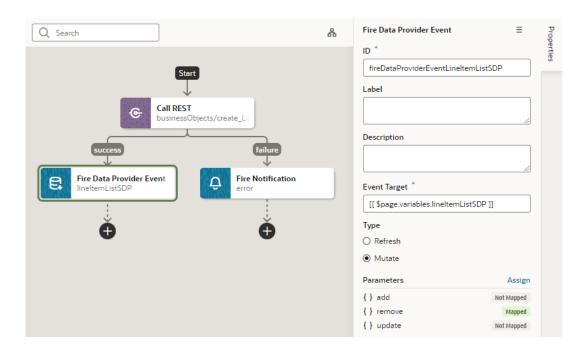
You can drag the action onto the Add icon () in the action chain, or between existing actions in the chain. The Properties pane opens when you add the action to the chain.



- 4. Update the ID field in the Properties pane to make the action more easily identifiable.
- 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.

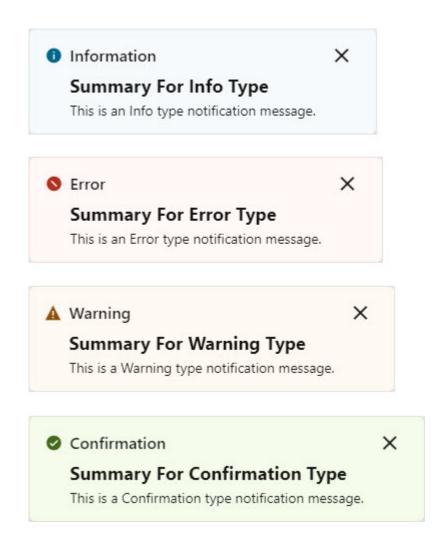
The event payload depends upon how the custom event is defined. You can use the Mapper to map the payload to a source, such as a page variable, or define a specific value or expression.

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

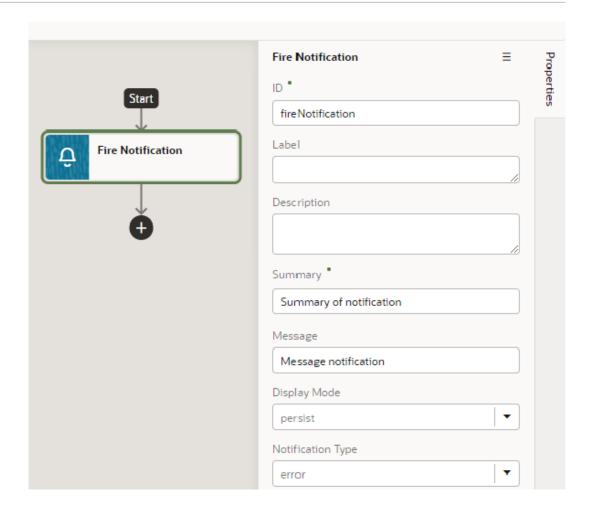
You add a Fire Notification action to display a notification to the user in the browser window.

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:

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:



- 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.

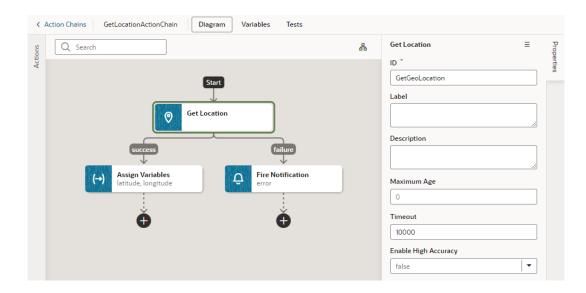
Add a Get Location Action

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.

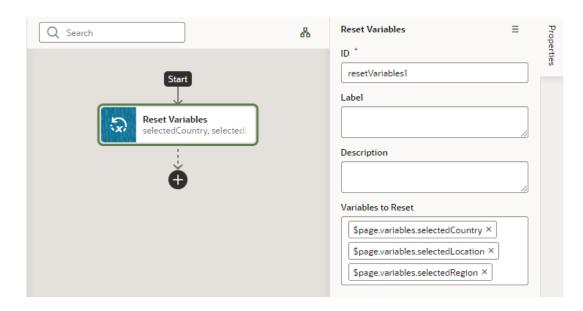
To add a Reset Variables action to an action chain:

- 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.
- 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 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.

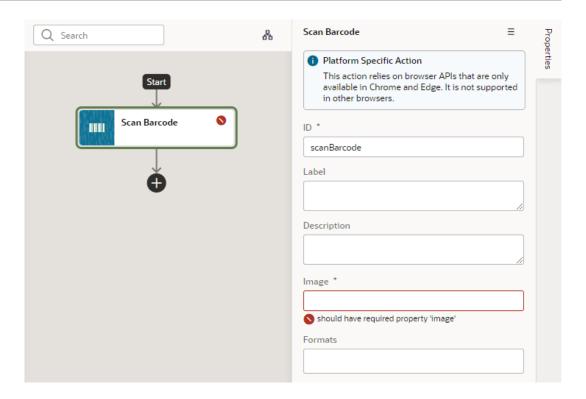
Note:

The Scan Barcode action relies on browser APIs and is supported only on Chrome for VB Studio apps.

To add a scan barcode action to an action chain:

- 1. Open the Actions editor for the page.
- 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 element) to decode.

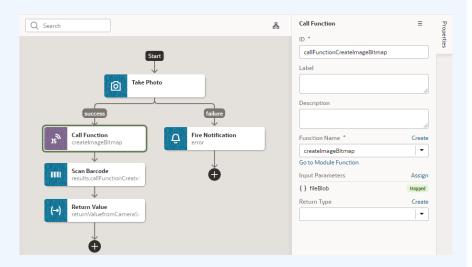
Note:

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:

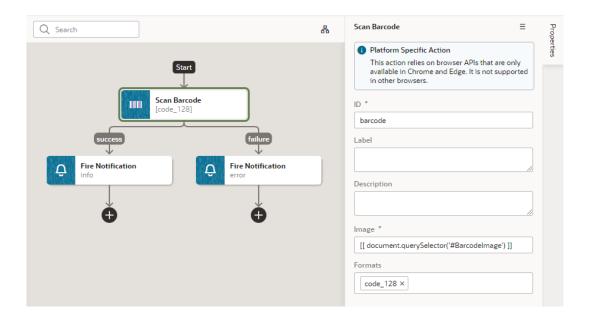
```
[[ $chain.results.callFunctionCreateImageBitmap ]]
```

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 <code>document.querySelector</code> to get the image, as shown here where the first image with the ID <code>BarcodeImage</code> will be returned:



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.



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:

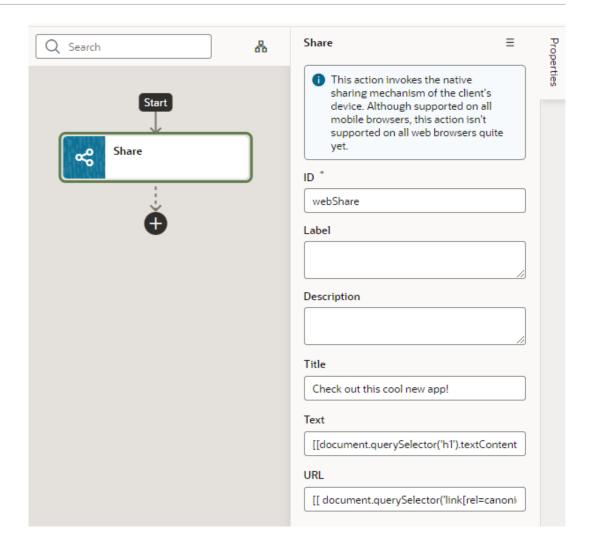
- 1. Open the Actions editor, for example, at the page level.
- 2. Create an action chain, or open an existing action chain to add the action in the editor.
- 3. Drag **Share** from the Actions palette into the action chain.

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.
 - 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 Take Photo Action

The Take Photo action, used to access the camera or the image gallery on the device where your application is installed, is deprecated. Use the JET file upload component, or the camera component in the Components palette which uses the JET file upload component.

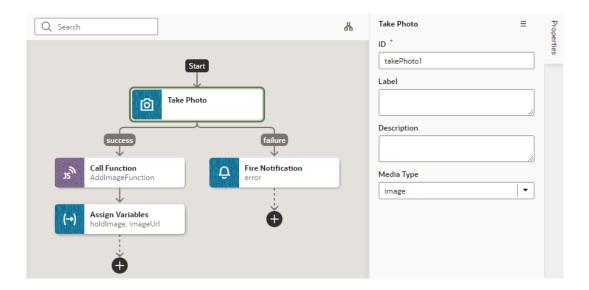
To add a Take Photo action to an action chain:

- 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.



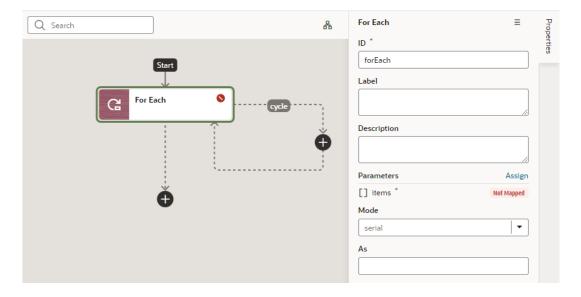
Add a For Each Action

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:

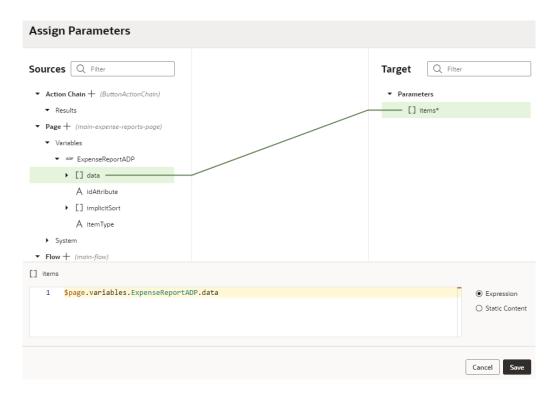
- 1. Open the Actions editor, for example, at the page level.
- 2. Create an action chain, or open an existing action chain to add the action in the editor.
- 3. Drag For Each from the Logic section of the Actions palette into the action chain.

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.

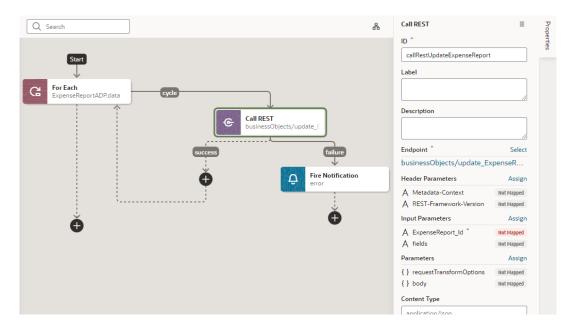
b. Click **Assign** next to Parameters to set up an expression for the items parameter that evaluates to an array, for example, \$page.variables.ExpenseReportADP.data:



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.



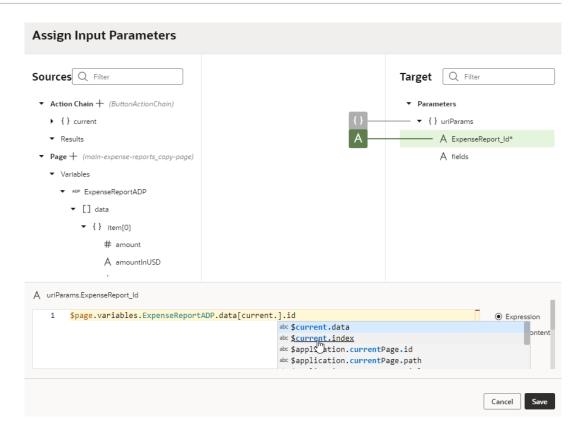


When assigning the results of the REST call to a variable, you can use the following parameter expressions for the called action:

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:

```
A uriParams.ExpenseReport_Id

1  $page.variables.ExpenseReportADP.data[foo.].id

abc foo.data
abc foo.index
abc $flow.info.description
abc $flow.info.id
```

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.

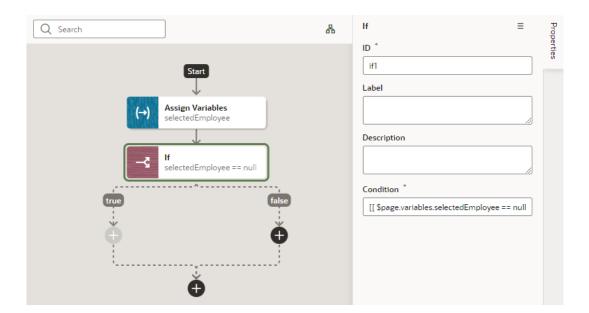
To add an If action to an action chain:

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.

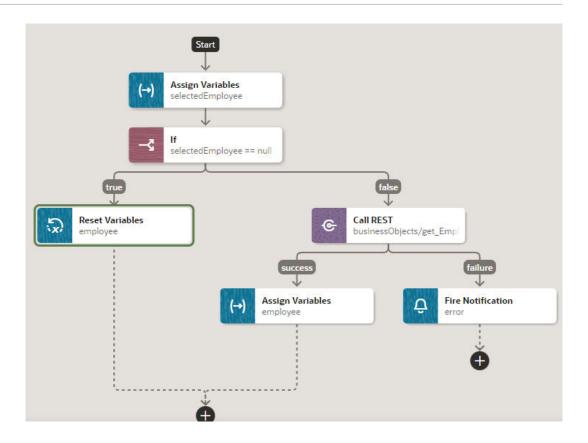
You can drag the action onto the Add icon () in the action chain, or between existing actions in the chain. The Properties pane opens when you add the action to the chain.

- 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.

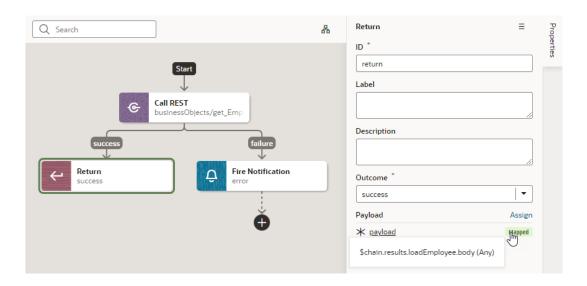
To add a Return action to an action chain:

- 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:



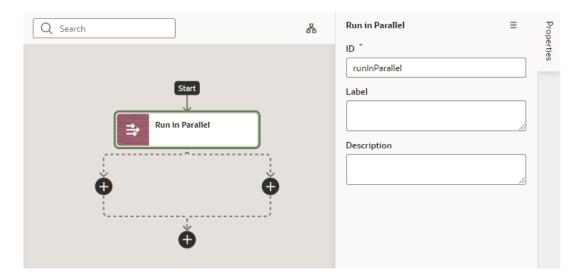
Add a Run In Parallel Action

You use the Run In Parallel action to run multiple action chain paths in parallel, wait for their responses, and produce a combined result.

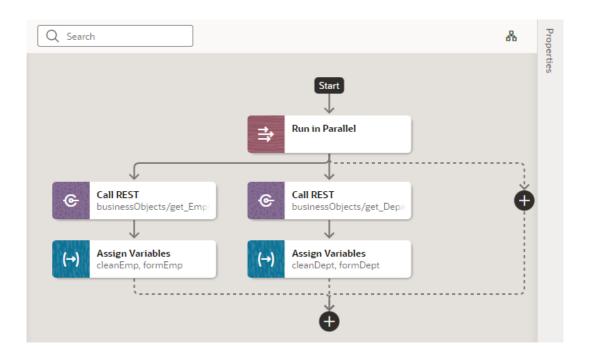
To add a Run In Parallel action to an action chain:

- 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:



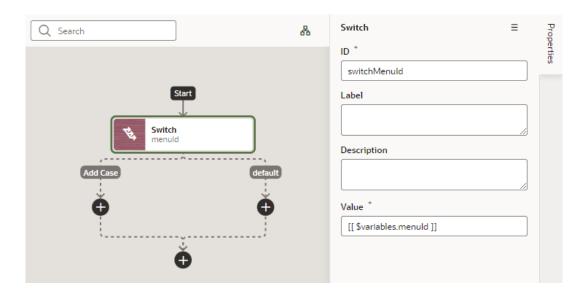
Add a Switch Action

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.

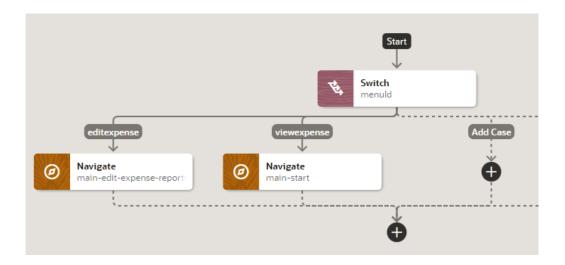
To add a Switch action to an action chain:

- 1. Open the Actions editor, for example, at the page level.
- 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:





- 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.



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 App UI 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.
 - App UI: Enables navigation to a flow or page within an App UI, which could be the current App UI or any other App UI.
 If you're working with fragments or layouts, App UI is your only option.

Complete the following steps as it applies to your use case:

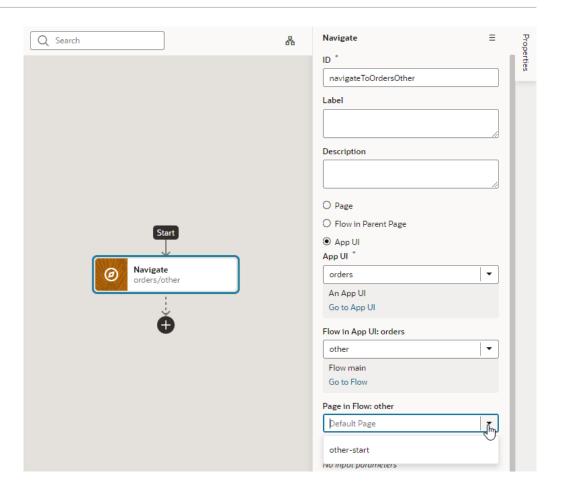
a. Select an existing page from the drop-down list of available pages, or click the **Create** link next to Page to create a new page as the target for the Navigate action.

The page you select can be in the current flow or in another flow, a different flow in the parent page, or in a flow within the current or another App UI. When navigating to another App UI, valid targets are:

- The default page of the App UI, or
- A page in the App UI that is externally accessible. To do this, navigation to that
 App UI must be enabled in the Settings editors, wherein the page-level Let other
 App UIs navigate to this page setting and the flow-level Let other App UIs
 navigate to this flow setting is selected for the page as well as the flow
 containing the page.

Here's an example of what you might see when navigating to a page in another App UI:





Note though that it's simpler to use the **Flow in Parent Page** option (instead of App UI) when you want to change the content of the flow in the current page, for example, to navigate from myAppUI/flowA/page to myAppUI/flowB/otherPage.

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.
- 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.

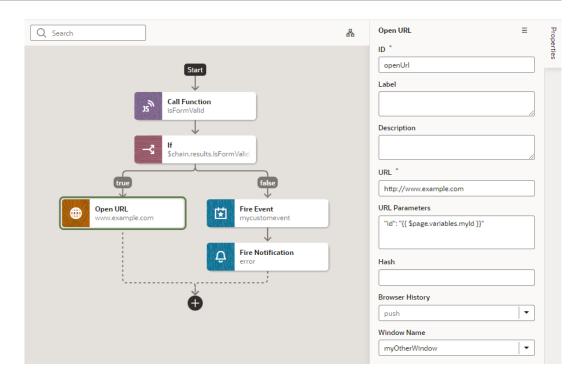
To add an Open URL action to an action chain:

- 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.
- 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 in the global Resources folder, which means they can be available to all the App UIs within your extension. For details about how to make a custom action available to other App UIs, see the end of this section, Add the Metadata.

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.

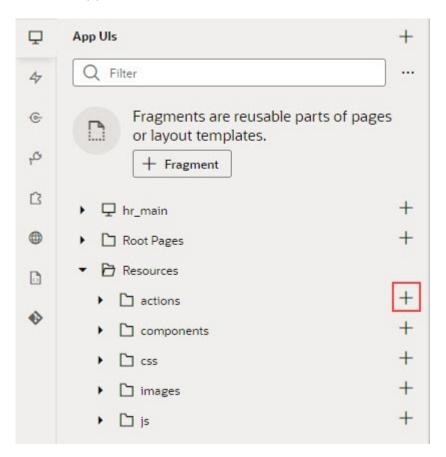
Here's an overview of what's required to create a custom action:

- 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 **App UIs** tab, expand the global **Resources** node, then click the Create Custom Action icon (+) next to the **actions** node:

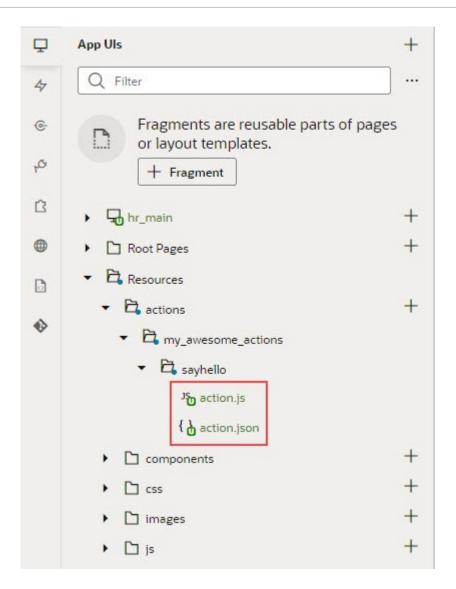


2. In the ID field, enter the name of the action group folder for your new custom action, followed by a forward slash and the name of your new action, as in: <action-group>/ <action-name>

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:



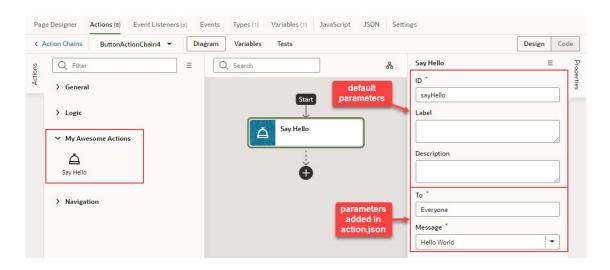


Add the Metadata

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:

```
1
               2
                     "id": "demo/sayHelloAction",
                     "idPrefix": "sayHello",
               3
                     "category": "My Awesome Actions",
    basic details
                      "defaultParameters": {},
                     "description": "Say hello to with a message",
               6
               7
                     "displayName": "Say Hello",
               8
                     "helpDescription": "Blah",
                      "iconClass": "oj-ux-ico-hospitality",
               9
                      "resultShape": {
              10
   return object
                       "result": "string"
              13
                      "propertyInspector": [
              14
                          "name": "to",
              15
                          "label": "To",
              16
                         "type": "string",
input parameters
                          "component": "inputText",
                          "placeholder": "To whom",
              19
                         "required": true
              20
              21
                       },
              22
              23
                          "name": "message",
                          "label": "Message",
              24
                          "type": "string",
              25
                          "component": "comboboxOne",
              26
                          "placeholder": "Hello, World!",
              27
              28
                          "required": true,
                          "options": [
              29
                            { "label": "Hello World",
              30
                              "value": "Hello, World!"
              31
              32
                           },
                            { "label": "Bye World",
              33
                              "value": "Bye, World!"
              34
              35
                           },
                            {"label": "Greetings World",
              36
                             "value": "Greetings, World!"
              37
              38
              39
                          "supportExpression": true
              40
              41
              42
              43
```

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.

If you want your action to be available to other App UIs, add the referenceable property to the action.json file and set its value to extension. For any new extension based on your application, the custom action will automatically appear in the dependent App UI's Actions editor.

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
		<u> </u>
"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, <i>Custom</i> .
"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 (namevalue 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	Require d	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.

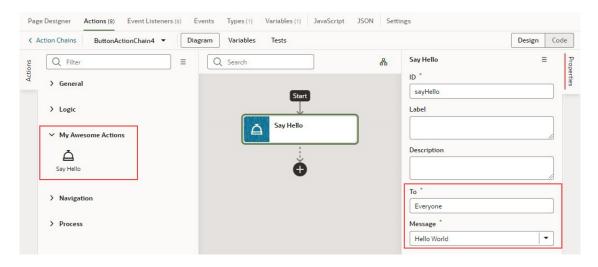
To provide the code for your action, use the perform() method, which receives the input parameter, parameters. The input parameter contains the values for the action's input parameters, as entered in the Properties pane.



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:

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 <code>sayHello</code> 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 VB Studio where the custom action's code file is stored, by adding a requirejs property to the app.json file. Here are the steps:

- 1. In the Navigator, on the **App UIs** tab, select the App UI, then select the **JSON** tab to open the app.json file.
- 2. Specify the path to the custom action's code file by adding a requirejs property in app.json and naming the path, using this format:

```
"<custom-action-ID>": "ui/self/resources/actions/<action_group>/<action_name>/action"
```

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": "ui/self/resources/actions/my_awesome_actions/sayhello/
action"
    }
},
```

Test Action Chains

VB Studio 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.



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.

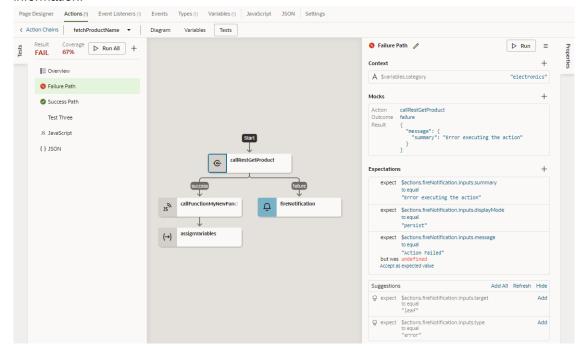
Access this editor from the **Tests** tab in the Actions editor of an individual action chain. The first time you access the Tests editor, click the **+ Test** button to create a test. The test name defaults to Test 1; specify an alternative name, if you want.

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. VB Studio 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.

The following image shows three tests defined for an action chain that fetches product information:





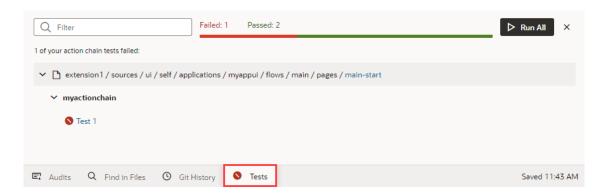
Two of these tests have run (Failure Path and Success Path), one has not (Test Three). The red icon beside the Failure Path test indicates that it failed; the reason for the failure is also marked red in the expectation for the fireNotification message, where the test author set the expected outcome 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 Your App UI

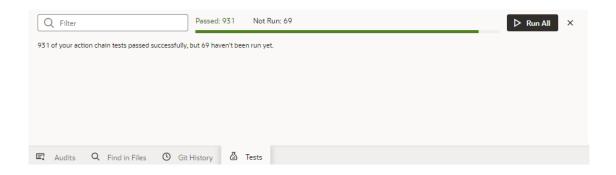
When you define multiple tests for each action chain in an App UI, it might be easier to manage tests at the App UI level. You can do this using the **Tests** tab at the bottom of your browser.



This aggregate view helps you get a quick look at the status of all action chain tests in an App UI. 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 App UI 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 App UI, and the results are saved. So if you are working on an App UI, 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 VB Studio 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.

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 <code>vbEnter</code> is fired to load a page. Or, use the <code>onValueChanged</code> 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 <code>ojAction</code> 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 <code>ojAction</code> 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.



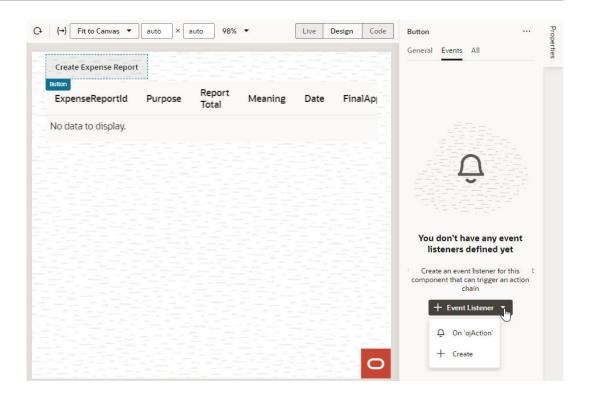
You can add an event to a component only from the component's Properties pane. You can't create one in the Events tab of pages.

To start an action with a component:

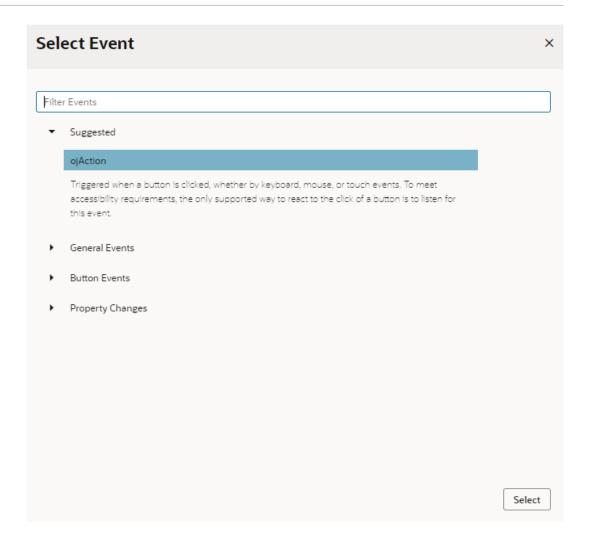
Select the component in a page or layout.

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.

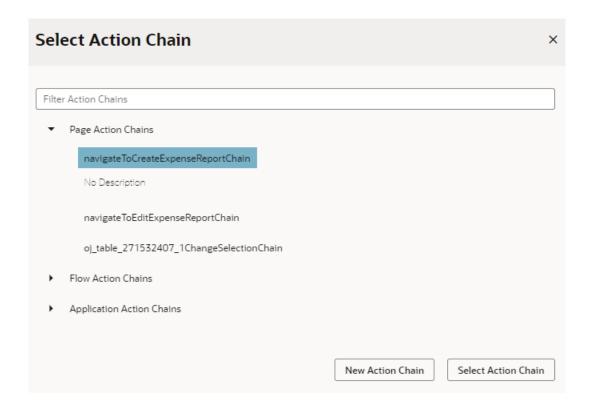




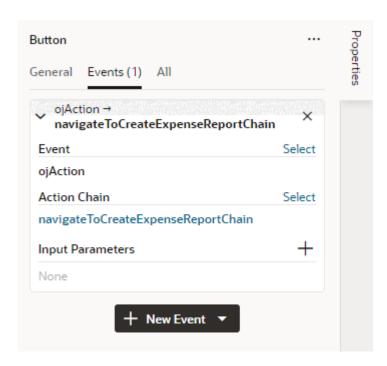
- 2. In the component's Events tab in the Properties pane, click **+ Event Listener**. You can choose the suggested event as a quick start or you can create a custom event to use a different event.
 - When you add the new event using the quick start, an action chain is created for you and the Action 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**.



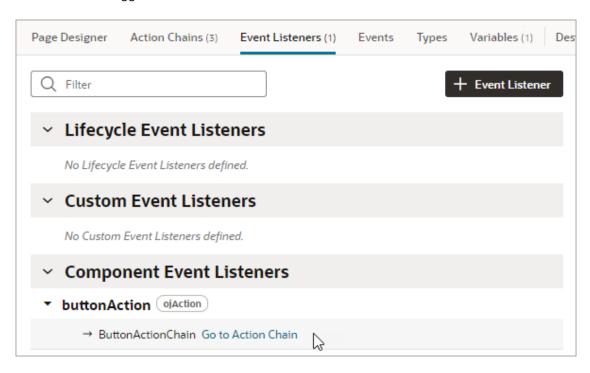
4. Select the action chain you want the event to trigger and click **Select Action Chain**. Alternatively, click **New Action Chain** to create a new action chain.



The Events tab in the Properties pane shows events on the component that VB Studio 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.

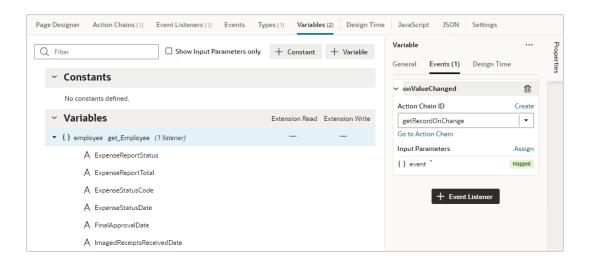
When you use an onValueChanged event to trigger an action chain, the trigger has the payload of the variable's old and new values. For example, let's say you changed the name property of an Employee and then reset the Employee; the framework will send an event that the Employee changed, and as part of the payload indicate that the name has changed.

To start an action chain when the value of a variable changes:

- Open the Variables tab of an artifact.
- 2. Select the variable in the list, then open the **Events** tab in the Properties pane.
- Click + Event Listener in the Events tab.
- 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 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.

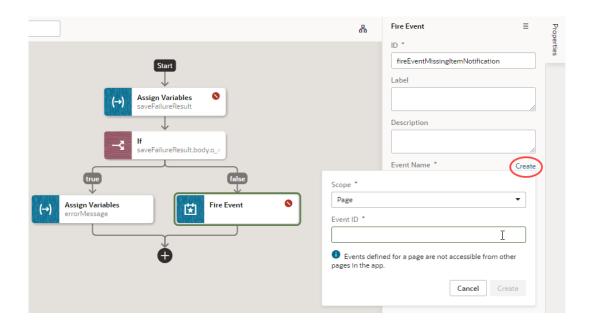
Each custom event has a Behavior property, which you can use to set whether action chains run serially or in parallel. The default behavior is "Notify", which allows the action chains to run in parallel. For more about setting an event's Behavior property, see Choose How Custom Events Call Event Listeners.

To start an action chain with a custom event:

- Open the Actions tab for a page, flow, or App UI.
- 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**.
 - 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.

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 <code>vbEnter</code> 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 App UI 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 <code>vbEnter</code> 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.

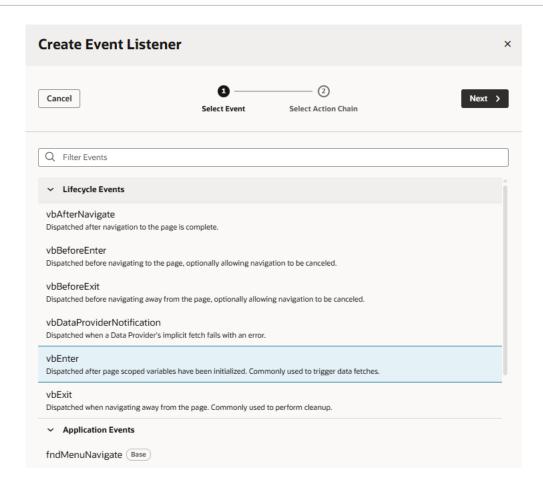
This table describes the lifecycle events you can use to start action chains:

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 App UI 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 App UI. Commonly used to fetch data.		
	 For this event, you can use these variable scopes to get data: \$application: All App UI 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 App UI. The event's payload (\$event) is an object with the following properties: currentPage <string>: Path of the current page</string>		
	previousPage <string>: Path of the previous page</string>		
	• currentPageParams <0bject>: Current page parameters		
	• previousPageParams <object>: Previous page parameters</object>		

To start an action from a lifecycle event:

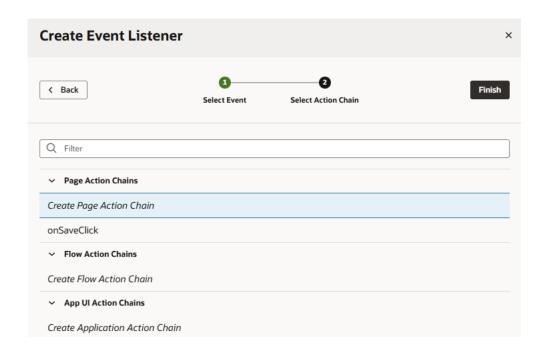
- 1. Open the Event Listeners tab for the page containing the event you want to trigger an action chain for.
- 2. Click + Event Listener.
- 3. In the Create Event Listener wizard, expand the Lifecycle Events category and select the event you want to trigger an action chain for. Click **Next**.





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 App UI.

If you want to create a new action chain now, you can click $^{\bigoplus}$ 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.

21

Work With Events and Event Listeners

Events provide the mechanism that triggers action chains when users interact with your App UI. For example, when someone clicks a button to navigate to the App UI's home page, an event triggers the navigate action. Or when a page is opened, a <code>vbEnter</code> lifecycle event might be raised to fetch data before the page loads.

There are many types of events, and all are used to execute action chains in an App UI. Your App UI reacts to events through event listeners, which declaratively specify the action chain to run when an event occurs.

Define Events in Your App UI

An event occurs when something happens in your App UI. 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 available for an App UI 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 Create a custom event in the Action chain. It can be triggered by a Fire Event Chains editor or in the Events editor, action in an action chain, or by using an then create an event listener for your event helper's fireCustomEvent() custom event in the Event Listeners editor. See Start an Action Chain By method in a module function (JavaScript Firing a Custom Event. function). See: Module Function Event Helper 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 (eventI d)Liste ner, in the same scope as your event.

Type of Event Description **How to Define** Lifecycle events Predefined events that are automatically Create an event listener for a lifecycle triggered during a page's lifecycle: event in the Event Listeners editor. See Start an Action Chain From a Lifecycle vbBeforeEnter is triggered before Event. 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 Studio 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. Other predefined events include: vbNotification: Triggered by the Fire Notification action to display a notification message in the browser. vbDataProviderNotification: Triggered by data providers to notify users of catastrophic errors that may occur when something goes wrong during an implicit fetch. vbResourceChanged: Triggered when an application has been updated to notify the user that they need to refresh to view the updated application.

Events in fragments work much the same as within a page, flow, or application—however, not all events supported within a page, flow, or application scope are available within a fragment. See Define Actions and Events in Fragments.

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:

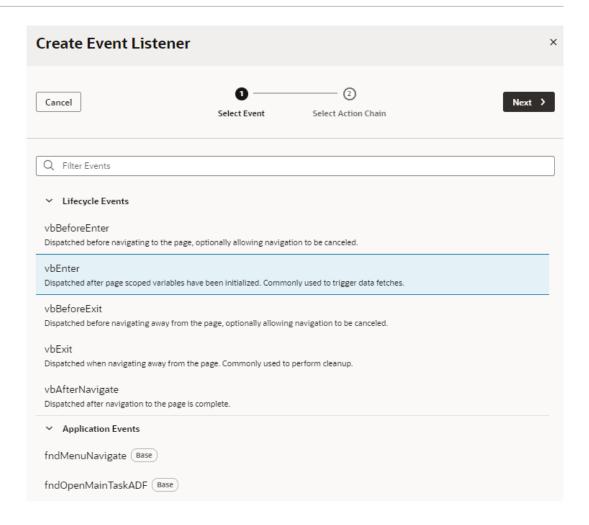


Tip:

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 <code>(eventId)Listener</code>, in the same scope as the event. Note that this option isn't available for custom events that fire to containers.

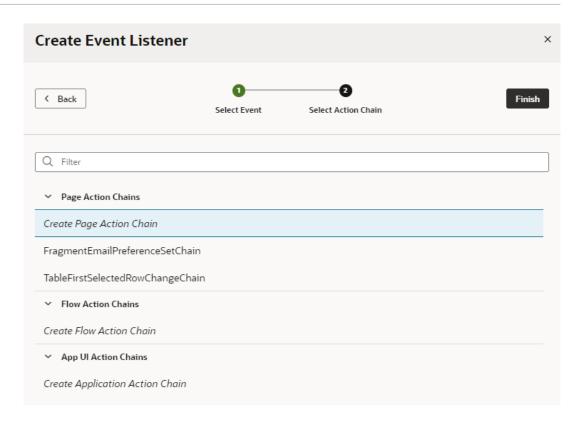
- 1. Open an artifact's **Event Listeners** tab, then click **+ Event Listener**.
- In the Create Event Listener wizard, select the event you want to trigger an action chain.
 Depending on where you're creating the listener, your list might include page events, flow
 events, and App UI events, in addition to lifecycle events.





Click Next.

4. Select the action chain you want to trigger, or create a new action chain which you can edit later.

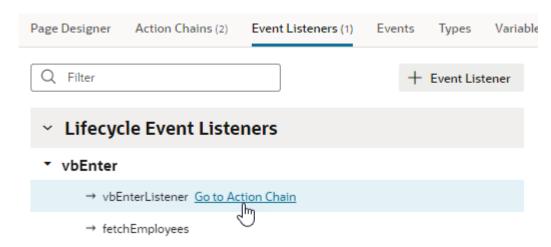


While an event listener can trigger multiple action chains, you can add only one action chain at a time in the wizard.

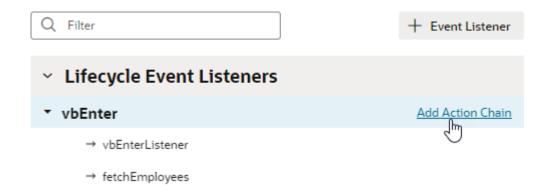
Click Finish.

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.

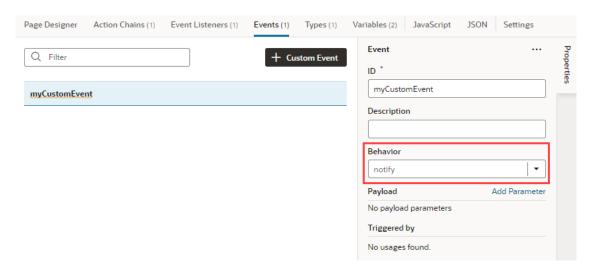
Choose How Custom Events Call Event Listeners

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.
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.

Enable Asynchronous Handling for Component Events

Some JET components, particularly editable tables, support asynchronous event handling—but you must explicitly enable this behavior for events in your VB Studio application. If you

enable this behavior on existing event listener configuration, make sure you rework your existing action chains to manage the async event.

Here's a scenario when you might want to enable async event handling: When working with editable tables, users often edit a single row (data) at a time, with a currentRow variable being used to track the row being edited. When a user moves out of one row and into the next, the application is notified of the change through the table's events: ojBeforeRowEdit and ojBeforeRowEditEnd.

When these events use non-async event behavior, the user may see data from the previously edited row (previous row) flash briefly in the new row that is being entered (next or new row). This occurs because of a split-second delay between the previous row going out of edit mode and the next row going into edit mode, both of which happen in parallel as they both use the currentRow variable. This delay—wherein the next row goes into edit mode on the UI, faster than the completion of the action chain populating currentRow with the next row's data causes the previous row's data to briefly display in the next row.

Async handling of these events can resolve this issue, by allowing VB Studio to provide a Promise that instructs the table to display a loading indicator while waiting for the Promise to resolve (or be rejected), until all action chains linked to the event have completed. Until then, the next row will not go into edit mode.



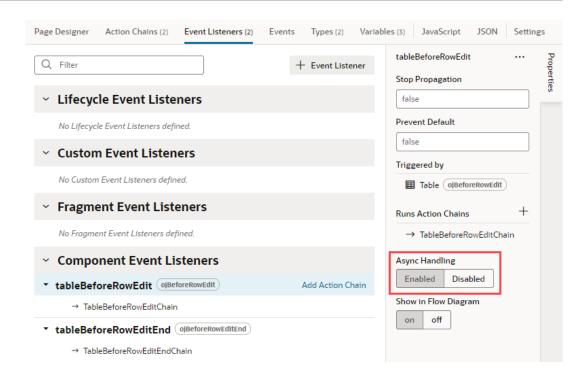
Caution:

Converting an existing event configured with non-async behavior may have unintended consequences when switched over to async behavior if the corresponding action chains are not reworked. It's important to review JET event documentation for components that supports async events (for example, the table component's oiBeforeRowEdit and oiBeforeRowEditEnd) to understand the implications of enabling async handling and to help implement your action chain logic suitably.

To enable async handling:

- Open the Event Listeners editor and select the event under Component Event Listeners. for example, tableBeforeRowEdit.
- 2. In the event listener's Properties pane, toggle Async Handling from Disabled (default) to Enabled.





Raise Fragment or Layout Events that Emit to the Parent Container

Layouts and fragments defined in your App UI 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 the example of a fragment that 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 event listenable by a layout or fragment's parent container (say, a page or a dynamic container), you'll need a Fire Event action in your action chain to emit the event's payload to the parent container. This makes it possible for the page (or container) 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 App UI.

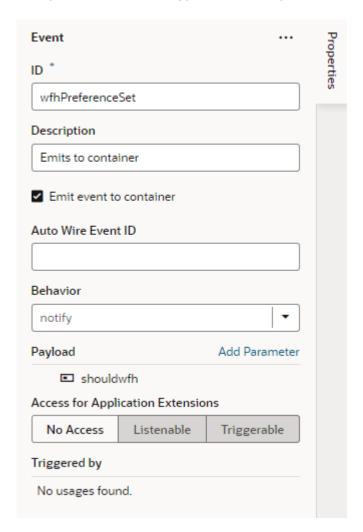
To make a layout or fragment event listenable on its parent container:

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, wfhPreferenceSet), then select the option to emit the event's payload to the parent container:
 - For a layout event, select Emit event to page.
 - For a fragment event, select Emit event to container.

Click Create.

- c. In the event's Properties pane, 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 is a boolean-type shouldwfh parameter:



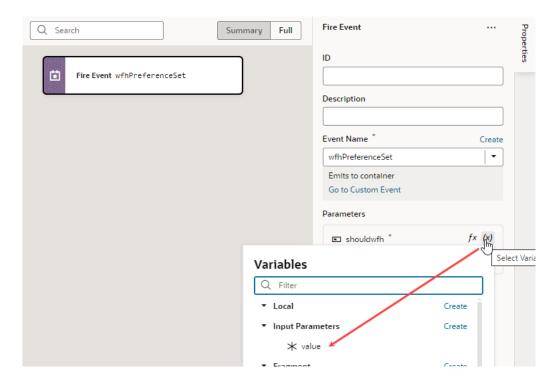




Tip:

When working with fragment events, you can choose to enter an **Auto Wire Event ID** to simplify the process of creating a listener for this event on the parent container. This option enables the event to be listed under Fragment Events when you create a listener for the event from the Event Listeners tab on pages or layouts that use the fragment. See Automatically Wire a Fragment's Custom Event to the Parent Container.

- 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, wfhPreferenceSet).
 - d. To pass in the payload parameter's value, click (x) next to shouldwfh under Parameters and select value under Input Parameters in the Variables picker.



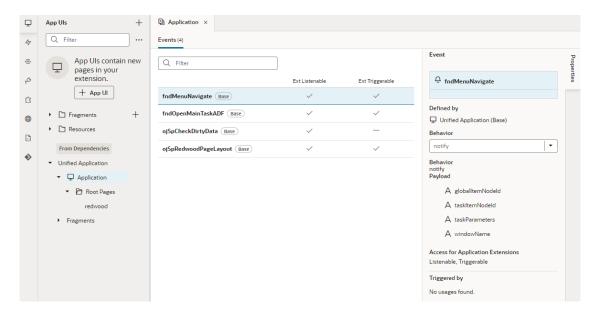
- 3. On the page that uses your layout or fragment, configure the layout or fragment's parent container to handle the custom event:
 - a. Go to the Page Designer and select the component (for example, a fragment) to open its Properties pane, click the **Events** tab, then click + **Event Listener** and select the suggested custom event (for example, wfhPreferenceSet).
 - b. Build the action chain that must be triggered when the event occurs. For example, you might want a notification to appear on the page when a user toggles a Switch in a

fragment. For a sample end-to-end scenario that illustrates this workflow, see Create Custom Events that Emit to a Fragment's Parent Container.

Use Events Defined in the Unified App

Because the Unified Application is considered a dependency for all extensions, every App UI in an extension can use global events defined in the Unified App to start action chains. For example, you can use the <code>fndMenuNavigate</code> event to trigger navigation from a page in your App UI to the Ask Oracle landing page.

To see the global events available to your App UI, click **Application** under **Unified Application** in the Navigator, then click the **Events** tab. Selecting an event in the Events tab shows details about the event in the Properties pane, such as where it is defined and what triggers it.



How you can use a global event in your App UI is defined by its designation: *Triggerable*, *Listenable*, or both.

Event Designation	Description
Triggerable	If an event is designated as Triggerable, you can fire the event from an action chain in your App UI using the Fire Event action.
Listenable	If an event is designated as Listenable, you can add an event listener to your App UI and configure it to trigger your action chain when the listenable event occurs.

In the preceding image, the <code>fndMenuNavigate</code> event is badged with base, and it's designated as listenable and triggerable, meaning you could add an event listener to your App UI that listens for the <code>fndMenuNavigate</code> event as well as use the Fire Event action in an action chain to call the event from your App UI.

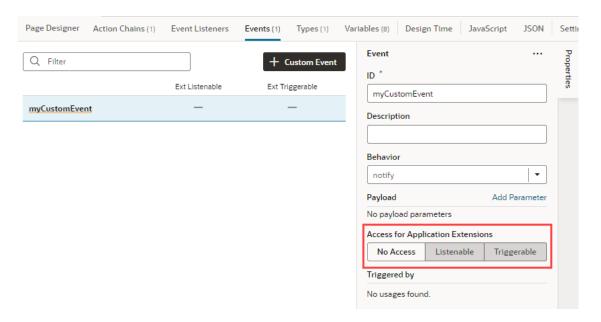
You can't change the designation of custom events defined in the Unified App, but you can use them to start action chains in the Unified App or in your App UI. For example, if you used the Fire Event action in an action chain to trigger the fndMenuNavigate event, any time

fndMenuNavigate is fired in your App UI, the Unified App listening for this event triggers the action chain to navigate from your App UI to the Ask Oracle landing page.

Make an Event Available to Extensions

If you want to let others extend your App UI's functionality, you can mark your custom event as available to extensions. This lets someone add a dependency on the extension that contains your App UI, so they can listen to and/or trigger the event.

To mark a custom event as available to extensions, go to the Events tab of a page or layout in your App UI, select the event and designate it as Listenable, Triggerable, or both in the **Access for Application Extensions** property in the event's Properties tab:



Note:

After you've made an event accessible to extensions, you should avoid renaming its ID. Renaming an ID might break the extensions that use it.

Event Designation	Description
Listenable	Click Listenable to let others start action chains defined in your App UI. When an event is designated as Listenable, other developers can add an event listener to their App UI and configure it to trigger your action chain when the listenable event occurs.
Triggerable	Click Triggerable to let others start action chains defined in your App UI using the Fire Event action. When an event is designated as Triggerable, others can use the Fire Event action to call the event from an action chain in their App UI.

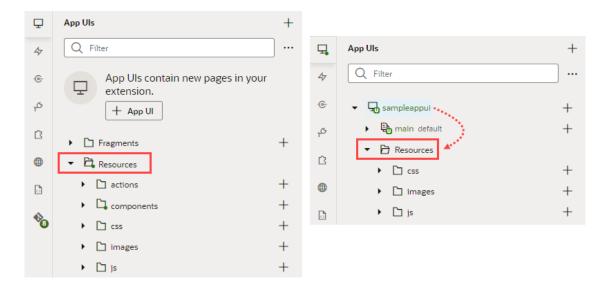
Triggering an event defined in an App UI might mean that you are also triggering an action chain or some function defined in the original App UI.

Work With Resource Files

As you develop an App UI, you can import and export static resource files for use in your App UI's pages.

Resources are typically files that you import to support or add functionality to pages in your App UI. 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, your extension includes a global **Resources** folder (shown on the left in the image) to store custom components, images, external JavaScript files, and other resource files that can be used by all App UIs in the extension. In addition, each App UI contains a **Resources** folder (shown on the right) to store resources that can be used in that App UI's pages alone.



To reiterate, global resources are accessible to all App UIs in the extension, but a particular's App UI's resources are available only to that App UI.

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 extension. Right-click the folder to either create or import a custom action.

compon Location for custom web components that are installed to an extension. Web components are reusable pieces of UI code that you can embed as custom HTML elements (by clicking Get Components under Custom in the Components palette or the Components tab in the Navigator).



Importing web components to the resources/components folder or creating them there makes them a part of your extension. 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 for your App UI. 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 if you have multiple App UIs that use the same components. Talk to your administrator for site-specific information on how to publish these components externally.

css	Location for stylesheets linked from pages in an App UI when you want to add custom styling to page elements.
images	Default location for custom images that you want to add to an App UI's pages.
js	Location for external JavaScript files that you want to use in an App UI's pages.

Import Resources

To add resources that you want to use in an App UI, you import individual files or ZIP archives by using the import option available when you right-click a resources folder or any artifact, or by dragging the file directly from your local file system onto a folder in the navigator.

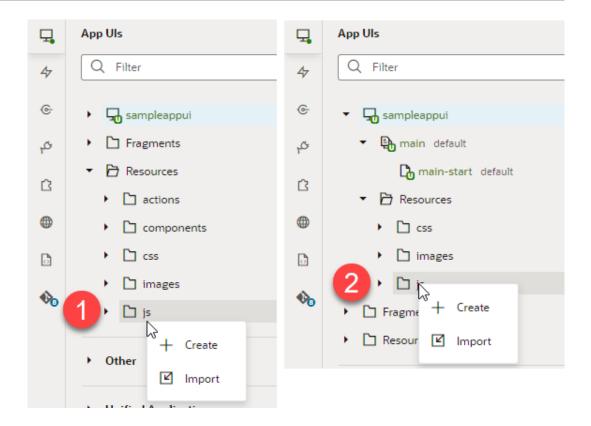
You can import resource files to the Resources folder, its subfolders, or any other artifact in the tree view (though not pages). The location you select will determine the scope of the resources you import. The import location is also important to make sure you're importing the resource file where you can access it. For example, it's possible to import an image into the flows folder, but the expected location for an image is an images folder under Resources. 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 App UI:

- In the App UIs pane, locate the folder or artifact where you want to import the resource file.
- Right-click the folder or artifact and choose **Import**.

For example, here's a combined screenshot showing the Import option when you right-click the js folder under an extension's global **Resources** folder (Label 1 in the image) and the js folder under the sampleappui's Resources folder (Label 2):





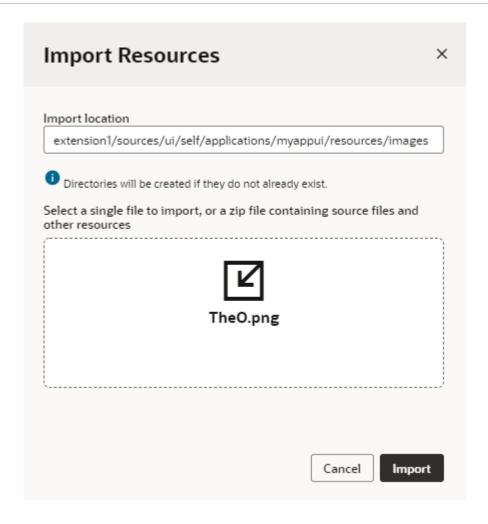
Alternatively, drag a file from your local file system onto the folder or artifact in the tree view to open the Import Resources dialog box.

Note:

It's not recommended that you actually refer to imported css files in your App UIs at this time, as doing so can produce unpredictable results.

3. In the Import Resources dialog box, choose the file or archive with the resources you want to import. You can drag the file into the drop target area or click the drop target area to navigate to the file on your local system.





Optionally, edit the path in the Import location field to create new folders.

Click Import.

A confirmation appears and your resource file is added at the location you specified.



Tip:

To import an extension's resources into another extension or to share them with a team member, export the extension's resources as a ZIP archive.

Work with the Image Gallery

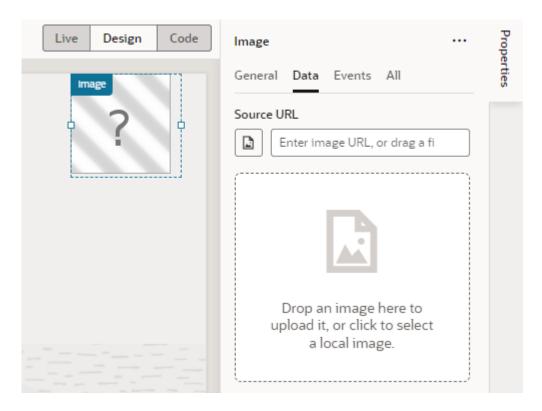
While it's possible to import image resources, then associate them with image components on an App UI's page, you can do both at one shot using the Image Gallery, which is accessible from the Data tab in the Properties pane when an image component is selected on the canvas.

The Image Gallery lets you view and manage images available for use in an App UI's pages—but it only displays the images that are added to an images folder under **Resources** either for a particular App UI or for the extension. Images stored in other locations won't show in the Image Gallery.

You can use the Image Gallery to import and apply an image to a UI component, or to select a previously imported image for a UI component.

To work with images in the Image Gallery:

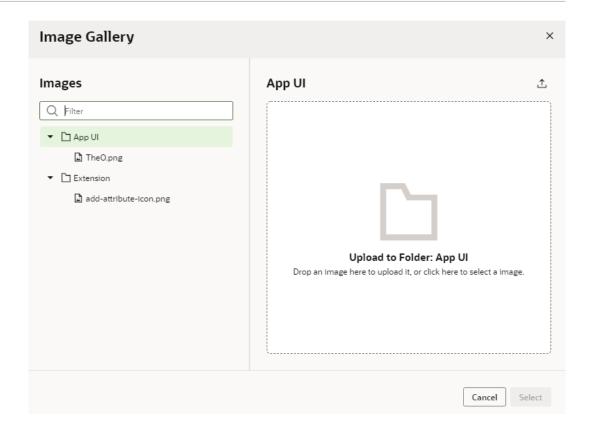
- 1. Open a page in the Page Designer and select an image or avatar component on the canvas.
- 2. Click the **Data** tab in the component's Properties pane.



The Data tab displays a Source URL field for the path to the stored image. This field is empty when an image has not yet been defined.

You need to access the Image Gallery to add images to the App UI or extension's <code>images</code> folder. If you were to drag an image into the drop target area in the Data tab, it will be added to the App UI's <code>images</code> folder (and the path to the image will be specified in the Source URL field), but the Image Gallery won't open.

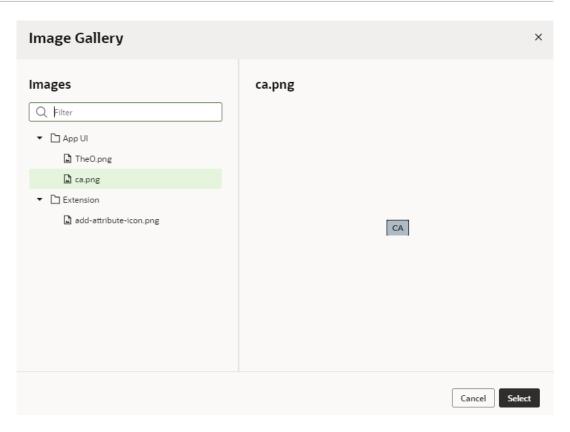
3. Click the Image Gallery icon (\blacksquare) under Source URL to open the Image Gallery.



4. If the image you want to apply to the UI component was previously imported and exists in the Image Gallery, select it in the Images panel, either under the App UI or Extension folder. If the image doesn't exist, select the folder where you want to add the image, then drag it into the drop target area. You can also click the drop target area and select an image from your local file system.

You can add as many images as you want, but you need to import them individually.

5. When you have your image, click **Select**.



The path to the image will be added to the image or avatar component's Source URL in the Data tab. If your image is stored under the App Ul's resources, you'll see something similar to [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources, you might see [[α] if it is stored as part of the extension's resources.

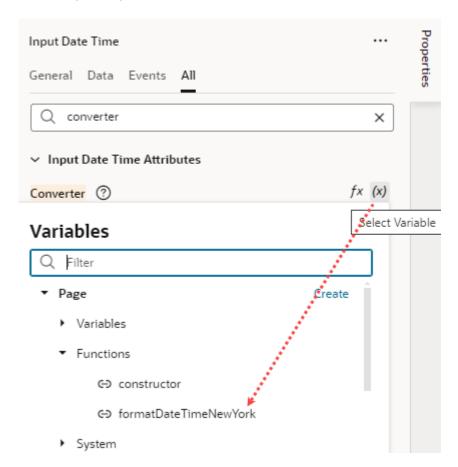
Create Declarative References to Imported Resources

You can import resources such as custom component, CSS files, and modules, to create declarative references to imported resources.

The **Imports** tab in the Settings editor enables you to manage resources imported within the scope of an App UI, a flow, or a page artifact, even a dynamic layout. You can manage custom components and CSS files, as well as modules containing code that you want to call in your App UI. Let's consider some scenarios of when you'd want to use this Imports tab:

- When your artifact includes components that are deprecated or no longer used, these
 component definitions stay intact in the artifact's metadata, but they might be flagged by
 audits as a deprecated or unused component dependency. While you can resolve this
 issue manually by editing the entries in 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 need to add an import
 statement manually to the page's JSON. But referencing custom CSS files on the Imports
 tab automatically adds an import statement to the JSON file. This way, you get to apply the
 imported CSS files to any page or pages in a flow without having to manually update
 JSON.
- When you want to use JavaScript modules 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 App UI's pages 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 to be used in an App UI:

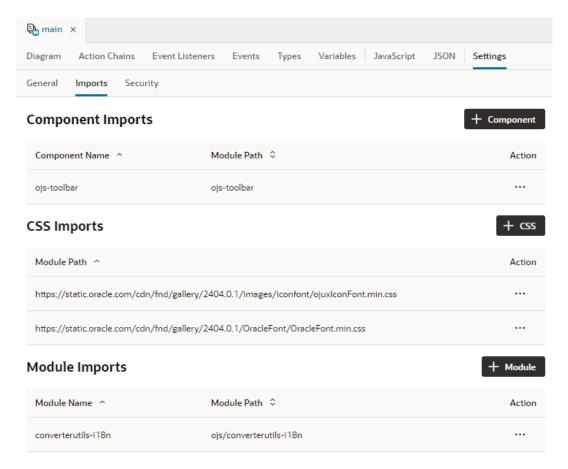
- Open the Imports tab under Settings for an App UI, a flow, page artifact, or dynamic layout. Imports at the App UI level can be shared between flows and pages in your App UI. Imports at the dynamic layout level can be shared between dynamic layout templates.
- 2. Import custom components, CSS, and modules:
 - To import custom components, 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 App UI's flows and pages or dynamic layouts, 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 App UI's resources, see Import 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 with code you want to call in your App UI's flows and pages or dynamic layouts, 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, which lets you use those resources in all pages within the flow:



3. Click Create or Create & New to repeat the action.

Work With Code

Most application development in VB Studio is visual and declarative. Sometimes though you might want to update your App UI's source code or add custom code to make your App UI richer—and code editors in the Designer let you do just that.

The JavaScript, JSON, and Code view editors give you direct access to the code created by visual tools when you develop your App UI. For example, when designing a page, you can choose to directly edit the source code of the 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.
JavaScript	Displays JavaScript functions at the App UI, 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 App UI, flow, or page level, as well as at the layout and fragment level. See Work With JSON.

VB Studio'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 App UI'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 <code>module</code>, which can be at the App UI, flow, or page level. An <code>AppModule</code> contains one or more flows, each with its own <code>FlowModule</code>, and each flow can have one or more pages, each with its own <code>PageModule</code>. 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 page-level 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.

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.



Caution:

When coding with JavaScript, it's your responsibility to follow secure coding standards as applicable. Make sure you follow best practices and actively take steps to keep credentials, certificates, and other sensitive details safe by not coding them in JavaScript or making them available on the client side.

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 App UI. You can also do this for layouts and fragments.

 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 App UI level, select the App UI node, then click the **JavaScript** tab:

```
¬ myappui ×
          Action Chains
                                                             Variables
                                                                        JavaScript
                                                                                     JSON
                                                                                             Settings
Diagram
                          Event Listeners
                                                    Types
       define([], () => {
  1
  2
          'use strict';
  4
         class AppModule {
  5
         }
  7
         return AppModule;
  8
       });
```

JavaScript functions are defined in different files based on the artifact's scope. An App UI uses the app.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:

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:

```
1
      define([], () \Rightarrow {
2
        'use strict';
 3
 4
        class PageModule {
 5
 6
          /**
7
           *
 8
           * @param {String} arg1
 9
           * @return {String}
10
          SayHello(arg1) {
11
12
            for
13
                == for
14
                for
15
                   foreach
16
        return
                forin
17
      });
                forof
18
                [@] FormData
                [@] FormDataEvent
                [@] FinalizationRegistry
                [@] Float32Array
                [@] Float64Array
                [ FileSystemDirectoryEntry
                [ø] FileSystemDirectoryReader
```

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.

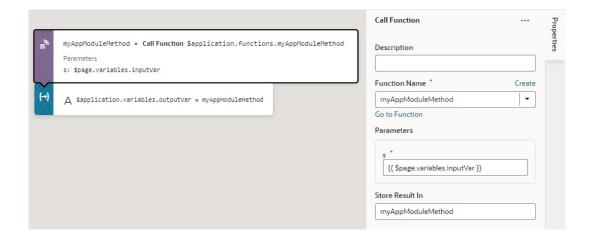
For help with JavaScript syntax, see https://developer.mozilla.org/en-US/docs/Web/JavaScript#reference. These additional resources can be helpful as well:

- 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:

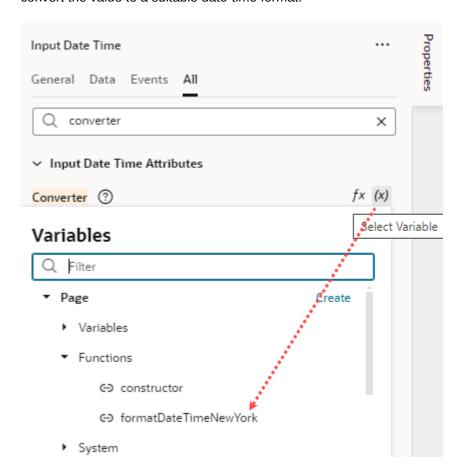
In an action chain, select the function in the Call Function action.

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 To 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 VB Studio 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 App UI'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 App UI.



JS libraries can be imported at the App UI level or at the extension level. For typical usage within an App UI, import your JS library to the resources directory at the App UI level (extension1/sources/ui/self/applications/ <app-ui-id>/resources). If you want to use external JS libraries with custom web components and fragments, import your JS library to the resources directory at the extension level (extension1/sources/ui/self/resources/).

- a. In the App UIs pane, go to the **Resources** folder, right-click **js**, and click **Import**.
 - If you're using Source view, go to extension1/sources/ui/self/applications/<app-ui-id>/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.
- Select the archive you want to import and click Import.
- 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 App Ul'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 Create Declarative References to Imported Resources.

Use RequireJS to Reference Third-Party JavaScript Libraries

If you want to use third-party JavaScript libraries in your App UI, you can import the library and add a requirejs statement to your App UI's definition.

JS libraries can be imported at the App UI level or at the extension level. For typical usage within an App UI, import your JS library to the resources directory at the App UI level (extension1/sources/ui/self/applications/<app-ui-id>/resources). If you want to use external JS libraries with custom web components and fragments, import your JS library to the resources directory at the extension level (extension1/sources/ui/self/resources/).

- 1. Open the app.json file for your App UI.
 - In the App UIs pane, select your App UI node, then click the JSON tab, or
 - In the Source view, locate the file for your App UI under extension1/sources/ui/self/applications/.

2. Add a requirejs statement to the App UI's definition. For example, if you've added gl-matrix-min.js to your App UI's resources under .../applications/<app-ui-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.

If you added a JS library (say, HumanResourcesUtils) to your extension's resources under extension1/sources/ui/self/resources/js/HumanResourcesUtils, your requirejs statement might be similar to:

```
"requirejs": {
   "paths": {
    "HumanResourcesUtils": "/ui/self/resources/js/HumanResourcesUtils"
   }
}
```

where HumanResourcesUtils is a folder containing JS files such as promotion.js, manager.js, and employee.js.

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:

Add JavaScript Modules As Global Functions

When different parts of your application routinely use similar JavaScript functions to transform or manipulate data, you can extract those functions as global functions and reuse them.

To make these functions reusable in pages, you can define their implementation as global functions and make them available in all pages (fragments or any other container), both within and across extensions.

Let's say extA defines two JavaScript utilities, dateUtils.js and standardUtils.js, shared by containers (page, fragment, and dynamic layout artifacts) in the extension. Rather

than copy the code for the utilities into every extension that requires the same logic, you can add them as global functions (at the <code>extension1/sources/ui/self/resources/level</code>) and declare rules for their usage in an associated <code>functions.json</code> metadata file. This way, dependent extensions (and their App UIs) can import the functions as resources and use them with minimal effort.

To add JavaScript modules as global functions to your extension:

1. Create an archive (for example, a functions.zip file) that contains your JavaScript files and a related functions.json file.

You can structure your files any way you want, meaning they can be a flat list of files or in subfolders of any depth, as long as the functions.json file uses the correct file paths. Here's an example of the structure you can use:

```
functions
| functions.json
| someFunction.js
| subfolder1/
| | myFunction.js
| subfolder2/
| otherFunction.js
```

Make sure your JavaScript files do not have any dependencies. See Example JS Module Defined As a Global Function.

The functions.json file defines the context name, label, description, and other necessary metadata and is required to expose the functions properly. See Example functions.json to Declare Global Functions.

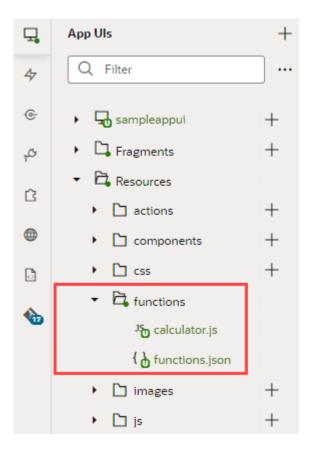
2. Import the archive to your extension at the global resources level.

Note:

Global functions can only be defined at the extension level. You cannot add them to an App UI's resources or any other folder, including those meant for the Unified Application.

- a. In the App UIs pane, right-click the global Resources folder and click Import. If you're using Source view, go to extension1/sources/ui/self/resources/ and click Import.
- b. Drag your archive (for example, functions.zip) from your local file system and drop it onto the target area.
- c. Click Import.
- 3. Once the functions folder is created in your extension, you can add or delete files as needed. Here's an example of the functions.zip imported with the calculator.js and functions.json files:

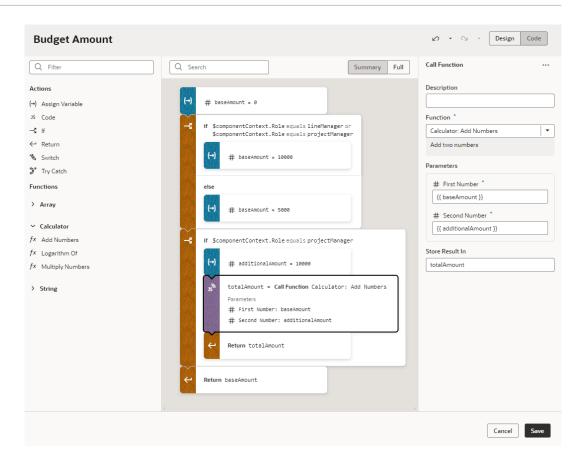




Your functions now become available to you as follows:

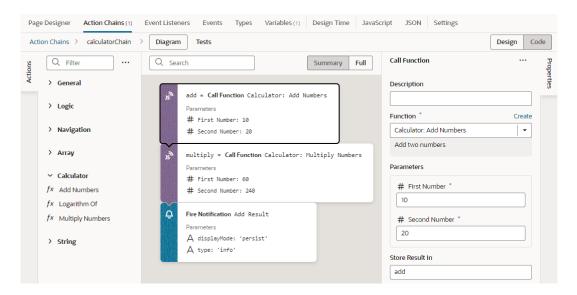
• In the Advanced Expressions editor for business rules, validation rules, and default values, where you can drag these functions from the Actions palette. To edit the code directly, use the format <code>\$modules.module-id.function-name(...)</code>, for example, <code>\$modules.calculator.add(baseAmount, additionalAmount)</code>:





For information on how to build expressions, see Build Advanced Expressions.

• In the JavaScript Action Chains editor, where you can drag these functions from the Actions palette. To edit the code directly, use the format \$modules.module-id.function-name(...), for example, \$modules.calculator.add(10, 20):



For information on how to use the Call Function action, see Add a Call Function Action.

If you get errors at runtime, make sure the JavaScript module is correctly imported to your container.

- To automatically import the module, drag the function from the palette and build your expression or action chain.
- To manually import the module:
 - 1. Open the container's **Imports** tab in the Settings editor.
 - 2. Click + Module under Module Imports, enter the module name (for example, calculator), then the path to the module in the format {extId}: {resourceType}/{resourceAlias}, where:
 - {extId} refers to the extension that the resource belongs to, either self to indicate the current extension or extensionID to refer to another extension
 - {resourceType} is a special keyword for /functions resources, for example, \$functions
 - {resourceAlias} is the JavaScript alias defined in functions.json.

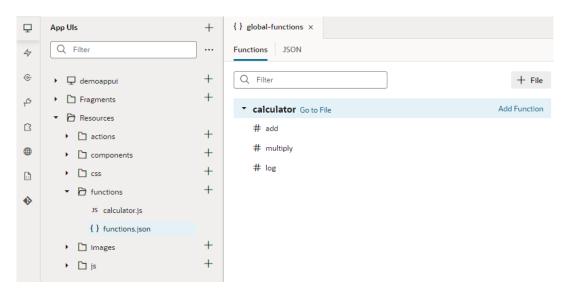
For example, the self:\$functions/calculator module path resolves to currentExt/ui/self/resources/functions/calculator.js while otherExt:\$functions/calculator.js resolves to otherExt/ui/self/resources/functions/calculator.js.

Click Create.

Manage Global Functions

Use the Functions editor to manage global functions added to your extension, where you can edit the functions.json file as well as create new JavaScript files that contain global functions.

1. In the App UIs pane, expand the global **Resources: functions** folder, then select functions.json to open the file in the Functions tab.

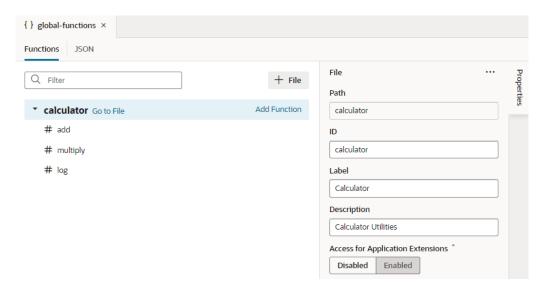


The Functions tab provides an intuitive interface where you can manage the JS files added to functions.json as well as the functions within each file. If you prefer to work with JSON, you can always switch to the **JSON** tab.

2. Add a new or existing JS file to functions.json:

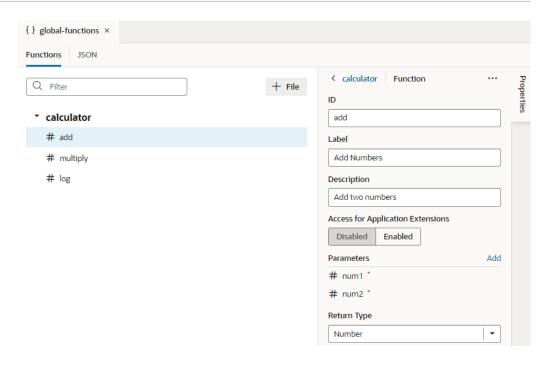


- To add a new file, click + File, select Create New File, enter a name for your JS file, and click Create. The new file is created in the Resources: functions folder and automatically added to functions.json.
- To add a file that was previously imported to the Resources: functions folder, click +
 File, select Add Existing File, then select the JS file you want to add in the Path list,
 and click Add. This option appears only if the JS file has already been imported to the
 functions folder.
- 3. Update the JS files (and functions) associated with functions.json:
 - To update the JS file's metadata, select the file (for example, calculator.js) and update its metadata in the Properties pane:

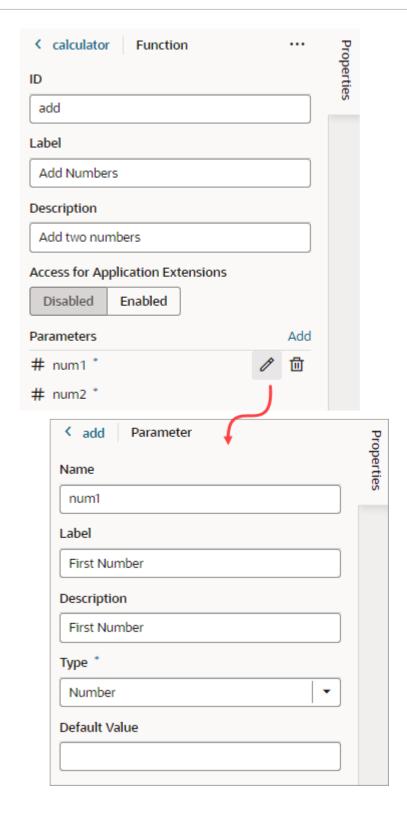


Except for the file path, you can change anything you want. See File Metadata Object for details.

- To add a function to a JS file, select the file and click Add Function. Enter an ID for the function and click Create. You can now use the Properties pane to define the function's metadata, which includes adding or editing input parameters. See Function Metadata Object and Param Metadata Object for details.
- To edit an existing function in a JS file, select the file and click **Go to File**. You'll also find this option in a file's context menu and its Action menu in the Properties pane.
- To update a function's metadata or input parameters, select the function (for example, add) and update its details in the Properties pane:



To manage the function's input parameters, hover over a parameter under **Parameters** and click \mathcal{O} to update the parameter's metadata:



Required parameters are indicated by an * in the Properties pane and are typically those without a default value. To add an optional parameter, set a default value for the parameter. If a default value isn't defined (using defaultValue in functions.json), you can set it in the parameter's Properties pane.

To add a new input parameter, click **Add** next to **Parameters**.

• To delete a JS file or a function in a JS file, right-click the file or the function and select **Delete**. Note that deleting a file only removes it from functions.json, it does not delete the file from the functions folder.

Example JS Module Defined As a Global Function

Here's an example of a calculator.js module, which defines three global functions:

```
define([], () => {
  'use strict';

function add(num1, num2) {
   return num1 + num2;
  }

function multiply(num1, num2) {
   return num1 * num2;
  }

function log(number, base=10) {
   // some calculation
   return 0;
  }
  return { add, multiply, log };

});
```

This calculator.js module defines the add function to return the sum of two numbers, the multiply function to multiply two numbers, and the log function to calculate the logarithm of a number with a specified base.

The functions.json file would include the calculator.js file in its list of JS modules in the files section.

Example functions.json to Declare Global Functions

The functions.json file is used to declare JavaScript modules as global functions and specifies the metadata of functions, such as module name, function name and parameters it takes, as well as the implementation file path. Here's an example of a functions.json file that references the calculator.js module:

```
"num1": {
              "label": "First Number",
              "description": "First Number",
              "type": "number"
            },
            "num2": {
              "label": "Second Number",
              "description": "Second Number",
              "type": "number"
            }
          },
          "return": "number",
          "referenceable": "extension"
        },
        "multiply": {
          "label": "Multiply Numbers",
          "description": "Multiply two numbers",
          "params": {
            "num1": {
              "label": "First Number",
              "description": "First Number",
              "type": "number"
            },
            "num2": {
              "label": "Second Number",
              "description": "Second Number",
              "type": "number"
            }
          },
          "return": "number",
          "referenceable": "self"
        },
        "log": {
          "label": "Logarithm Of",
          "description": "Logarithm of given number",
          "params": {
            "number": {
              "label": "Number",
              "description": "Number",
              "type": "number"
            },
            "base": {
              "label": "Base",
              "description": "Log Base",
              "type": "number",
              "defaultValue": 10
            }
          },
          "return": "number",
          "referenceable": "extension"
     }
   }
 }
}
```

File Metadata Object

The files section can include one or more JS files. Each file path (relative to dynamicLayouts/self/resources/functions/) maps to a metadata object that describes a group of functions.

Name	Туре	Description
path	string	Relative path of the implementation file. In the example above, the path to the .js file is oracle/calculator. When the file is a RequireJS module, the .js extension can be dropped.
label	string	A string naming this function group.
description	string	A string describing this function group.
referenceab le	self extension	(Optional) Whether the module is accessible to the current extension or dependent extensions. For example, the calculator module is set to "referenceable": "extensible", meaning the file is accessible to dependent extensions. If referenceable isn't defined, its value defaults to self, which means only the current extension (where the module is defined) can access its functions.
functions	Object <string, function=""></string,>	(Required) An object that maps function name to function metadata. See Function Metadata Object.

Function Metadata Object

The functions section can be used to list functions that are available to callers.

Name	Туре	Description
label	string	A string naming this function group.
description	string	A string describing this function group.
params	Object <string, function=""></string,>	(Required) An object that maps parameter name to a param object. See Param Metadata Object.
return	string	Function's return type. Only primitive types, array, object, and any are supported.



Name	Туре	Description
referenceab le	self extension	Whether the function can be referenced from the current or a dependent extension. A function can be less permissive about its access, but it cannot supersede the access set on the module. For example, the calculator module allows access to all dependent extensions ("referenceable": "extension"), whereas the multiply method only allows access to the current extension ("referenceable": "self"). This definition means a dependent extension that imports the calculator module will not be able to call the multiply function.
		Consider another example:
		<pre>"dateLocalUtils": { "path": "date/dateUtils", "label": "Date Utility Functions", "referenceable": "self", "functions": { "dateToIsoString": { "referenceable": "extension"</pre>
		where the dateLocalUtils module is only accessible to the current extension ("referenceable": "self"), but the dateToIsoString function within dateLocalUtils expands its access beyond what the module allows. This is not allowed, so the function can only be called by artifacts in the current extension.
		If a function does not define referenceable, it assumes the access set on the module. The default access for a module is self.

Param Metadata Object

Name	Туре	Description
label	string	A string naming this function group.
description	string	A string describing this function group.
type	string	Parameter type. Only primitive types, array, object, and any are supported. Default is any.
defaultValu e	string	(Optional) Default value for the parameter, used to indicate whether a parameter is required or optional. If a default value isn't specified, the parameter is considered required.

Work With JSON

When you're building an App UI, 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 App UI, flow, and page has its own JSON file to store metadata, as does each layout and fragment. By default, an App UI 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:

 Select the artifact, then click the JSON tab. For example, here's a view of the page-level JSON editor:

```
main-public-workers-detail ×
Page Designer
               Action Chains (2)
                                Event Listeners (2)
                                                            Types (1)
                                                                       Variables (3)
                                                                                                               JSON
  1
         "title": "publicWorkers Detail",
  2
  3
         "description": "Displays publicWorkers record",
  4
          "variables": {
           "empId": {
   "type": "number"
  6
  8
           "publicWorkers": {
              "type": "getPublicWorkersResponse"
  9
 10
 11
            "publicWorkersId": {
              "type": "string",
 12
              "required": true,
 13
              "input": "fromUrl"
 14
 15
 16
          "metadata": {},
 17
         "types": {
 18
 19
            "@decorators.getPublicWorkersResponse": {
              "endpoint": "get_publicWorkers",
"service": "site_myappuiExtension:publicWorkers",
 20
 21
              "path": "request"
 22
 23
 24
            "getPublicWorkersResponse": {
              "assignments": {
 25
 26
                "items": [
 27
                     "DepartmentName": "string",
 28
 29
                    "FullPartTime": "string",
                     "JobCode": "string",
 30
                    "JobName": "string",
"LocationCountry": "string",
 31
 32
                    "LocationTownOrCity": "string"
 33
                ]
 35
 36
 37
              "DisplayName": "string"
```

2. Update the metadata as required.

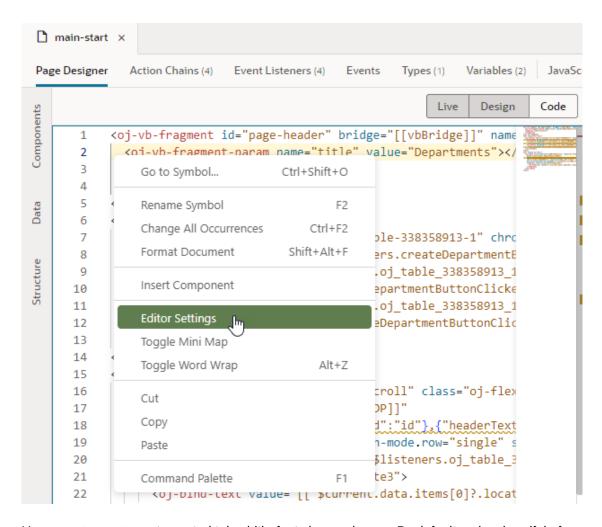
Trigger Code Insight

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:

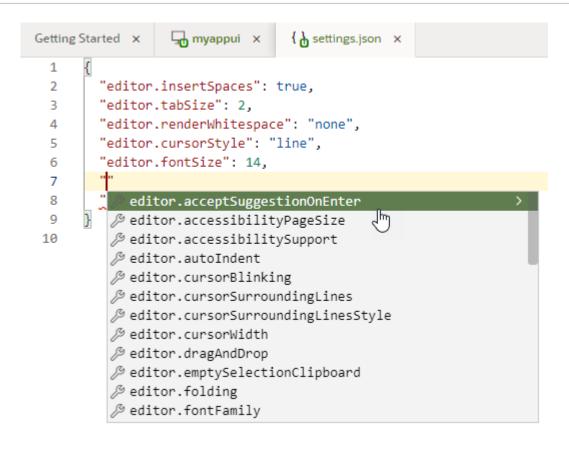
Editor	Use this to trigger insight:
JavaScript	Enter any character to trigger insight. URL selector and standard imports for a module are not supported.
JSON	Enter any character to trigger insight based on the file's associated JSON schema.

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 App UI, 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. Enter double quotation marks (") in the editor to see the available options:



Setting	Description	Default
editor.acceptSuggestion OnEnter	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.accessibilityPag eSize	Number of lines read out by a screen reader	None
editor.accessibilitySup	 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 	advanc ed
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 at the current position	line
editor.cursorWidth	When cursorStyle is set to line, controls the width of the line	2
editor.cursorSurroundin gLines	Minimum number of lines visible above and below the cursor, starting with 0	0
editor.cursorSurroundin gLinesStyle	Whether cursorSurroundingLines should be enforced: default: Enforce cursorSurroundingLines only when cursor is changed using the mouse all: Enforce cursorSurroundingLines always	defaul t
editor.dragAndDrop	Enable or disable drag and drop of a selection: true or false	false
editor.emptySelectionCl ipboard	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	monosp
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	normal
carcor . ronewerghe	numbers between 1 and 1000	



editor.formatOnType Whether a line should be automatically formatted while typing: true of false false typing: true of false false typing: true of false true editor.insertSpaces Insert spaces (instead of tabs) when the Tab key is used for indentation true true editor.lineHeight Control spacing between letters, in pixels None editor.matchBrackets Whether matching brackets should be highlighted when the cursor is at a brace: always, never, of near always editor.mouseWheelScroll Numbers of lines to scroll when the mouse wheel is used 1 editor.mouseWheelZcom Whether pressing the Control key and the mouse wheel should change font size: true of false 1 editor.mouseWheelZcom Whether pressing the Control key on Windows and the mouse wheel should change font size: true of false 1 editor.mouseWheelZcom Whether pressing the Control key and the mouse wheel is should change font size: true of false 1 editor.mouseWheelZcom Whether pressing the Control key and the mouse wheel is should the mouse wheel is should read the mouse wheel is outsors: a little with a mouse click to create multiple cursors: a little with a mouse click to create multiple cursors: a little with a mouse click to create multiple cursors: a little with a mouse click to create multiple cursors: a little with a mouse click to create multiple cursors: a little with a mouse wheel should be click to create multiple cursors	Setting	Description	Default
ditor.letterSpacing Control spacing between letters, in pixels None	editor.formatOnType		false
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, of near editor.mouseWheelScroll Sensitivity Sen	editor.insertSpaces		true
editor.matchBrackets Whether matching brackets should be highlighted when the cursor is at a brace: always, never, or near Numbers of lines to scroll when the mouse wheel is used editor.mouseWheelZoom Whether pressing the Control key and the mouse wheel should change font size: true or false editor.multiCursorModif ier Whether 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 editor.occurrencesHighl of the current word or variable: true or false editor.renderLineHighli Gentrol how the current line is highlight other occurrences of the current word or variable: true or false editor.renderWhitespace Control how the current line as well as the gutter line: Only highlight the current line gutter: Only highlight the current line egutter: Only highlight the current line should be reached to select dext true or select dext true or false editor.selectOnLineNumb ers clicked: true or false editor.showFoldingContr ols mouseover: Show the folding controls only when the mouse be sover the gutter always; Always should be faded out: true or false whether unused variables should be faded out: true or false editor.showUnused Whether insight suggestions None editor.suggestFontSize editor.suggestInringger Whether insight should be triggered by special	editor.letterSpacing	Control spacing between letters, in pixels	None
editor.mouseWheelScroll Sensitivity editor.mouseWheelZcom whether pressing the Control key and the mouse wheel is used whether pressing the Control key and the mouse wheel should change font size: true or false editor.multiCursorModif ier alt: Maps to the Alt key on Windows and to the Option key on Mac crrlCmd: Maps to the Control key on Windows and to the Option key on Mac crrlCmd: Maps to the Control key on Windows and to the Option key on Mac editor.occurrencesHighlight ght controls how the current line is highlight other occurrences of the current word or variable: true or false Controls how the current line is highlighted: alt: Highlight the current line as well as the gutter line: Only highlight the current line gutter: Only highlight the current line editor.renderWhitespace Control how the editor should render whitespace characters boundary: Render whitespace characters except for single spaces between words selected text trailing: Render only trailing whitespace characters only on selected text trailing: Render only trailing whitespace characters editor.showFoldingControls control when folding controls show: always: Always show the folding controls only when the mouse is over the gutter editor.showUnused Whether unused variables should be faded out: true or false Whether unused variables should be faded out: true or false editor.suggestFontSize editor.suggestLineHeigh tuee Whether insight suggestions None	editor.lineHeight	Control height of a line	None
editor.mouseWheelZoom Whether pressing the Control key and the mouse wheel should change font size: true or false editor.multiCursorModif cursors: alt: Maps to the Alt key on Windows and to the Option key on Mac editor.occurrencesHighl ight chief current word or variable: true or false editor.renderLineHighli ght Controls how the current line is highlight other occurrences in alt: Highlight the current line as well as the gutter line: Only highlight the current line is equiter none: Do not highlight the current line editor.renderWhitespace Control how the editor should render whitespace characters none: Do not render whitespace characters except for single spaces between words selected text trailing: Render only trailing whitespace characters editor.showFoldingContr ols Control when folding controls show: always: Always show the folding controls only when the mouse is over the gutter editor.showUnused Whether unused variables should be faded out: true or false Whether insight suggestions None editor.suggestFontSize editor.suggestOnTrigger Whether insight should be triggered by special	editor.matchBrackets		always
editor.multiCursorModifier 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 editor.occurrencesHighlight to track cursor and highlight other occurrences of the current word or variable: true or false editor.renderLineHighlight all: Highlight the current line as well as the gutter line: Only highlight the current line as well as the gutter line: Only highlight the current line as well as the gutter none: Do not highlight the current line: Squtter none: Do not highlight the current line as well as the gutter line: Only highlight the current line as well as the gutter none: Do not highlight the current line: Squtter none: Do not highlight the current line: Squtter none: Do not highlight the current line as well as the gutter line: Only highlight the current line: Squtter none: Do not highlight the current line: Squtter none: Do not render whitespace characters except for single spaces between words selected text trailing: Render whitespace characters except for single spaces between words selected text trailing: Render only trailing whitespace characters except for single spaces between words all: Render all whitespace characters only on selected text trailing: Render only trailing whitespace characters except for single spaces between words all: Render all whitespace characters whitespace characters except for single spaces between words all: Render all whitespace characters whitespace characters except for single spaces between words all: Render all whitespace characters whitespace characters except for single spaces between words all: Render all whitespace characters whitespace characters except for single spaces between words all: Render all whitespace characters except for single spaces between words all: Render all whitespace characters except for single spaces between words all: Render all whitespace characters except for single spaces between words all: Render			1
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 editor.occurrencesHighl ight Whether to track cursor and highlight other occurrences of the current word or variable: true or false editor.renderLineHighli ght Controls how the current line is highlighted: all: Highlight the current line as well as the gutter line: Only highlight the current line's gutter line: Only highlight the current line gutter: Only highlight the current line gutter: Only highlight the current line gutter: Only highlight the current line one: Do not highlight the current line editor.renderWhitespace Control how the editor should render whitespace characters houndary: Render whitespace characters except for single spaces between words selection: Render whitespace characters only on selected text trailing: Render only trailing whitespace characters editor.selectOnLineNumb ers editor.showFoldingContr ols All: Render all whitespace characters editor.showFoldingContr ols All: Render all whitespace characters editor.showFoldingContr ols All: Render all whitespace of the line number is clicked: true or false editor.showUnused Whether the line should be selected if the line number is clicked: true or false editor.showUnused Whether unused variables should be faded out: true or false editor.suggestFontSize Font size for insight suggestions None editor.suggestLineHeigh true Whether insight should be triggered by special	editor.mouseWheelZoom		false
ight of the current word or variable: true or false editor.renderLineHighlight Controls how the current line is highlighted:		 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 	alt
all: Highlight the current line as well as the gutter line: Only highlight the current line gutter: Only highlight the current line gutter: Only highlight the current line gutter: Only highlight the current line editor.renderWhitespace	_		true
characters:	_	 all: Highlight the current line as well as the gutter line: Only highlight the current line gutter: Only highlight the current line's gutter 	all
editor.showFoldingContr ols editor.showFoldingContr ols elways: Always show the folding controls only when the mouse is over the gutter editor.showUnused whether unused variables should be faded out: true or false editor.suggestFontSize Font size for insight suggestions editor.suggestLineHeigh the folding controls only when the mouse is over the gutter None editor.suggestLineHeigh to insight suggestions editor.suggestOnTrigger Whether insight should be triggered by special true	editor.renderWhitespace	 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 	
• always: Always show the folding controls • mouseover: Show the folding controls only when the mouse is over the gutter editor.showUnused			true
editor.suggestFontSize Font size for insight suggestions None editor.suggestLineHeigh Line height for insight suggestions None t editor.suggestOnTrigger Whether insight should be triggered by special true	-	always: Always show the folding controlsmouseover: Show the folding controls only when	
editor.suggestLineHeigh Line height for insight suggestions t editor.suggestOnTrigger Whether insight should be triggered by special true	editor.showUnused		None
t editor.suggestOnTrigger Whether insight should be triggered by special true	editor.suggestFontSize	Font size for insight suggestions	None
		Line height for insight suggestions	None
			true



Setting	Description	Default
editor.suggestSelection	Controls how suggestion history works: first: Always select the first suggestion	None
	 recentlyUsed: Select recent suggestions 	
	 recentlyUsedByPrefix: Select suggestions based on previous prefixes that have completed those suggestions 	
editor.tabCompletion	 Enable or disable completion by pressing the Tab key: on: Insert the best matching suggestion when pressing Tab off: Disable Tab completion 	None
	 onlySnippets: Tab complete snippets when their prefixes match 	
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	redwoo d
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 	fit
	be larger than the editor (no scrolling)	
editor.minimap.side	Where to render the minimap: right or left	right
$\begin{array}{l} \texttt{editor.minimap.renderCh} \\ \texttt{aracters} \end{array}$	Render characters on a line as opposed to color blocks: ${\tt true} \ {\tt or} \ {\tt false}$	true
editor.minimap.scale	Scale for rendering the minimap, starting with 1	1



Files in Source View

When you work with the tree view in the App UIs pane, you are essentially viewing a logical representation of an App UI, with visual editors that show content that resides in several source files. If you want to work directly with the source files underlying an artifact, you can access them in the Navigator's **Source** view.

Source view () provides the complete file structure of your extension's artifacts. Sometimes, two or three separate files might describe an artifact's behavior and properties. This file structure is also what's checked into your Git repo for version control.

Here's an example of folders and files you might see in Source view—opening a source file will display its contents in the corresponding editor:

```
extension1/
| sources/
    | dynamicLayouts/
        | oracle-cx-digitalsalesUi/
            | products/
                | data-description-overlay-x.js
                | data-description-overlay-x.json
                | layout-x.html
                | layout-x.js
                | layout-x.json
        | self/
            | orders/
                | data-description-overlay.js
                | data-description-overlay.json
                | layout.html
                | layout.js
                | layout.json
    | services
        | self/
       | | catalog.json
           | orders/
                | openapi3.json
    ∣ ui
        | base/
            | app-flow-x.js
            | app-flow-x.json
          self/
            | applications/
                | orders/
                    I flows/
                            | flows/
                            | pages/
                                | main-start-page.html
                                | main-start-page.js
                                | main-start-page.json
                            | main-flow.js
                            | main-flow.json
                   | resources/
               | fragments/
```

				order-details/
				chains/
				LoadCustomerDetails.json
				SwitchValueChangeChain.json
				order-details-fragment.html
				order-details-fragment.js
				order-details-fragment.json
			reso	urces/
				images/
				foo.png
	exte	nsio	n.js	on

Your file structure may be different depending on your configuration. In general, extensions to the base application can be found under the /base folder, extensions to other dependent extensions can be found under /<other_extensionId> (where other_extensionId is the ID of the dependent extension, for example, oracle-cx-digitalsalesUi), and artifacts that are completely new in this extension can be found under /self.

Under these folders, you might see files with a -x suffix, for example, app-x.json or layout-x.json. These are extension files, either from dependencies or created in your extension, that you can modify. The -x files allow you to extend base files. Suppose ExtA has a layout.json file that defines a dynamic layout; ExtB which uses ExtA as a dependency would have a layout-x.json file that extends the one in ExtA. As an ExtB developer, you can modify layout-x.json, but you won't be able to modify the layout.json file from ExtA.

Here's a summary of the different source files and what they are used for:

Folder/File Name		Description		
dynamicLayouts/		Contains dynamic layout files from dependent extensions as well as new ones created in the extension:		
•	layout.html layout.js layout.json	For each dynamic layout, a JSON file that contains dynamic layout metadata including rule sets, layout fields, and field/form templates used, an HTML file that contains the structure used by field and form templates, and a JavaScript file that contains functions associated with the layout.		
•	data- description- overlay.js data- description- overlay.json	For each dynamic layout, client metadata files that define additional fields or field metadata to augment the service's OpenAPI definition.		
services/		Contains files from dependent services as well as new ones created in the extension:		
•	openapi3.json catalog.json	 OpenAPI3-based specification file that contains metadata and request and response schema for static service connections. OpenAPI3-compliant file that contains backend definitions and metadata for dynamic service connections. 		
ui/self/ applications/an- appui/		Contains files added when you create a brand new App UI:		
•	app.js app.json	Files describing the App UI that can be used by every artifact in the App UI, for example, variables that are App UI-scoped, types that describe data structures, and security settings.		



	Description
flowname- flow.jsflowname- flow.json	For each flow, a JSON file that contains flow metadata and a JavaScript file that contains flow-level functions.
• flowname- pagename- page.html • flowname- pagename- page.js • flowname- pagename- pagename- pagename- page.json	For each page, an HTML file that specifies page elements, a JavaScript file that determines page functions, and a JSON file that contains page metadata.
 actionchainna me.json actionchainna me-tests.json 	For each action in an action chain, a JSON metadata file about the action. If tests are defined, a JSON metadata file for all tests in the action chain.
 fragmentname- fragment.html fragmentname- fragment.js fragmentname- fragment.json 	For each fragment, an HTML file that specifies fragment elements, a JavaScript file that determines fragment functions, and a JSON file that contains fragment metadata.
resources/ extension.json	Resource files, such as images (.png, .jpg, and so on) and style sheets (css), which are which are newly imported to your App UI and stored in the corresponding folder. If custom actions are defined, you'll find each action's implementation (action.js) and metadata (action.json) files as well. The resources folder can be at the extension level or at the App UI level. Contains extension metadata.



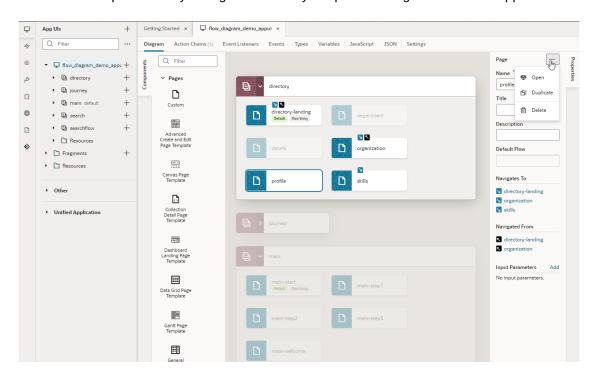
Work With the Diagram View

As your App UI takes shape in the Designer, you can use the Diagram view for a visual representation of your App UI's structure.

The Diagram view, shown only for App UI and flow artifacts, displays an App UI's flows and pages within those 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 App UI in context; for example, you can change the default page in a flow and see how your updates change your App UI's structure.

When you open an App UI artifact, the Diagram view displays 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 App UI 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 profile page shows navigation icons (\mathbb{N} on the directory-landing and organization page tiles, indicating that you can navigate from profile 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 \mathbb{N} icon, as shown on the skills 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.

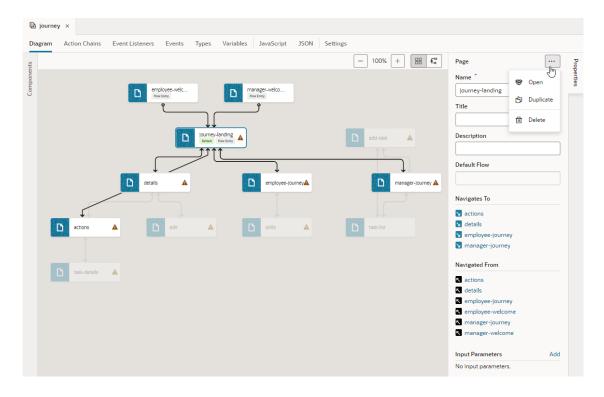
It's possible to make changes to your App UI from the Properties pane. You can change page titles and descriptions as well as update the app's default page in a flow. You can even open, duplicate, and delete selected items. Alternatively, you can double-click an artifact (a flow or a page) to open up the artifact's editor and make changes as required.

The Diagram view also flags pages with audit issues (\triangle or \bigcirc). 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 pefault 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:

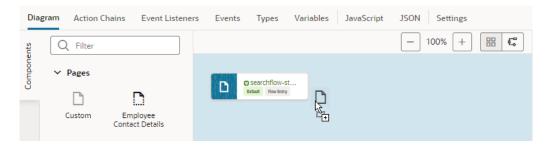
Opening a flow that was built using the page editors only shows navigation by default in the Flow Diagram. But if you were to build your flow from scratch in the Flow Diagram, all pages and associated action chains will also show. To change this setting, see Show or Hide an Action Chain in the Flow Diagram.

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 subflows, and can serve as a starting point for a new workflow.

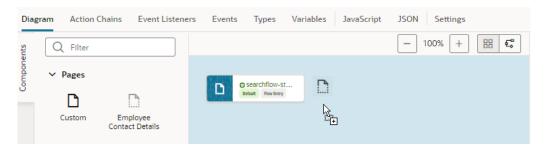
Add a Page in the Flow Diagram

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 ($^{\square}$) as well as the Grid view ($^{\square}$).

- 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.

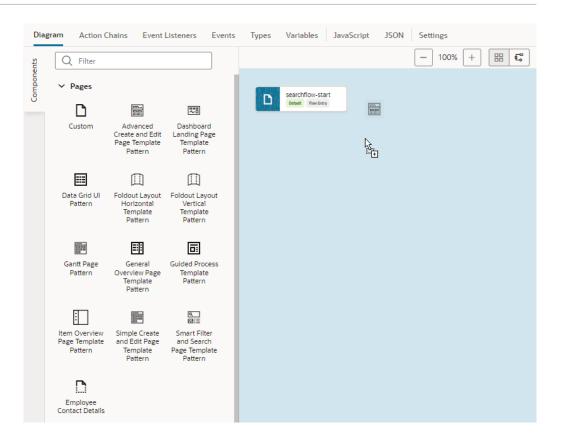


• To create a page containing a specific fragment, drag the fragment under Pages in the Components palette and drop it onto the diagram.



• 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 extension (and its dependencies) 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.

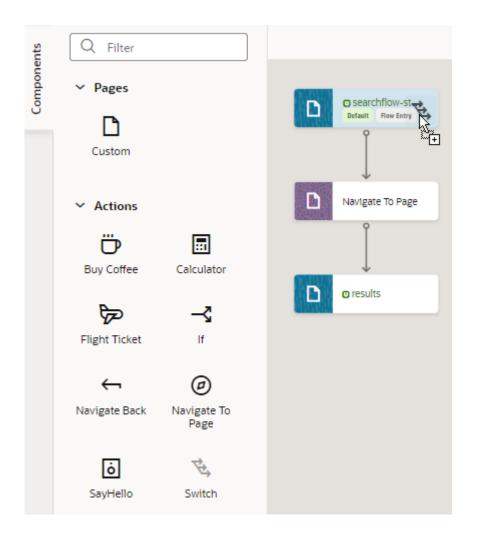


Note:

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:

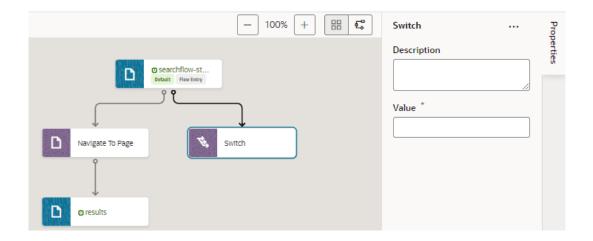
- Select an application's flow to open it the Flow Diagram view ().
- 2. Drag and drop an action (built-in or custom) under Actions in the Components palette and drop it onto a page in the diagram.



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.

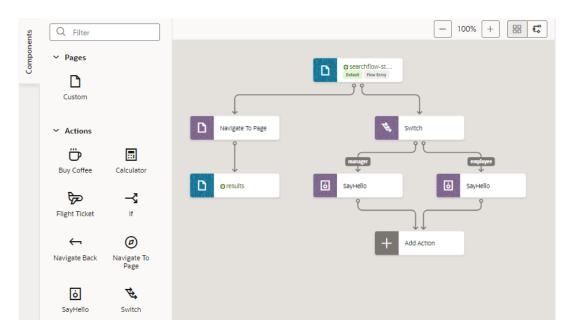
A new action chain is created with your action as the root. Use the action's Properties pane to suitably configure your action's properties. The properties that display are typical for built-in actions. For example, here's what you'll see for a Switch action:



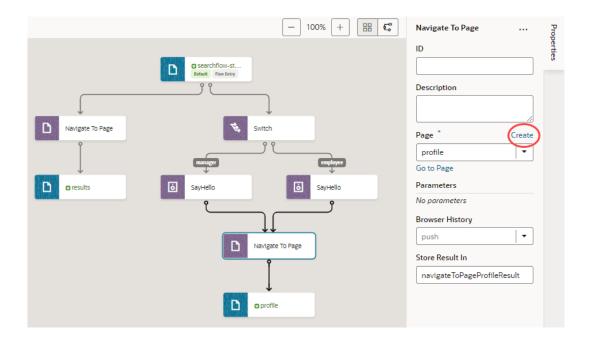


4. If your action involves additional steps, drag and drop additional actions as needed.

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:



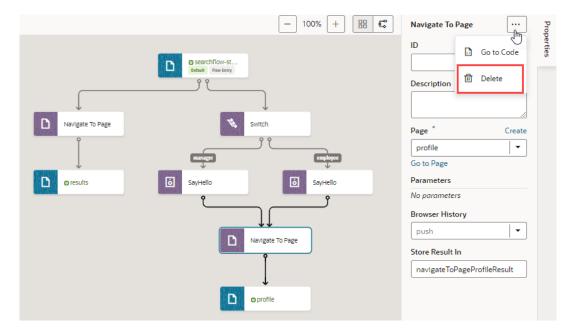
The cases you add for the Switch action show as labels, as do outcomes of decision nodes for an If action. The first action you drop onto an If action is considered the true outcome and the second is considered the false outcome. When the If action has both true and false outcomes or a Switch action 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.)

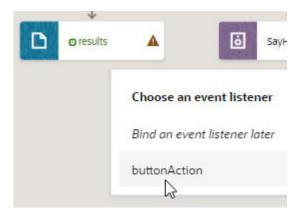
If you want to delete an action, click Menu in the action's properties and click **Delete**.



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.
 - 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.
- 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:





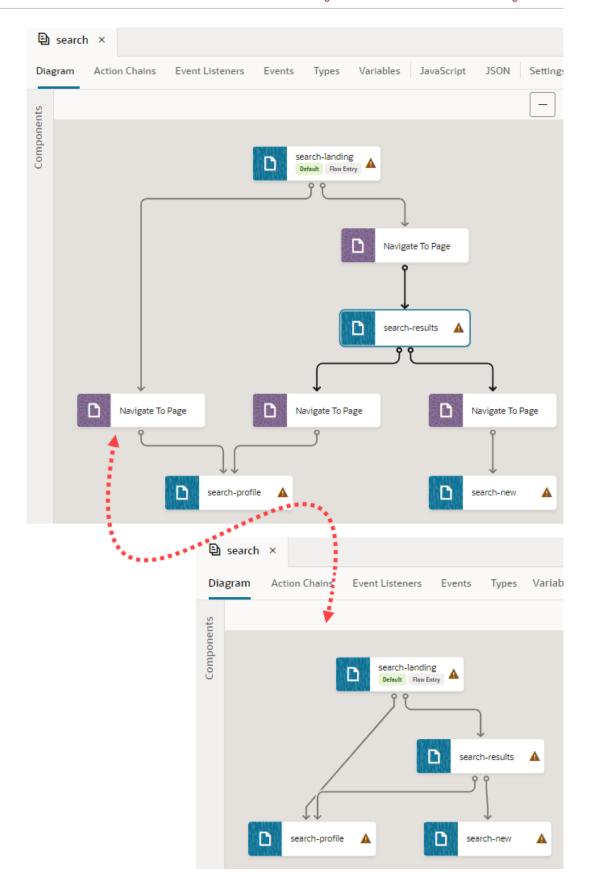
Show or Hide an Action Chain in the Flow Diagram

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 ($^{\cupe5}$).
- 2. Double-click the page containing the action chain you want to show or hide.
- 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:





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.



Work With Fragments

When your App UI involves complex pages—pages with a foldout layout, for example, or with multiple panels or tabs—you might find it easier to define those sections as fragments. Fragments are reusable pieces of UI that you can include in multiple pages across your App UI.

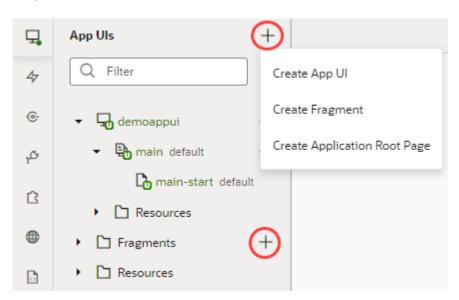
Earlier chapters describe why you might want to use fragments, so we won't get into the details here. If you're not familiar with these concepts, review What Are Fragments? as well as Develop an App UI or Fragment, then learn more in this chapter.

Create and Add a Fragment to a Page

Create one or more fragments to define sections of a page, then add them to your App UI's pages. 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 App UI, 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.

You can create fragments using the **+ Fragment** option in the App UIs pane or the **Create Fragment** option under the Menu (+), as shown here and described in Create an App UI or

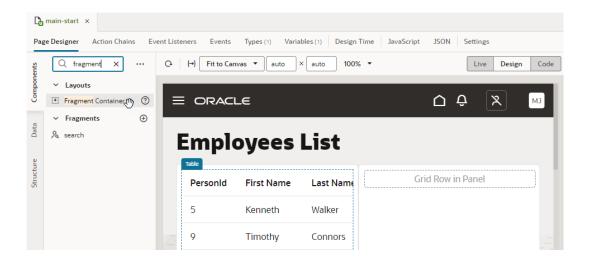
Fragment:



Alternatively, create a fragment when designing a page by clicking \bigoplus 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:

Open your App UI, then go to 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.

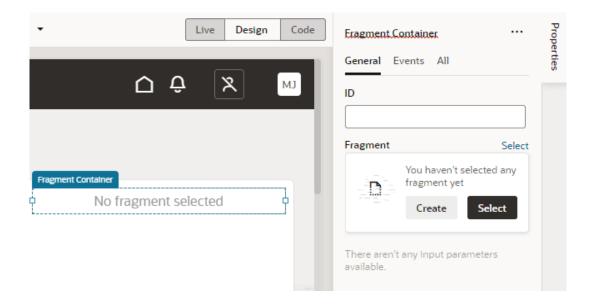
2. Drag a Fragment Container from the Components palette onto the canvas and drop it where you want a fragment to display.



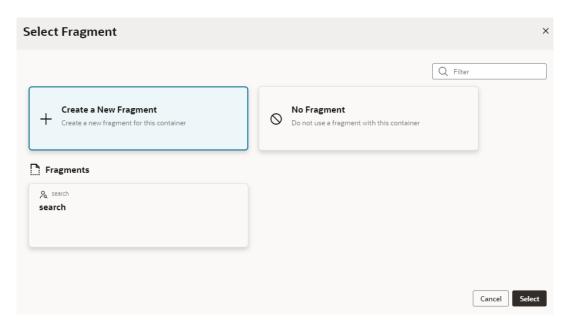
Tip:

Notice how a search fragment shows up in addition to the Fragment Container when components are filtered? That's because all available fragments—including extendable ones defined in a dependent extension—by default become available for use in your App UI'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.

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.

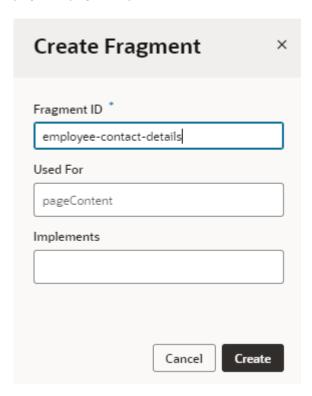


If you click **Select** and find that the available fragments don't meet your needs, you can create a new fragment even from the Select Fragment dialog, as shown here:



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.



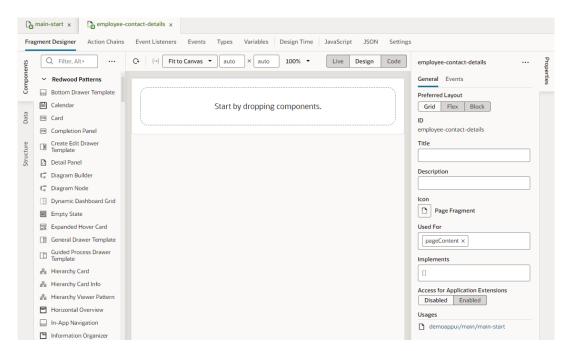
By default, every fragment is tagged as pageContent, surfacing it under Fragments in the Components palette as generic content available for use in any page. But using

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 are 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 <code>employee-contact-details</code> fragment is being used by the <code>main-start</code> page in the <code>demoappui's</code> main flow. Clicking the <code>demoappui/main/main-start</code> link will open the page in the Page Designer tab.

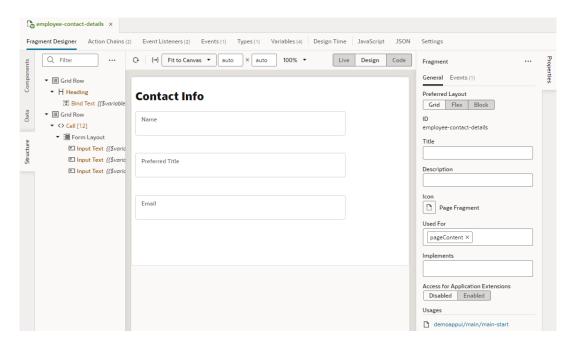
5. Now design your fragment in the Fragment Designer.

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 the 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.

Here's an example of a fragment set up to show an employee's details in a form:



(For steps on how to design this sample fragment and wire up the necessary parameters, see Sample Scenario: Create a Fragment and Pass Values.)

- 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.

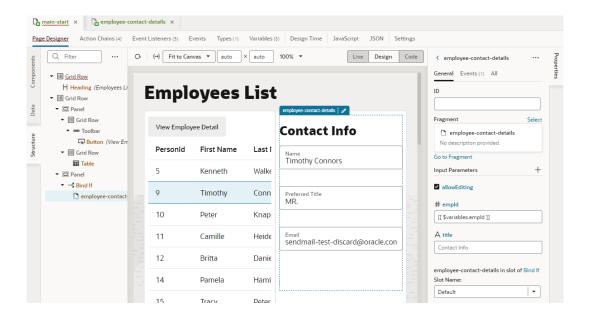
Alternatively, repeat steps 2 to 5 to create and add fragments starting with a fragment container on a page.



Tip:

It's also possible to create pages starting with the contents of an existing fragment, essentially using the fragment as a page template.

Once a fragment is added to a page, its content will automatically render on the page that consumes it, as shown here:



You can then use the fragment container'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.
- Click Select.

Define Actions and Events in Fragments

When developing fragments, you use *actions* to define what should happen in your user interface and *events* to trigger those actions, much the same as in pages or flows. However, the scope of these actions and events is confined to the fragment itself, ensuring that the fragment is independent but unaware of its parent container's context.

This means a fragment cannot call action chains defined in the container (say, a page, layout, or even another fragment) that uses this fragment. It also cannot listen to events fired by the parent container, but it can fire custom events that "emit" to the parent container. See Pass Data Between a Fragment and Its Parent Container.

How you define actions events in a fragment is similar to those in other scopes, but not all events from other scopes are available in fragments. Here's a list of event types and events supported within a fragment:

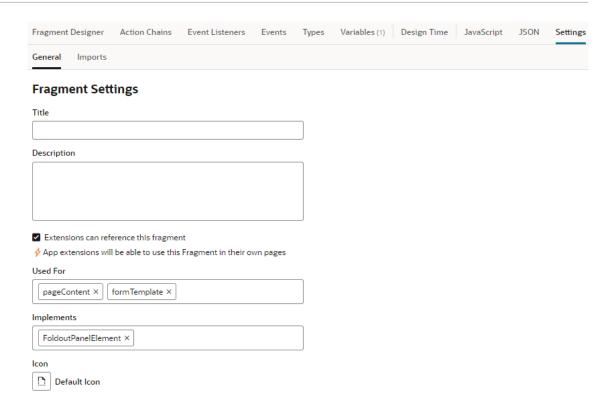
Type of Event	Description	How to Define	
Component events	An event associated with a UI component in a page, including those in dynamic components. It's possible to choose which event the component triggers, but available events will depend on the component. For example, an ojAction event is available to a button but not to an input text.	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.	Start an Action Chain When a Variable Changes	
Lifecycle events	Predefined events that are automatically triggered during a fragment's lifecycle: • vbEnter is triggered when the fragment state is activated. • vbExit is triggered when the fragment state is disposed (generally by navigating away from a page or the page is disposed).	Start an Action Chain From a Lifecycle Event	
Custom events	A user-defined event to start an action chain. Two types of custom events are supported in fragments: • An event that can be handled by listeners defined in the fragment. This event is similar to other custom events in VB Studio and is used to communicate consolidated state changes made in the fragment to the outer container. • An event that "emits" to the container. This event is used to propagate values to a fragment's parent container.	 Start an Action Chain By Firing a Custom Event Raise Fragment or Layout Events that Emit to the Parent Container 	

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 its Settings editor:





Here's how you can use the different settings (which are also available on a fragment's Properties pane):

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.
Extensions can reference this fragment	Option to allow your fragment to be referenced by App UIs in other extensions. When this option is selected, users in other extensions can reference the fragment in new pages after they add the extension containing the fragment as a dependency.



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 Fragments).		
	If you tag a fragment as formTemplate or fieldTemplate, you can indicate how fragment metadata, specifically databinding 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		

Reuse a Fragment

Imports tab

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.

open the Icon Gallery, then make your selection.

Contains settings to manage resources such as custom CSS files, modules, and components imported at the flow level, allowing you to create declarative references that can be shared among pages in the flow. See Create Declarative References to Imported Resources.

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 extension level, they can be used in any page, in any flow within the App UI. To reuse the fragment in another extension, you'll need to first publish your extension, then add it as a dependency to the other extension.

- To use a fragment in multiple pages:
 - 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.
 - 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 VB Studio 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 and its extensions 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.

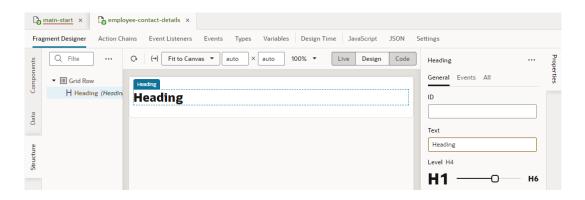
Enable Fragment Variables as Input Parameters

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.

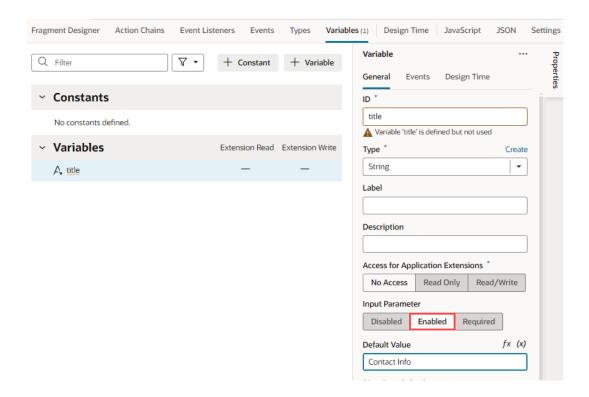
To enable a fragment variable as an input parameter:

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:

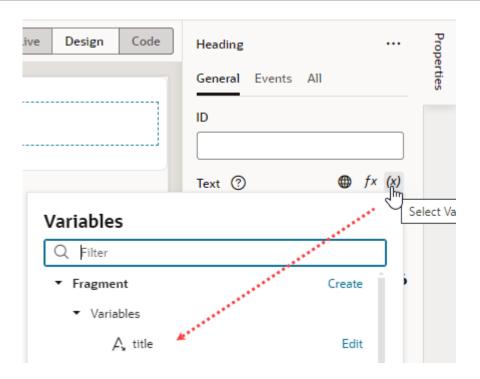




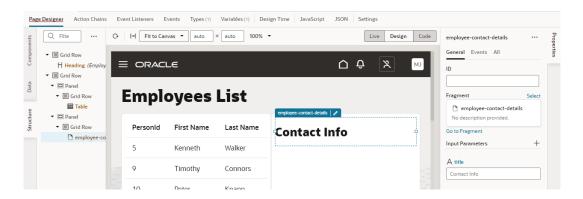
2. On the fragment's Variables tab, create a string-type variable for the title (for example, title) and enable it as an input parameter. Optionally, set a default value (say, Contact Info).



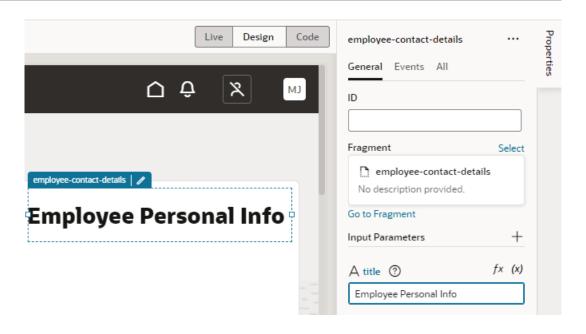
3. Switch to the Fragment Designer and bind the Heading component to the variable you just created.

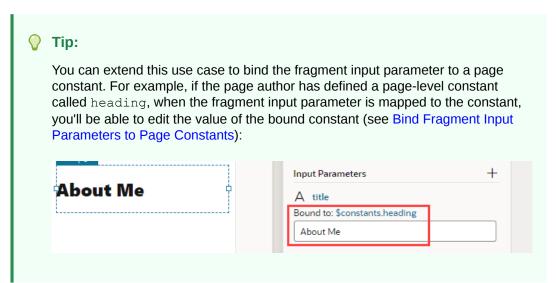


4. Now open the page where the employee-contact-details fragment is used (drag and drop the fragment onto the canvas, if needed). Fragment variables marked as input parameters become available to you on the page.



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).



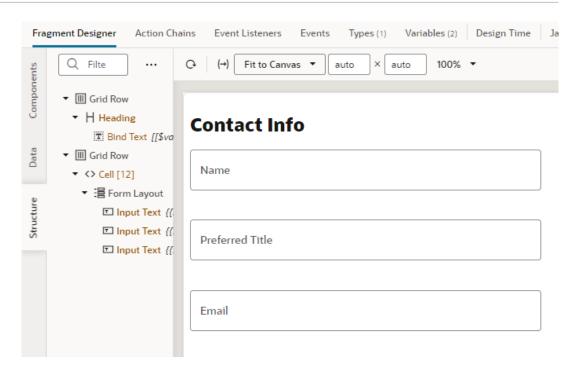


Enable Page Variables to Provide Initial Values for a Fragment's Input Parameters

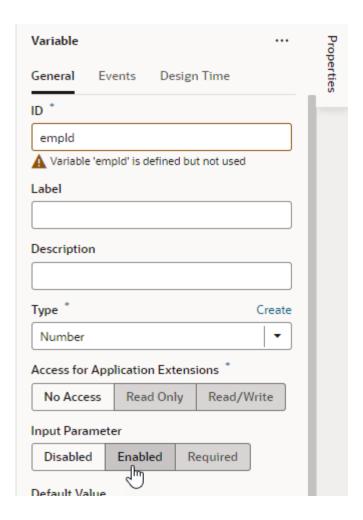
You can pass a page variable as an input parameter from the page to a fragment that it references in order to provide initial values for the fragment's input parameters.

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:



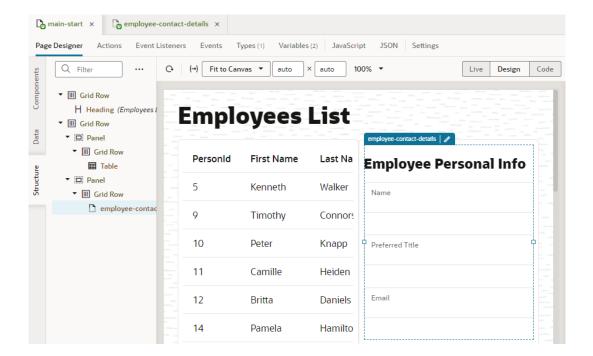
2. On the fragment's Variables tab, create a number-type variable for the employee's ID (for example, empId) and enable it as an input parameter.



Note:

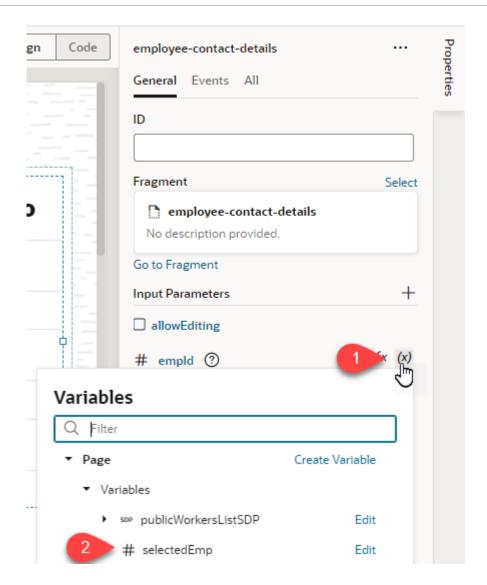
You can mark the variable to be automatically created on the container that uses the fragment. This way, when the fragment is dropped onto an existing page or container, the variable is created on the page and wired back to the fragment variable's value. If you don't select this option, you'll need to 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).

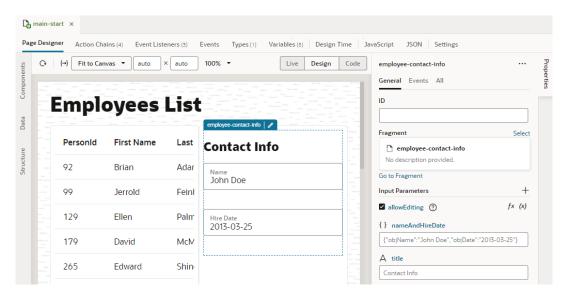
If you want to refresh the fragment's content on the page when the variable's value changes (in other words, when another employee is selected), an <code>onValueChanged</code> event can be triggered on the fragment variable that calls an action chain to update the contact details of the newly selected employee.

Bind Fragment Input Parameters to Page Constants

When a fragment is used in a page, you can bind its input parameters to page constants. This way, the value of the constant becomes the value of the input parameter. So if the constant's value is changed, that new value is passed to the fragment input parameter it is bound to. Further, if these constants are exposed to extension authors, it gives them the ability to effectively see and control the fragment input parameter values, something they wouldn't normally have access to.

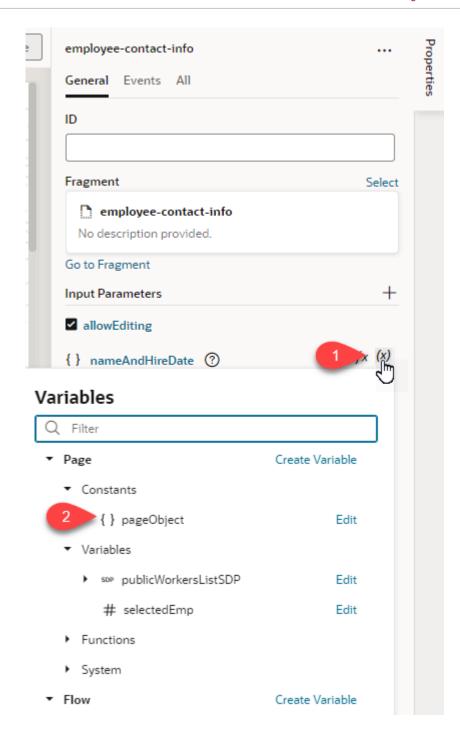
After a fragment (with variables enabled as input parameters) is added to a page, here's how to bind the fragment input parameter to a page constant:

1. Select the fragment used on the page, then look for the Input Parameters available to the page on the fragment's Properties pane:

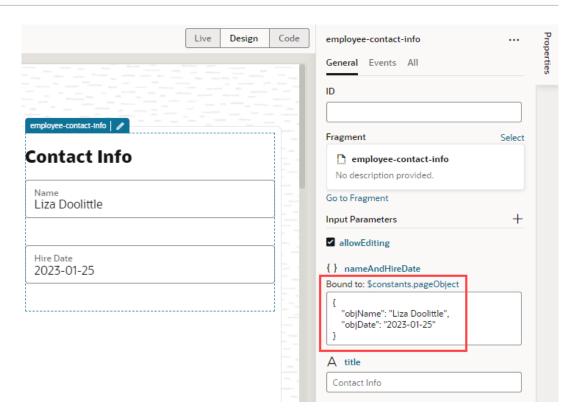


Take note of the object-type nameAndHireDate fragment input parameter whose default values display on the page in the Name and Hire Date fields. Let's assume that the page author has defined a pageObject constant to provide different values for this fragment input parameter.

 Click (x) on the nameAndHireDate input parameter, then select pageObject under Page and Constants:



3. Use the constant's default value if one is defined (for example, Liza Doolittle and 2023-01-25), or enter new values:



The constant that the input parameter is bound to appears next to **Bound to**. You can click the expression (\$constants.pageObject) to view the constant's definition on the page-level Variables editor.

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 <code>empAvatar</code> 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 <code>avatar</code> 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 <code>avatar</code> variable on the fragment is updated to use a new URL, the change is written back to the outer page variable (<code>empAvatar</code>). Depending on your setup, this might also update the table of employees where the new picture is shown in the currently selected row.

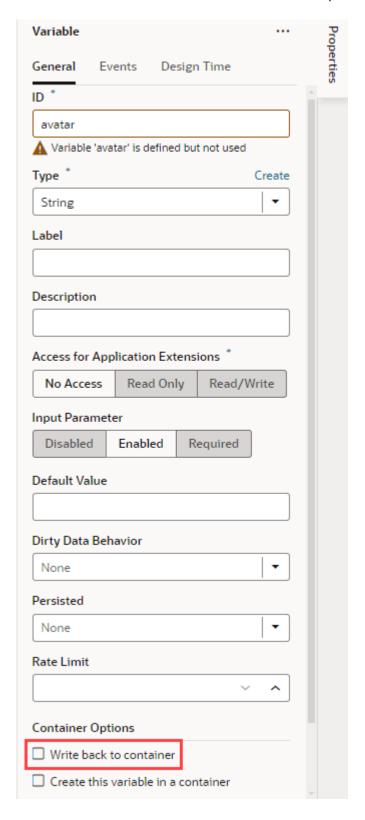
(To see an example of writeback in action, see the Passing Values To and From Fragments blog post.)

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:

 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 in 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, VB Studio creates the variable (or constant) on the page and wires it back to the fragment variable's (or constant's) value.

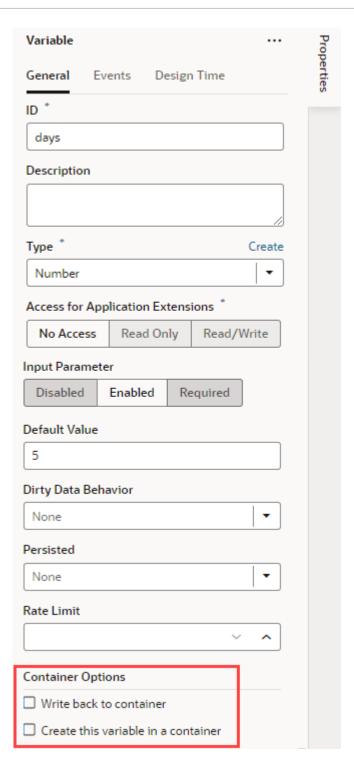
This option is especially useful when input parameters must be passed from a page for the fragment to work. By automatically wiring the required input parameters, you won't have to create and configure those variables on the page when the fragment is added to it, though you'd still need to assign values. 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.



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 this variable in container or Create this constant in container:



When this option is selected, the <code>@dt.createOptions</code> metadata is added to the fragment's JSON definition; for example:

```
"variables": {
  "days": {
    "type": "number",
    "input": "fromCaller",
    "defaultValue": "5",
    "@dt": {
        "createOptions": {}
    }
},
```

4. Optional: By default, constants are created as constants and variables as variables in the container (Auto under Create As). In most cases, you can keep this setting, but in some scenarios—say, when fragments accept page-level properties as input parameters whose values are expressions that are evaluated within the scope of the page—you may want to switch one for the other: a constant for a variable, or a variable for a constant.

For example, assume your fragment defines title as a variable, whose value is passed from a page based on an expression like this:

```
"title": {
    "defaultValue": "[[ 'Edit Employee: ' + $variables.data.firstName + '
' + $variables.data.lastName ]]",
    },
```

In such a scenario, the firstName and lastName values come from data fetched internally by the fragment when the page loads and are made available to the page as part of the \$variables.data variable, so that the fragment can actually resolve the expression on the page. Now if the page-level title isn't likely to change at all, you can choose to autocreate title as a constant (instead of a variable) on the page. This way, title stays a variable in the fragment (allowing \$variables.data changes on the page to reflect in the fragment) while it can be autocreated on the page as a constant.

Further, it's useful to switch a variable to a constant when you want extension authors to be able to customize fragment input parameter values on a page.

5. Optional: If you want to make the variable or constant in the container available to App UIs in another extension:

- Under Access for Application Extensions in the variable's Properties pane, select Read Only to allow other App UIs to read the variable's value; select Read/Write to allow other App UIs to read and modify the variable's value.
- Under Access for Application Extensions in the constant's Properties pane, select Read/Override to allow other App UIs to read or override the constant.
- 6. Optional: If you want to make the variable or constant in the container an input parameter of the container:
 - Under Input Parameter, select Enabled to make the container variable or constant an
 optional input parameter, or select Required to make the container variable or
 constant a required input parameter.
 - Select Pass on URL if you want to pass this input parameter to the container as part
 of the URL.
- 7. Optional: Set Binding to Page to On if you want to add binding metadata to the variable or constant in the container to retrieve the input parameter's data definition from the fragment and display it on the page. With this setting enabled, the variable or constant displays customized input values on the page just as it does in the fragment. So if you defined an enumerated list for a constant on the fragment, those same values show on the page as part of the fragment's properties.

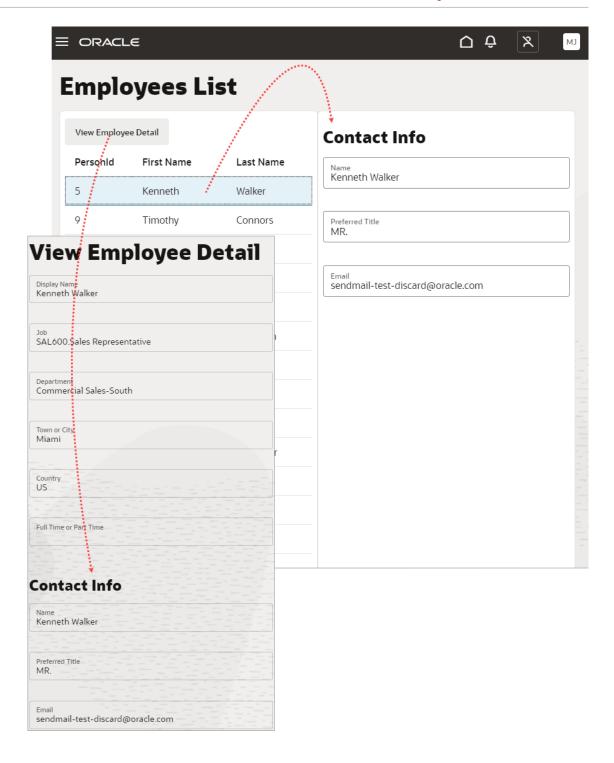
After you add the fragment to a page or a container, you'll see your variable or constant created on the page or container's Variables editor with the settings you specified. If you were to look at your page's code, you'll see the parameters wired in HTML, for example:

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.

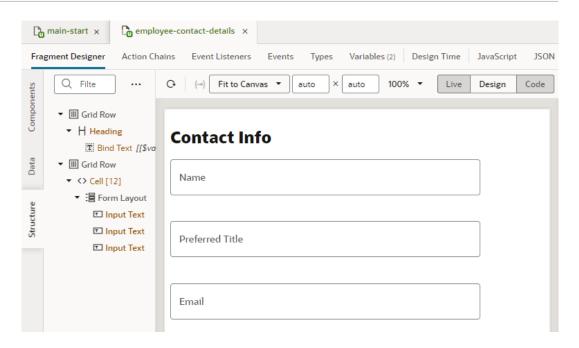
Say you have a page with employee data in a table. Clicking a row in the page's table brings up the employee's contact information. You have another page that lets users view additional information about an employee, including their contact details. To save time and effort, you can define the contact information part of your UI in a fragment and pull it into both pages. Additionally, each page can provide an input parameter that instructs the fragment to present a UI that allows edits if needed.





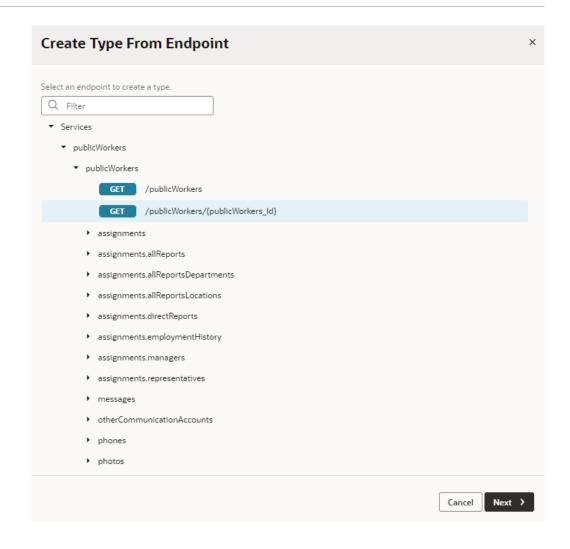
In both cases, the page containing the fragment passes the selected employee's ID as an input parameter to the fragment; the fragment receives the ID, retrieves the contact information, and renders it. If a user updates the selected employee (in other words, when the input variable's value changes), 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 and Input Text components:



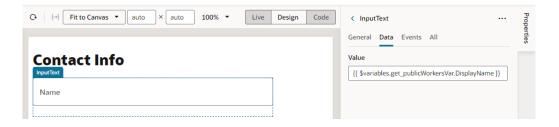
To provides values to these components, we'll create a type and a variable that holds this information, retrieved from the GET publicWorkers/{publicWorkers_Id} endpoint (our data source).

- 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 Services, select the Get/publicWorkers/{publicWorkers_Id} under publicWorkers, and click Next.



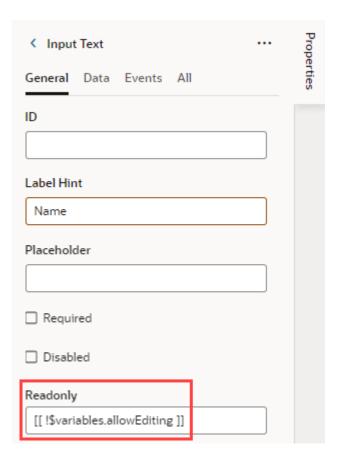
- c. Select the fields you want to display in the fragment, for example, DisplayName, Title, and WorkEmail. Click Finish.
- d. Right-click the newly created get_publicWorkers type and select Create Variable. A new get_publicWorkersVar (with get_publicWorkers as the type) is created on the fragment's Variables tab.
- e. Switch to the Fragment Designer and bind each component to the <code>get_publicWorkers</code> variable's corresponding value. For example, to bind the Name component to the employee's name, select the input text component, then in the component's Data tab,

hover over the **Value** field and click $^{(x)}$ to open the Variables picker. Select <code>DisplayName</code> under the <code>get_publicWorkersVar</code> fragment variable.



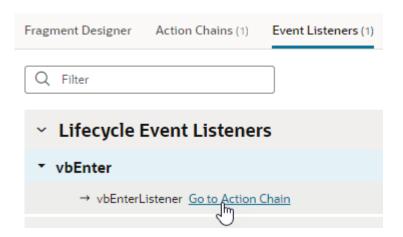


- Now that we have what we want to display, let's set up how we want employee information to show in the pages that use this fragment. Some pages might simply display the information as read-only data while others might need a way to edit it.
 - a. On the fragment's Variables tab, define a Boolean-type 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.

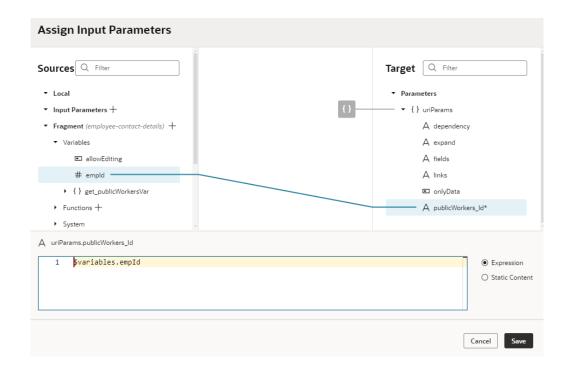


- 3. Because we want the selected employee's contact details to be displayed when the fragment is loaded on a page, we'll add a "vbEnter" lifecycle event that triggers an action chain to retrieve the correct employee information. This way, when the fragment is loaded, it takes the employee ID selected 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.

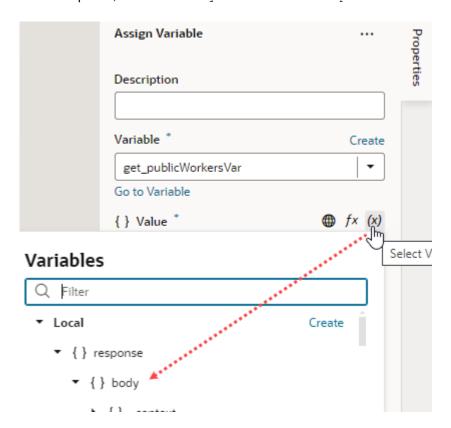


- d. When the vbEnterListener action chain opens in the editor, drag and drop a Call REST action onto the canvas. In the action's Properties pane, click Select next to Endpoint, expand Services, and drill down to the Get/publicWorkers/ {publicWorkers Id} endpoint. Click Select.
- e. Under Input Parameters in the action's Properties pane, click publicWorkers_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 publicWorkers_Id on the Target pane. Click Save.

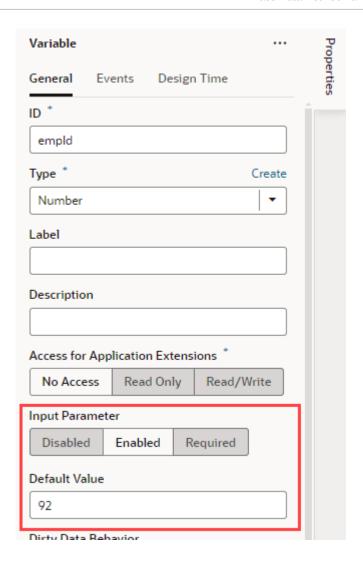




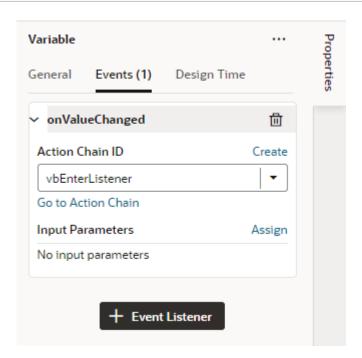
f. Now double click the Assign Variable action in the palette to add it to the end of the action chain. In the action's Properties pane, select get_publicWorkersVar under Fragment in the Variable drop-down list. Hover over the Value property and open the Variables picker, then select body under Local and response.



g. Switch to the **Variables** tab and look for the empld 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 92 as the **Default Value** for the input parameter.

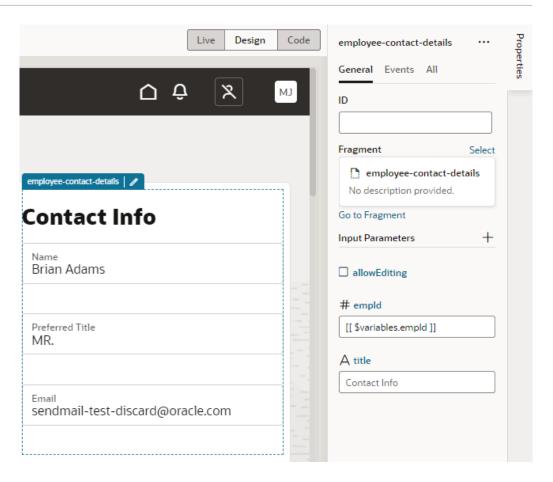


- h. With the variable enabled as an input parameter, select **Create this variable in a container**. This option automatically creates this variable on the page that uses this fragment and wires its value back to the value of the fragment input parameter.
- i. Click the variable's **Events** tab, then 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 <code>vbEnterListener</code> 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.

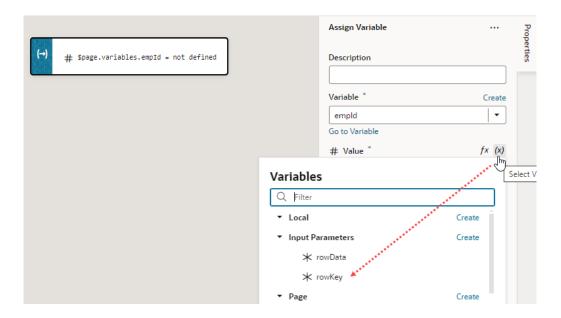


Take note of the fragment's Properties pane on the page. Because we marked the empId fragment input parameter to be autowired on the fragment container, the empId variable is automatically created on the page (look in the Variables tab) and wired back to the fragment input parameter's value (92 by default).

- **c.** To make the employee's contact information editable on this page, select the allowEditing input parameter.
- d. 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 **If** component in the Structure view; then in the Properties pane's Test condition, use the Variables picker to select the empId variable.

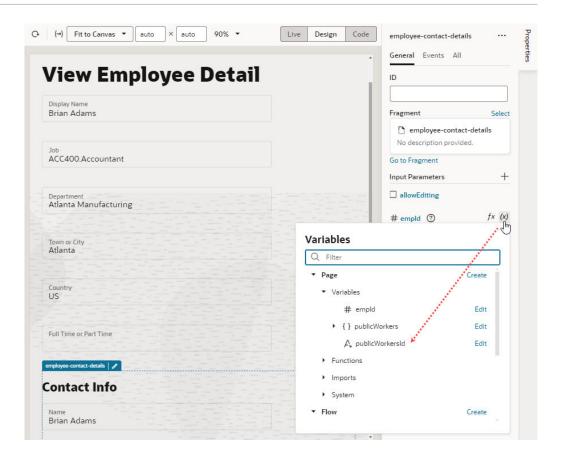


- Set up the employee table to retrieve information based on the employee ID in the selected row.
 - a. Click the table component on the page, select the Events tab, then click + Event Listener and select On 'First Selected Row' to retrieve information about the selected row.
 - b. When the TableFirstSelectedRowChangeChain 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.

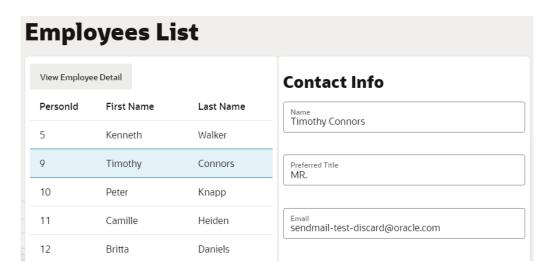


- **d.** Return to the page designer and click **Live**, then select a row in the table to see employee contact information reflected in the fragment based on the ID.
- e. Return to **Design** view.
- 6. Add the same contacts fragments to another page.
 - a. In the Page Designer, select the table and add a detail page using the Add Detail Page guick start.
 - b. Click Live, select an employee in the table, and click View Employee Detail 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 empld input parameter in the Properties pane, select the publicWorkersId variable.





- 7. As a final step, switch to the main-start page, then 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 editable values.



b. Click **View Employee Detail** for the selected row to view the employee's information, including contact details, on the View Employee Detail page.

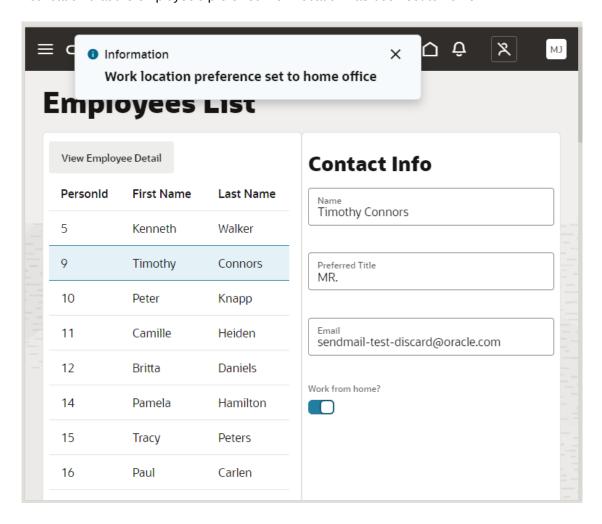
	Name hy Connors
Job IST10	O.Istore Administrator
Departr Vision	nent Corporation Enterprise
Town or New Y	
Country	
	tact Info
Name	hy Connors
Preferre MR.	d Title
_{Email} sendn	nail-test-discard@oracle.com



Create Custom Events that Emit to a Fragment's Parent Container

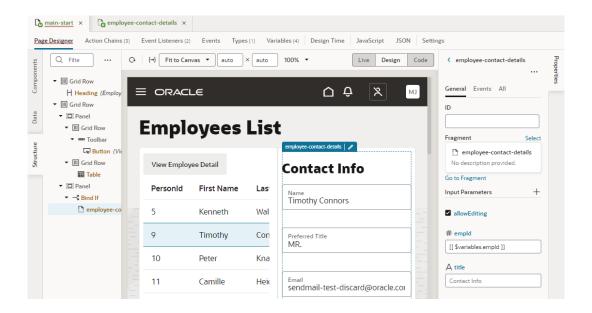
One way to pass data from a fragment to its parent container (say, a page or another container like a different outer fragment) is by raising custom events that "emit" to the container.

Let's extend the employee contacts use case to see how to do this. Here, the employee contact information fragment lets you specify whether home is the employee's preferred work location. A user who toggles the **Work from home?** switch in the fragment will see a notification that the employee's preferred work location has been set to home:

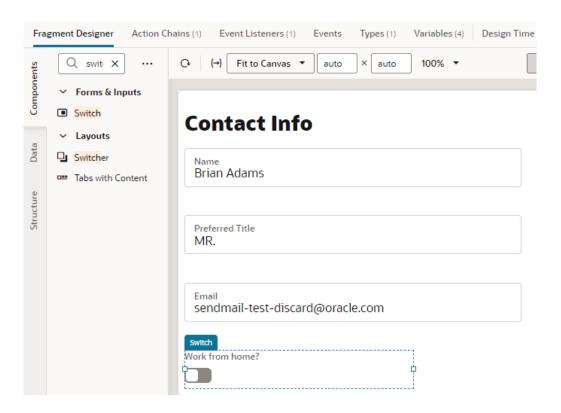


Behind the scenes, when the user toggles the **Work from home?** switch, an action chain defined on the fragment fires an event that "emits" the event's payload to the fragment container. An event listener on the fragment container, watching for this fragment event to fire, triggers a page-level action chain to perform some action—which, in our example, is firing a notification that the selected employee's work location preference has been set to home.

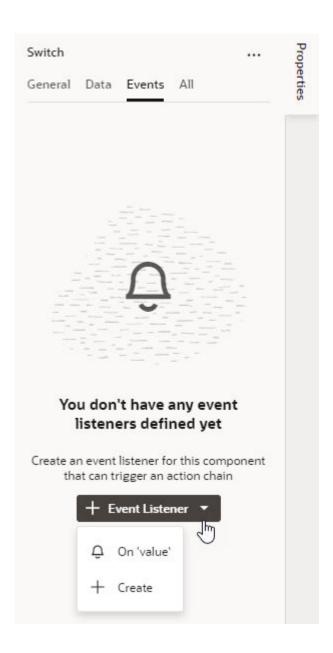
1. Configure your fragment to raise a custom event that the parent container listens to. Here's an example of a page that pulls in the employee-contact-details fragment:



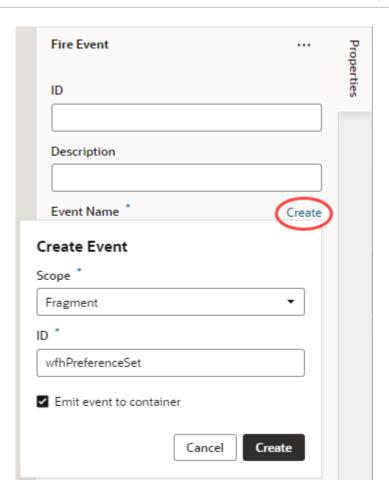
- a. From the fragment's Properties pane on the page, click **Go to Fragment** to access the Fragment Designer. You can also click the icon that appears next to the fragment name on the canvas.
- b. Drag and drop a Switch component in your details fragment.
- c. In the Properties pane, change the Label Hint text in the General tab to Work from home?.



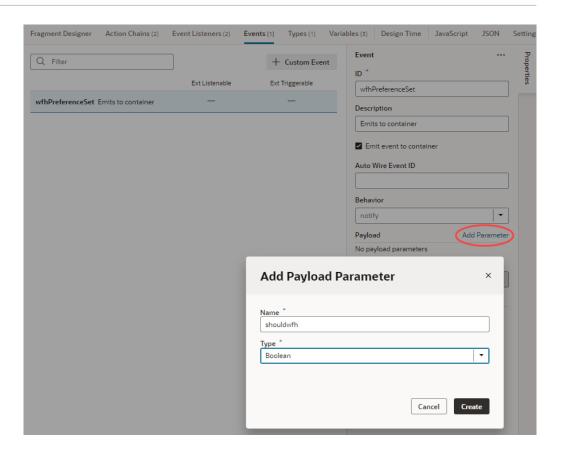
d. Switch to the component's Events tab, then click + Event Listener and select On 'value'.



- e. In the SwitchValueChangeChain action chain, drag and drop a Fire Event action.
- f. Click **Create** next to Event Name in the Properties pane.
- g. With the Scope set to Fragment, enter an ID (for example, wfhPreferenceSet), and select **Emit event to container**. Make sure the event name starts with a lowercase letter, though camel case is allowed. Hyphens are not supported. Click **Create**.



- **h.** When the new event is created, click **Go to Custom Event** under the new event to go to the Events editor and define the event's payload.
- i. In the wfhPreferenceSet event's Properties pane, click Add Parameter next to Payload. In the Add Payload Parameter dialog, enter an ID (say, shouldwfh), select the type as Boolean, and click **Create**.

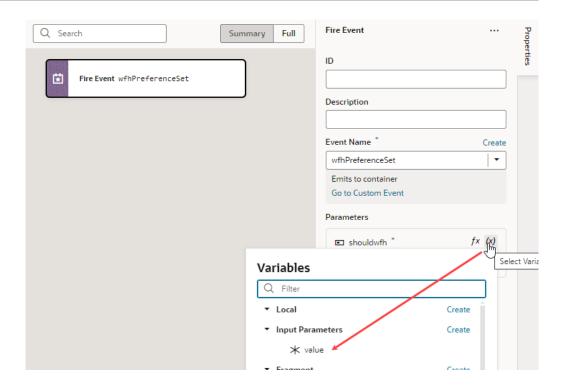


0

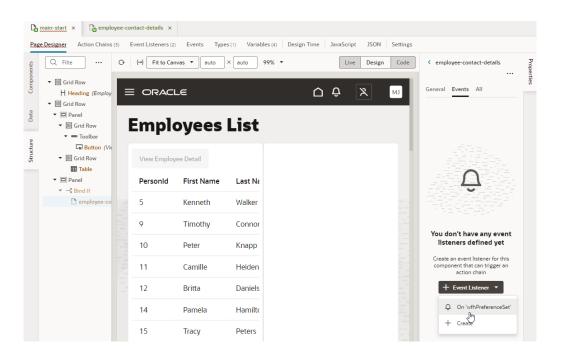
Tip:

To simplify the task of creating an event listener for this event on the fragment's parent container, you can choose to enter an **Auto Wire Event ID** (for example, wfhPreferenceSetAutoWire). This option enables the wfhPreferenceSet event to be listed under Fragment Events when you create a listener for the event from the Event Listeners tab on pages that use the fragment. See Automatically Wire a Fragment's Custom Event to the Parent Container.

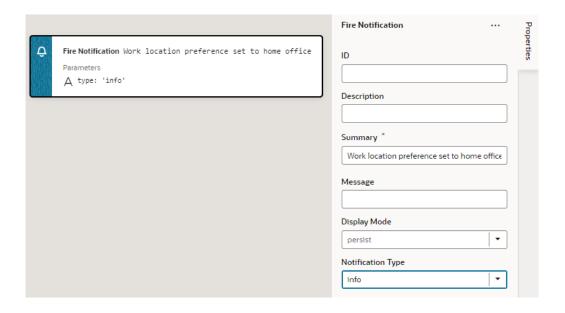
- j. Under Triggered by, click **SwitchValueChangeChain** to return to the action chains editor, and switch to **Design** view.
- k. In the Fire Event action's Properties pane, hover over the shouldwfh property under Parameters, click (x) to open the Variables picker, and select value under Input Parameters.



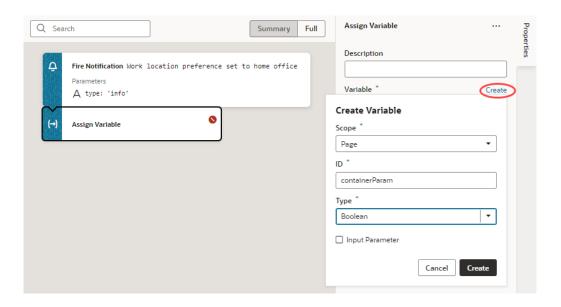
- Configure your page's fragment container to receive and process the fragment's custom event.
 - In the Page Designer, select the fragment, then in its Properties pane's Events tab, click + New Event and select the On 'wfhPreferenceSet' custom event.



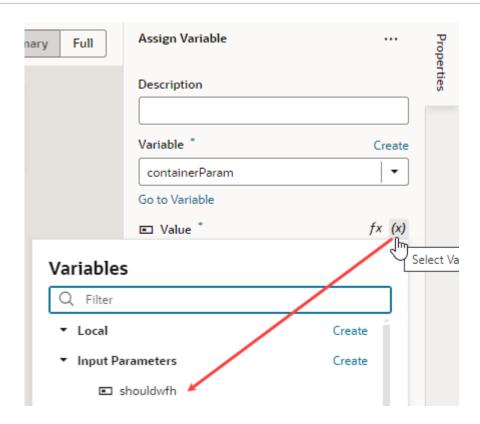
- b. Drag and drop a Fire Notification action to the FragmentWfhPreferenceSetChain action chain.
- c. Enter a summary in the Properties pane, something like Work location preference set to home office, then select Info as the Notification type.



- 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 **shouldwfh** under Input Parameters.



 Now return to the Page Designer, click Live in the toolbar, select an employee, and toggle Work from home?. You'll see a message that the employee's work location preference is set.

Automatically Wire a Fragment's Custom Event to the Parent Container

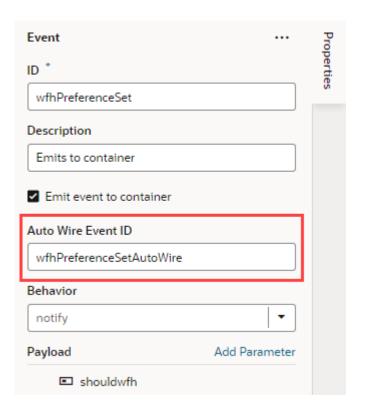
When working with a fragment custom event set up to emit payload to the parent container, you can automatically wire the event to the fragment's parent container (say, a page or dynamic container), and its extension. Autowiring an event makes it easier to create an event listener on the parent container, without having to define an event handler in HTML.

Here's a scenario where autowiring the fragment event to its parent container is particularly useful: Suppose you have a page that contains a dynamic container, which in turn contains a fragment. When the fragment uses a custom event to pass data to the parent dynamic container, you would need to define a component event listener on the dynamic container's page (immediate parent container) to listen for the fragment event and trigger an action chain. Now suppose this extension was published and a user chose to extend this dynamic container by adding a new section, there'd be no way to invoke the event listener defined on the dynamic container's base page. Instead, the user would need to create another component event listener on the extended dynamic container page for the same logic—leading to duplicate code.

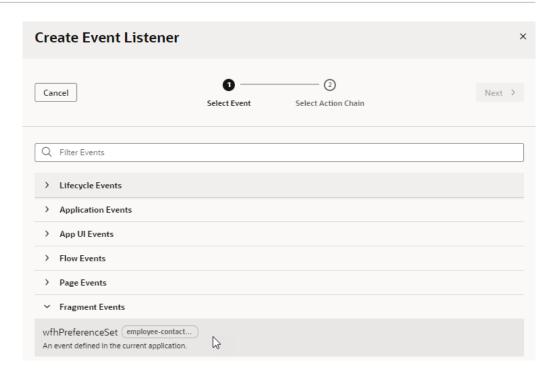
Automatically wiring a fragment event allows the event listener on the parent container to be invoked whenever the event is fired on the fragment. Additionally, if the event listener on the fragment's base parent container is enabled for extensions, it can be invoked even when the autowired event is fired on the extended parent container (in our example, the extended dynamic container page). If the fragment's base parent container and extended parent container both have autowired event listeners defined, both event listeners are invoked.

Here's how to automatically wire a fragment event to create a event listener on the fragment's parent container:

- Create a custom event that emits its payload to the parent container, then enable it for autowiring on the parent container.
 - a. In the fragment's **Events** tab, create a custom event that emits its payload to the parent container (**Emit event to container** must be selected and parameters added to specify the payload that will be passed to the parent container).
 - b. Enter an **Auto Wire Event ID**, which will identify the event's listener on the parent container once created.



- Create a listener for the autowired event on the fragment's immediate parent container. If
 your fragment is used on a page (for example), create an event listener on the page's
 Event Listeners tab; if it is used in a layout's form template, create the event listener on the
 layout's Event Listeners tab.
 - a. Go to the parent container's **Event Listeners** tab, click **+ Event Listener**, and select the autowired event under Fragment Events. Click **Next**.



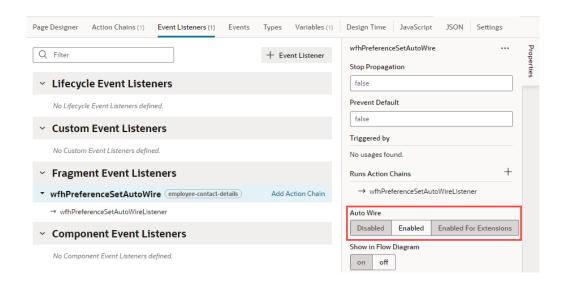
- Create (or select) the action chain that must be triggered when the event occurs. Click Finish.
- c. If you chose to create an action chain, look for the newly created event listener under Fragment Events on the Event Listeners page—it will use the ID you specified for the autowired event with the suffix Listener added to it (for example, wfhPreferenceSetAutoWireListener). Hover next to it and click Go to Action Chain to open the action chain and build the sequence that will be triggered when the event occurs.



When a fragment event is autowired to its parent container, the event listener's autoWire property is set to selfOnly in the fragment model, indicating that the event can only be handled by event listeners defined in the fragment's immediate parent container.

- 3. If you want the autowired event listener to be invoked when the autowired event is fired from a fragment that is part of an extended parent container, you must additionally enable the autowired event listener for extensions.
 - a. On the parent container's Event Listeners page, select the autowired event listener under Fragment Event Listeners.





b. Select Enabled for Extensions.



With this setting, the event listener's autoWire property is set to full in the fragment model, indicating that the event can be handled by event listeners defined in the fragment's immediate parent container as well as in the extended parent container.

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 VB Studio 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:

- 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	valueOption s	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.n ame]]]". 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.
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.

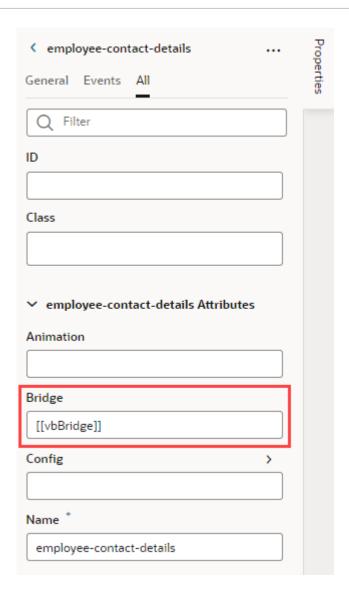
Pass a Fragment's Context to VDOM or Custom Web Components

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:

- 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.
- In the Bridge property, enter [[vbBridge]].

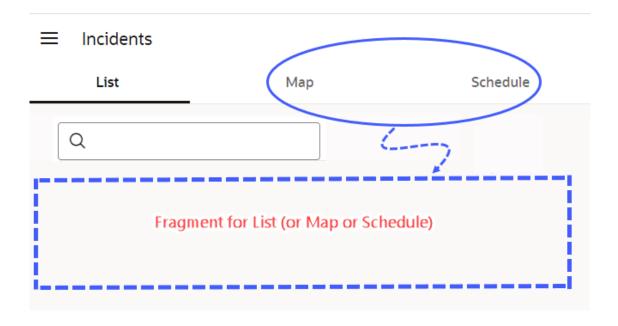




Defer Rendering of a Fragment's Content

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.

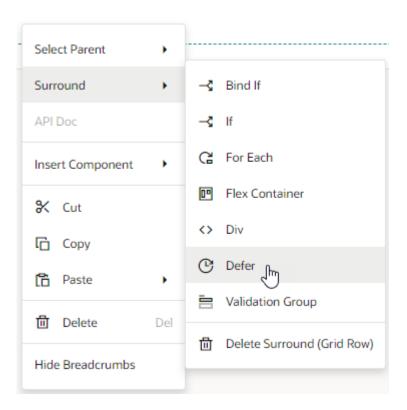
What triggers hidden fragments to render is configurable. It could be a button click, selecting a tab, opening a dialog, or an oj-if that uses conditions to display content. In these cases, UI events or application state determines when the fragment is loaded. In the Incidents example, the hidden Map 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:

- Open the page that contains fragments.
- 2. Select the fragment whose content you want to render later, either on the canvas or in the Structure view.
- 3. Right-click and select Surround, then Defer.



The Defer element is added to the fragment, both on the canvas and in Structure view. If you click Code view, you'll see <code>oj-defer</code> surrounding <code>oj-vb-fragment</code>. 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 }}">
id="list">List
 id="map">Map
  id="Schedule">Schedule/li>
</oj-tab-bar>
<oj-switcher value="[[ $variables.incidents ]]">
  <div slot="list">
      <oj-vb-fragment id="incidentslist" name="incidentsList"></oj-vb-</pre>
fragment>
  </div>
  <div slot="map">
    <oj-defer>
      <oj-vb-fragment id="incidentsmap" name="incidentsMap"></oj-vb-</pre>
fragment>
    </oj-defer>
  </div>
  <div slot="schedule">
    <oj-defer>
      <oj-vb-fragment id="incidentsschedule" name="incidentsSchedule"></oj-</pre>
vb-fragment>
    </oj-defer>
  </div>
</oj-switcher>
```

Make a Fragment Available to Other Extensions

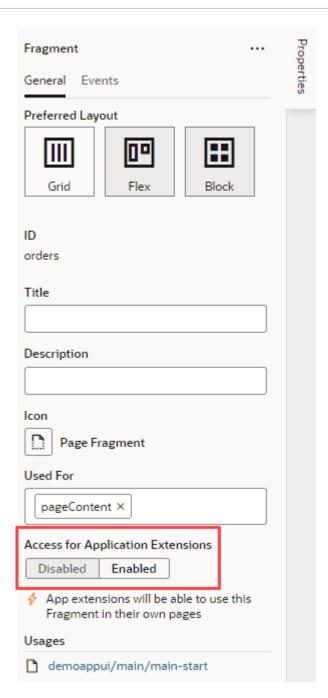
By default, fragments can be shared across App UIs in the same extension. If you want users in another extension to reference your fragment, you need to explicitly mark the fragment as available to extensions.

Making a fragment accessible to extensions makes it *referenceable*. You might want to do this to let others reuse your fragment in new pages, but if you want others to customize your fragment to change some behavior, the fragment needs to be *extendable*. A fragment becomes extendable when at least one of its artifacts—a variable, a dynamic component, or something else—is marked available to extensions.

To make a fragment referenceable in an extension:

 Open the fragment that you want to make available, then in its Properties pane, select Enabled under Access for Application Extensions.







Tip:

Alternatively, you can set this property in the fragment's Settings editor.

2. As a best practice, enter a description so developers who want to use this fragment know its purpose.

Fragments that are accessible to extensions have the referenceable property set to extension in their JSON metadata.

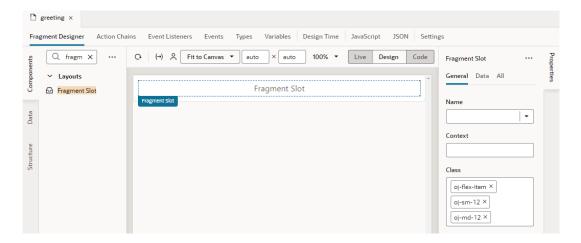
Add Slots to a Fragment

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.

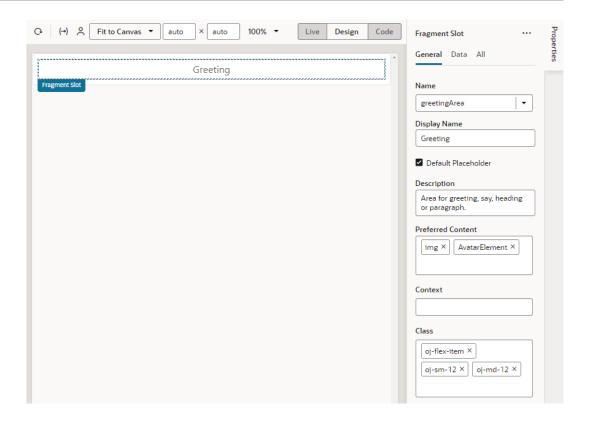
To add slots to a fragment:

- 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.

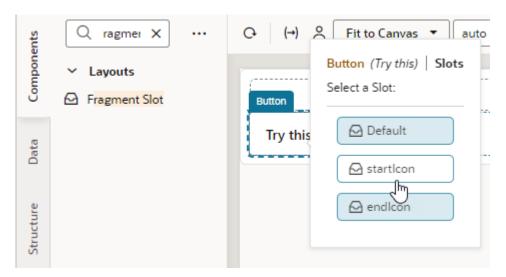


- In the Fragment Slot's Properties pane, enter a slot name in the Name property (for example, greetingArea).
- 4. Define other properties as needed:
 - a. To use a more descriptive identifier instead of the slot name, enter a **Display Name** that will appear in the slot's placeholder area as well as 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.
 - To allow fragment users to see contextual data in the custom slot component, set up the Context.

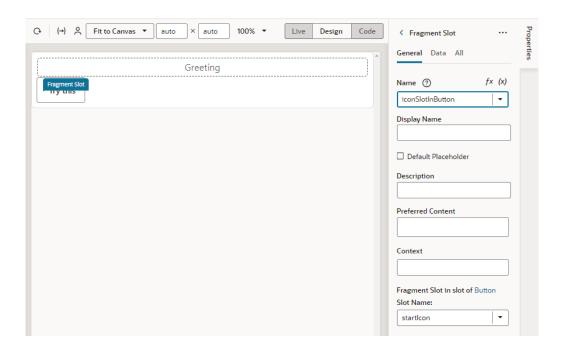




- 5. It's also possible to add a fragment slot to a slot inside another component (for example, a slot inside a button), allowing fragment users to customize those slots in the fragment. To do this:
 - a. Drag and drop a Button onto the fragment canvas, then set it up as desired.
 - b. Drag a Fragment Slot and drop it onto the Button, then select a slot in the button (for example, startIcon):



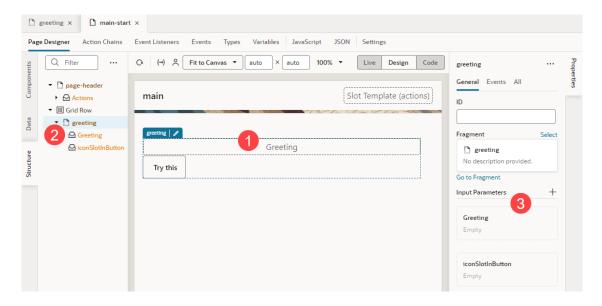
c. Enter a name for the fragment slot in the button's slot (for example, iconSlotInButton); optionally, define other properties:



Note:

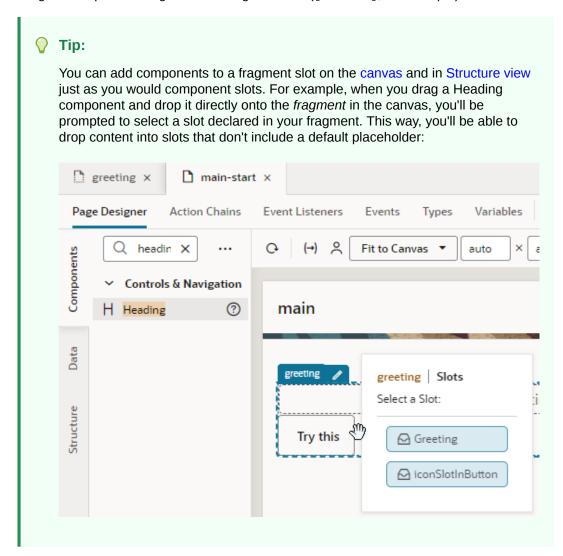
As a best practice, fragment authors should provide default content for any slots they define, just in case the user does not provide their own. See Add Default Content to a Fragment Slot.

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):

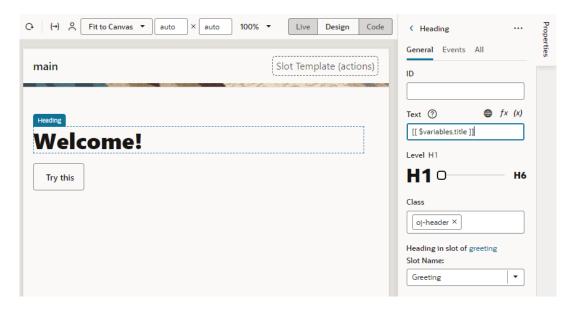


As a fragment consumer, you can now add the component of your choice to the slots revealed in the fragment. For demo purposes, let continue our greeting example and add a heading to the fragment slot on the canvas.

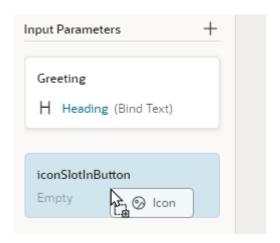
1. Drag and drop a Heading onto the Fragment Slot (greeting, for example).



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:

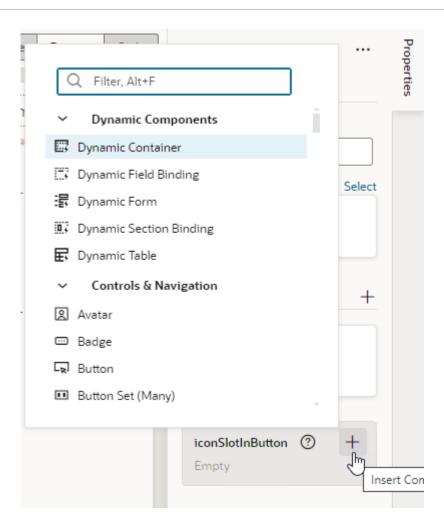


3. To customize the fragment slot in the slot inside the button, select the fragment to view the fragment's Properties pane, then simply drag and drop a component of choice onto the fragment slot. For example, drag an Icon from the Components palette onto the iconSlotInButton.



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:



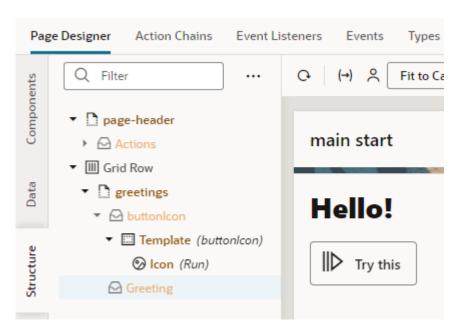
Add Default Content to a Fragment Slot

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 \neq template in your fragment slot's HTML.

For example, you can default to a standard greeting of <code>Hello!</code> in the <code>greetingArea</code> 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:

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 <code>oj-table</code>, <code>oj-list-view</code>, or <code>oj-bind-for-each</code>, 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 <code>oj-list-view</code> supports an <code>"itemTemplate"</code> slot, which provides contextual item data via a standard variable called <code>\$current</code>.



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:

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:

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:



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:

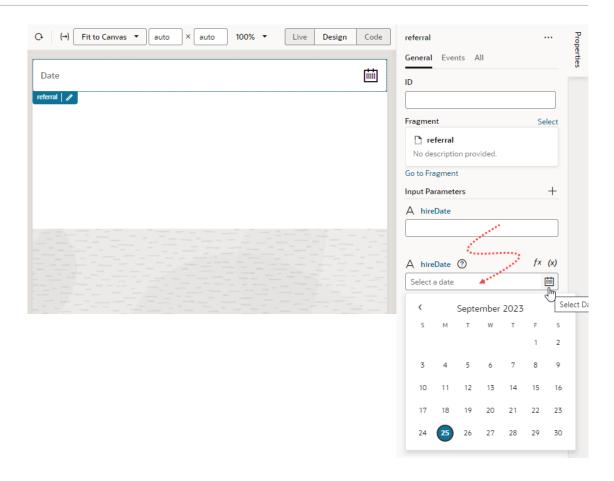
Option	See
Display an enhanced UI for each fragment input parameter in the fragment's Properties pane when a fragment is selected on a page.	Customize How a Fragment Variable is Displayed in the Properties Pane
Display a fragment's input parameters and other properties in sections on the fragment's Properties 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.

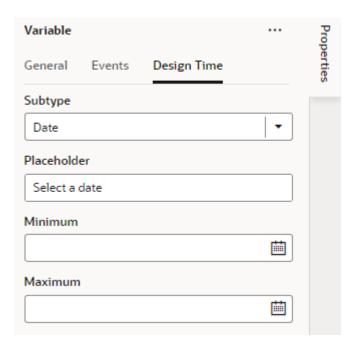
Customize a Variable in the Variables Editor

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:



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.

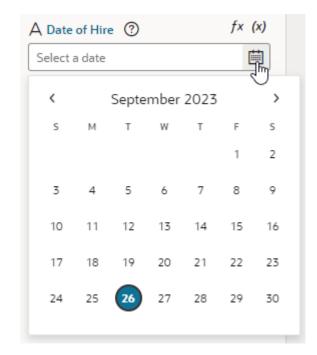
Customiz Steps in Design Time Tab Result in Page Designer ation Option To display For a string-type variable or constant: a color A textColor In the **Design Time** tab, select the **Subtype** as picker: Color. \odot Choose a color Optional: In the Placeholder field, specify a hint text for the variable; for example, Choose a color. 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.



Result in Page Designer

To display a date or date-andtime picker: For a string-type variable or constant:

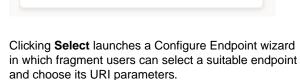
- In the Design Time tab, select the Subtype as Date or Date Time.
- b. Optional: In the Placeholder field, specify a hint text for the variable; for example, Select a date.
- c. Optional: In the **Minimum** field, set the bottom (inclusive) limit of a date or date-and-time range for the value in the 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.
- 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:

- 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.



You haven't selected an

Endpoint yet

Select

Select

Tip:

Endpoints

If you cannot find the endpoint you want or prefer to manually set up your endpoint, click the Manual Setup of Endpoint icon

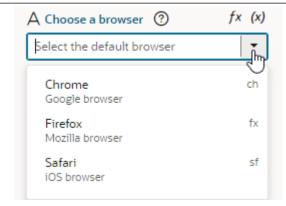
() in the wizard, then select from the available endpoints and configure its URI parameters.



The **Placeholder** field does not take effect in the Properties pane when you use the Endpoint subtype.



Result in Page Designer



To display a dropdown menu containing an array of possible values: For a string-type variable or constant:

- 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.
- 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.



Result in Page Designer

To display a dropdown list of values retrieved from a service, such as a REST endpoint: For a string-type variable or constant:

- a. In the Design Time tab, select the Subtype as LOV.
- b. Optional: In the Placeholder field, specify a hint text for the variable; for example, Select an option from the list.
- c. Click Select Endpoint and choose the endpoint whose response is used to provide the list of values. The REST endpoint must already be set up and available to your App UI. The endpoint response must also be in JSON format.
- d. When the REST call requires input and URI parameters, click the associated Assign or Not Mapped link and map the parameters to the value they should be replaced with. The value can be a VB Studio constant.

If the path and URI parameters depend on other constants (for example, your REST service uses the result of an earlier selection as part of its own request), use expression notation in the parameter values to indicate which constant values to use. You'll need to do this in the source JSON. For example, you would use the following expression to instruct a service request to use the current value of the "dept" constant as the value for the path parameter "department":

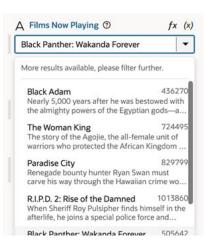
```
"pathParameters": {
    "department":
"[[ $constants.dept ]]"
}
```

When writing the expression:

- Use simple direct references. Calculated expressions such as "[[\$constants.dept + " "]]" do not work as expected.
- Use constants; variables cannot be used.
- Make sure the referenced constants are accessible to the extension making the LOV service call.
- e. If your endpoint response is not an array, in the Items Path, specify a dot-separated path from the root of the response object to the array containing the list of values. If your response is an array, you can leave this field empty.

For example, you would enter results when the response is not an array, as shown here:

```
"now": {
    "type": "string",
```





Result in Page Designer

```
"description": "wow",
     "defaultValue": "505642",
     "input": "none",
     "@dt": {
       "label": "Films Now Playing",
       "subtype": "lov",
       "valueOptions": {
         "service": {
           "request": {
              "endpoint":
"site_myappuiExtension:movies/
getNowPlaying",
             "uriParameters": {
                  "api key":
"4174b7d9a7b4bf87342c98e2289c6ee6"
      }
           },
           "response": {
             "itemsPath": "results",
              "mapping": {
               "value": "id",
                "label": "title",
                "description":
"overview"
         }
     }
   }
}
```

- f. In the **Value** field, specify the field from the response object to use as the actual value of the variable or constant, for example, id.
- g. Optional: In the Label field, enter the field from the response object to use as the primary display name of the item, for example, title. It can also be an array of field names, for example, ["firstName", "lastName"].
- h. Optional: In the **Description** field, enter the field from the response object to use as the item description, for example, overview.

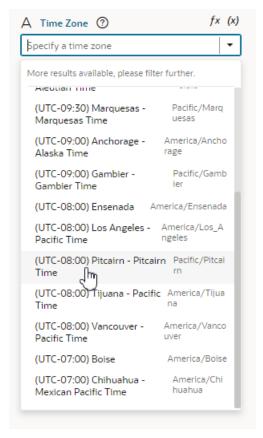


Customiz Steps in Design Time Tab Result in Page Designer ation Option To display For a string-type variable or constant: a time Enter a time between 09:00 AM and 05:00 PM. A Time (?) In the **Design Time** tab, select the **Subtype** as picker: Select a time Time. Optional: In the **Placeholder** field, specify a hint text for the variable; for example, Select a time. Optional: In the Minimum property, set the bottom (inclusive) limit of a time range for the value in the Properties pane. Optional: In the Maximum property, set the top (inclusive) limit of a time range for the value in the Properties pane. Optional: Switch to the General tab, then in the Label field, enter a user-friendly name for the

To display a dropdown menu with a list of time zones: For a string-type variable or constant:

variable.

- 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.



Customiz ation Option	Steps in Design Time Tab		Result in Page Designer	
To limit the	Fo	r a number-type variable or constant:		
input values to a number in a range:	a.	In the Design Time tab, specify a hint text for the variable, for example, Enter Quantity, in the Placeholder field.	Enter a number between 0 and 99.	# Number of items(
	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.		ol
	c.	Optional: Switch to the General tab, then in the Label field, enter a user-friendly name for the		

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"
            }
        }
    }
}
```

variable.

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:

- Open the fragment's JSON editor.
- 2. Update the variable or constant's definition by setting the <code>@dt</code> element, then use the <code>subtype</code> 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 <code>subtype</code> property:

```
"variables": {
9
10
         "myObject": {
           "type": "string",
11
           "input": "fromCaller",
12
13
            "@dt": {
           "subtype":
14
                           "businessObject"
15
                         ₽ "color"
16
         },
         "title": {

■ "date"

17
           "type": "stri 🖶 "date-time"
18
           "defaultValue 🗗 "dynamicContainer"
19
           "input": "frc 🖶 "dynamicField"
20
           "@dt": {
                        🚽 "dynamicField[]"
21
             "subtype": 🗐 "dynamicLayoutContext"
22
23

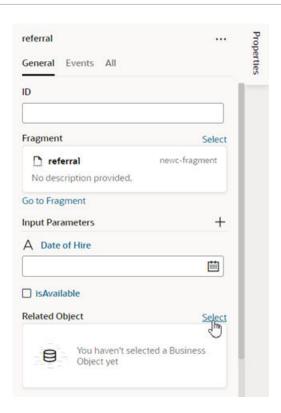
─ "endpoint"

                         🧬 "enum"
24
                        o a "lov"
25
                         ᆗ "time"
26
       "referenceable": "self"
27
28
```

For example, here's how you can show a component for selecting a business object by setting the subtype property to businessObject:

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:

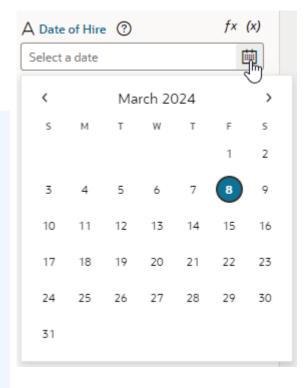
Property	Туре	Description
label	string	Use this property to specify a user- friendly name for the variable.



Property Subtype Available subtypes: businessObject color date date-time endpoint enum lov time timezone

Description

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 **Properties** pane:

- dyna micC onta iner
- dyna micF ield
- dyna
 micF
 iel
 d[]
- dyna micL ayou tCon text

For details, see Set the Binding



Property	Туре		Description
		Type for Variables in Dynamic Compone nts.	
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
dynamicCont ainer	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.



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:

```
{ } employee

{
    "active": false,
    "date-of-birth": "2001-01-01",
    "name": "Norman"
}
```



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:

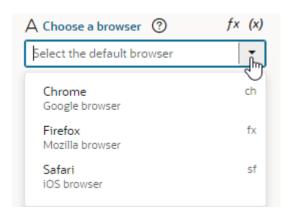
enum

values placeholder

Use <code>enum</code> to display a drop-down menu showing all values for each item in the array. The only <code>valueOptions</code> property for the <code>enum</code> subtype is <code>values</code>. In addition to a <code>value</code>, each item in the array can have an optional <code>label</code> and <code>description</code>.

```
"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:

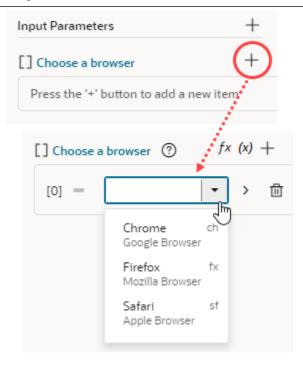


If you are using an array of primitives (say, string[])), you can use <code>enum</code> to display a drop-down menu showing all values for each item in the array:

```
"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:





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:

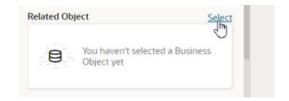


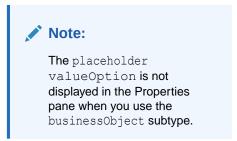
Subtype	valueOptions	Usage
empty	placeholder	Use the placeholder to specify a hint text for the variable.
	translatable	This can be used for all variables when there is no subtype.
		If a default value is supplied by the fragment variable, then that default value is used as the default placeholder. If 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:
		<pre>} "placeholder": "Search",</pre>
		"translatable": true
		}
business0	bj placeholder	Use the businessObject subtype to display a business object picker in the Properties pane.

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:



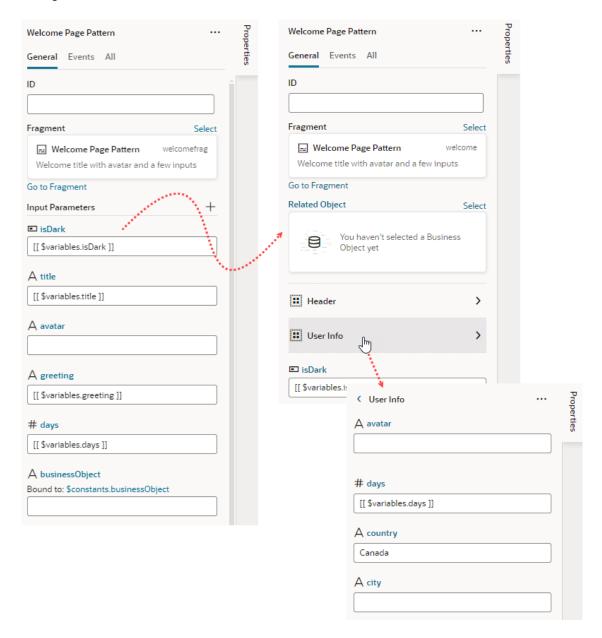


Section Fragment Properties for Display in the Properties Pane

Define a custom layout of sections to display a fragment's most important properties when the fragment is selected on the page or container that uses it.

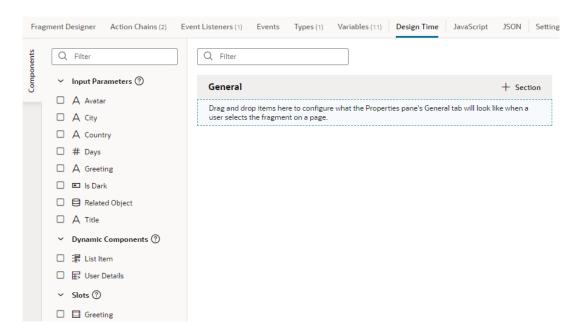
Typically, when a fragment is added to a page or container, its input parameters display in the Properties pane based on the order they appear in the fragment's source. To best represent the fragment's structure for those who use the fragment, you can further organize its input parameters—as well as 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 <code>Welcome Page Pattern</code> 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:

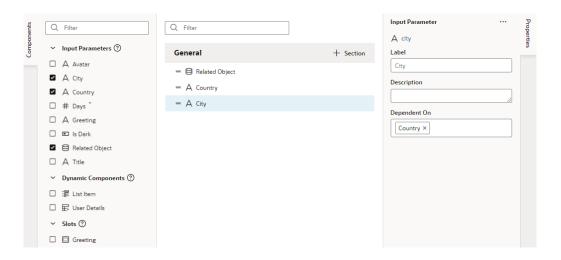


Open the fragment's **Design Time** editor.

- 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.

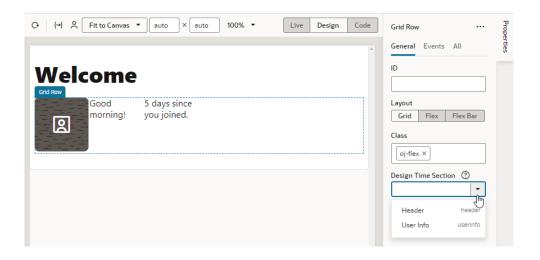




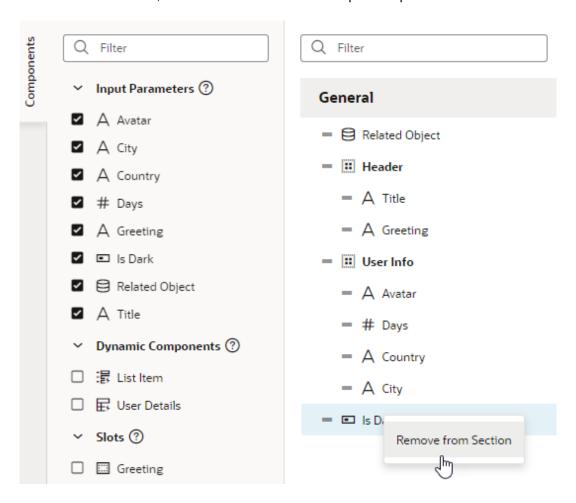
- To create a section and add items to it:
 - a. Click + Section, enter a section label in the pop-up (for example, User Info), and click Create.
 - b. Drag the items you want to add to the section and drop it onto the section header (for example, drag the avatar and days input parameters (among others) 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.





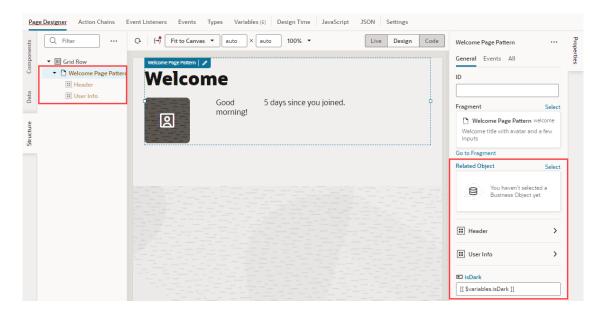
To remove a parameter you added from a section, right-click the parameter and select **Remove from Section**, or deselect the item in the Components palette.



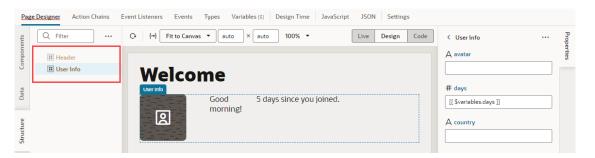
To delete a section, right-click the section and click **Delete Section**.

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.

Common Use Cases

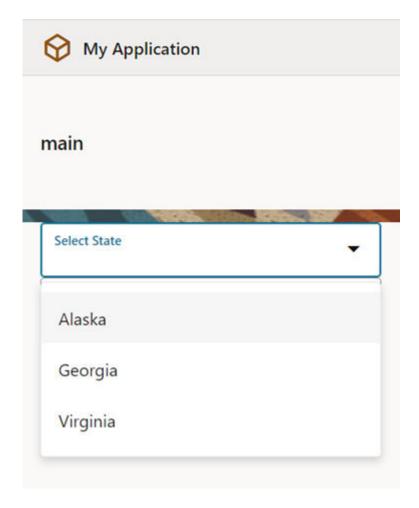
If you're familiar with using Oracle Cloud Application's Application Composer or Page Composer to create your apps, you may find these topics helpful as you grow accustomed to using VB Studio instead to build App UIs.

Populate a List of Values Based on Another LOV

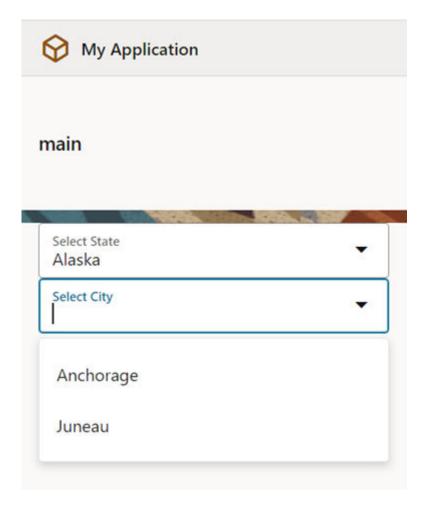
Within a dynamic form, you can use one drop-down menu to determine what appears in a second menu.

For example, suppose you want to present a list of states as a List of Values (LOV). When the user selects a state, a second LOV is populated with the cities in that state.

Here's what this might actually look like in your application: the first field, Select State, displays a list of states for users to select from:



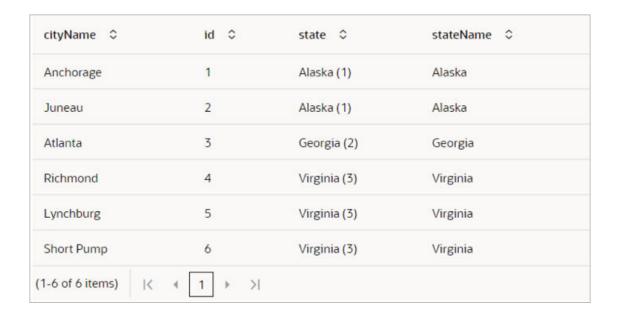
Once the user chooses a state—say, Alaska—the next field displays a list of Alaskan cities for the user to choose from:



For this to work, there must be a relationship between the data sources associated with each drop-down list. In this example, the data sources are two business objects (but they could come from another source as well, such as a service connection): state and city. Here's an excerpt of data from the state business object:



And here's some data from the \mathtt{city} business object:

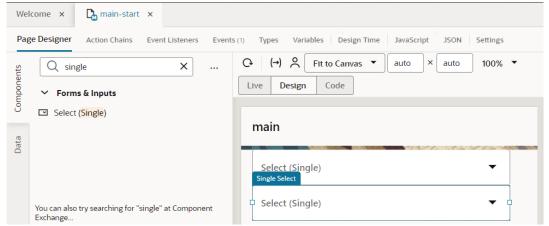


Let's take a closer look at the city business object—specifically, the state field. The person who created this business object set up this field as a *reference* type. A reference type field refers to a key field (in this case, the id field) in the state business object, to link the two business objects together.

Notice that the id field in the state business object assigns a unique number to each state, and that this number also appears in the city business object's state field. For example, Alaska has a value of 1 in the state business object, while cities IN Alaska have the value "Alaska (1)" in the city business object's state field. When VB Studio sees a city with Alaska(1) in the state field, it knows to grab the state with a value of 1 in the state business object. It's important to understand this relationship a bit so you'll understand what's going on as we move through this procedure.

To populate a list of values in a drop-down menu when an item in another LOV is selected:

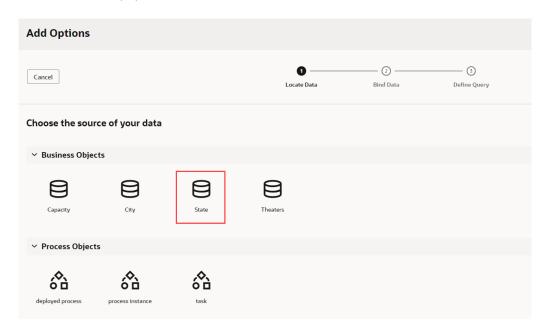
1. In the Page Designer, add a drop-down list component for each business object by dragging two Single Select (<oj-select-single>) components from the Components palette onto the canvas:



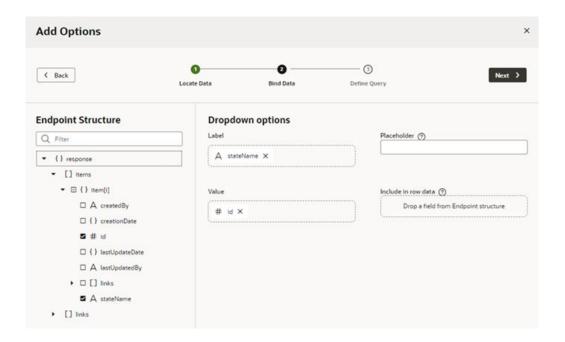
- 2. Now we need to tie these components to the data provided by the two business objects. Let's start with the first Single Select component:
 - a. Select the top component on the canvas, then click Quick Start in the Properties pane. Click Add Options:



b. On the Locate Data page, select the State business object, to indicate that we want this list of values populated with state names:



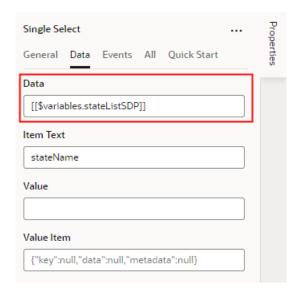
- c. Click Next.
- d. In the Bind Data page, drag the stateName field from the Endpoint Structure pane into the Label field.
- e. Drag the id field into the Value field.



Remember that the id field is the one that will tie this business object to the city business object.

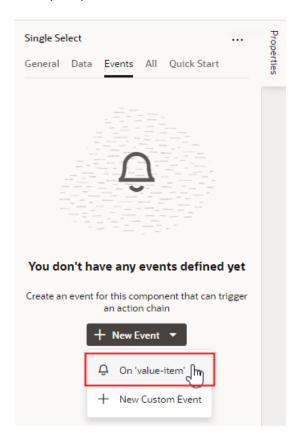
- f. Click Next.
- Click Finish on the Define Query page to close the Quick Start.
- Repeat these steps for the second Single Select component, this time choosing the city business object.

What actually retrieves the data from the business object? In this case, a Service Data Provider (SDP), which the Quick Start created for you. You can check this for yourself by checking the Data tab in the Properties pane:



Getting data from a data source can be tricky, so the Quick Starts automate this process by creating a special type of variable known as a Service Data Provider (SDP). Simply put, the SDP does whatever is required to fetch the correct data from the correct source.

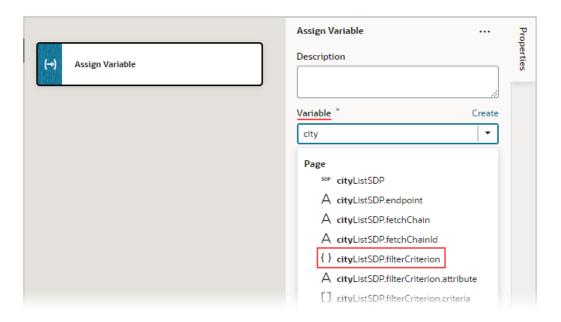
- 4. To create the action chain that filters the list of cities, we'll need a page variable that's assigned the selected state's ID after each selection. We'll use this variable to create a filter criterion for the city SDP (cityListSDP).
 - a. Select the state Single Select component, and on the Properties pane's **Data** tab, hover over **Value** and click (x) to open the Variables picker.
 - b. Click the Page node's **Create** link to create a new page variable.
 - c. For ID, enter stateID, and for Type, select Number.
 Now, each time a state is selected, its ID is saved to this variable.
- 5. Next, we need to create an event for the state Single Select component to start an action chain when the component's value changes. The action chain will filter out cities from the city component that aren't in the selected state.
 - a. On the canvas, select the state Single Select component.
 - b. In the Properties pane, click Events, then click New Event. Select On 'value-item' when prompted:



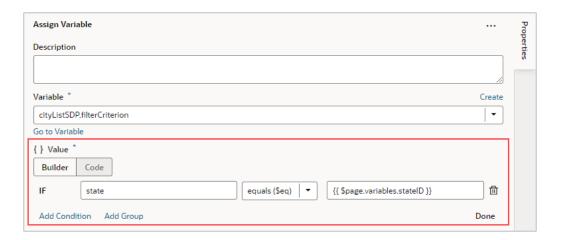
When you create the event, VB Studio automatically creates a new action chain and event listener for you. The event listener will listen for an event, which in turn will trigger a sequence of actions defined in the action chain.

- 6. Let's create the action chain to filter the list of cities shown in the second drop-down based on the state selected in the first drop-down.
 - a. In the Action Chain editor, add an Assign Variables action to the canvas.
 - b. In the Properties pane, type city in the Variable field, then select cityListSDP.filterCriterion:



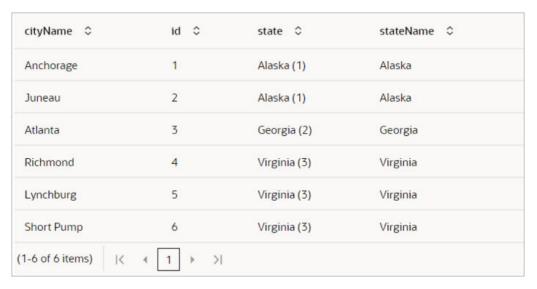


- c. For the Value property, click its Click to add condition link to create a criterion to filter out the cities that aren't in the selected state.
- d. In the condition builder:
 - i. Select state for the first Attribute drop-down list.
 - ii. Select equals (\$eq) for the Operator drop-down list.
 - iii. Type stateID in the second Attribute field, then select <code>spage.variables.stateID</code>. Recall, whenever a state is selected, this variable gets assigned its ID.



This condition essentially says, "For each row in <code>cityListSDP</code>, if the state ID equals the ID stored in the <code>stateID</code> page variable, add the row to the displayed list of cities." Remember the city business object from earlier?

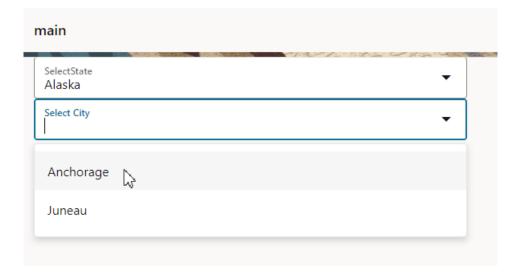




e. Click Done in the condition builder.



Let's preview the page by selecting the Page Designer's Live view. Select a state, then notice how the city drop-down list shows only the cities in that state:



If you open the list of cities *without* first selecting a state, the action chain won't be triggered, so the list won't be filtered. In other words, you'll see all the cities for all the states.

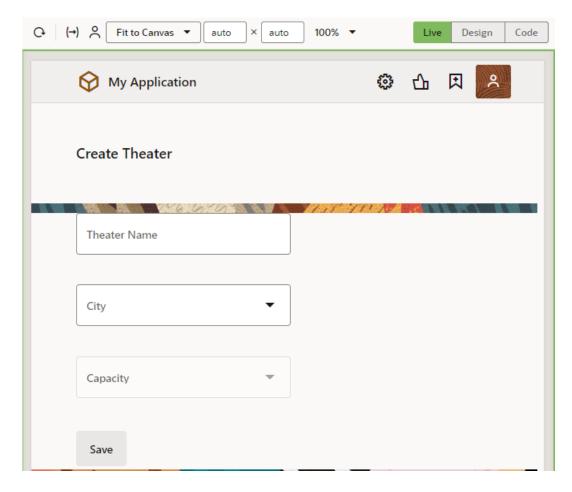
Selectively Enable a Field

Within a form (simple or dynamic), you can disable a field by default, and enable it only after a certain action or event occurs within the page.

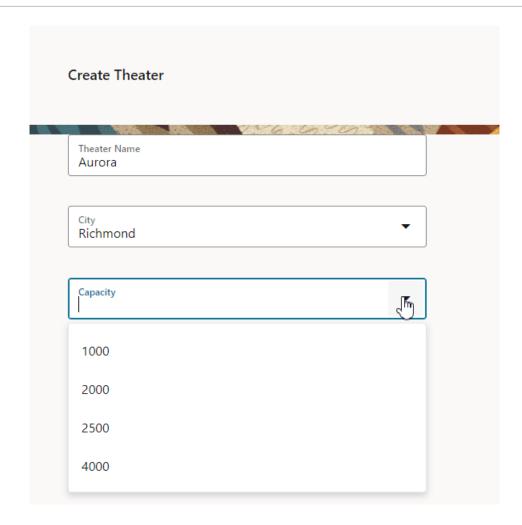
For example, suppose you have a form for entering data about new movie theaters, which contains three fields:

- An input field for the name of the theater
- A list of values (LOV) for the city where the theater is located
- An LOV for the new theater's capacity.

In this scenario, you want the Capacity field disabled by default (that is, grayed out), like this:



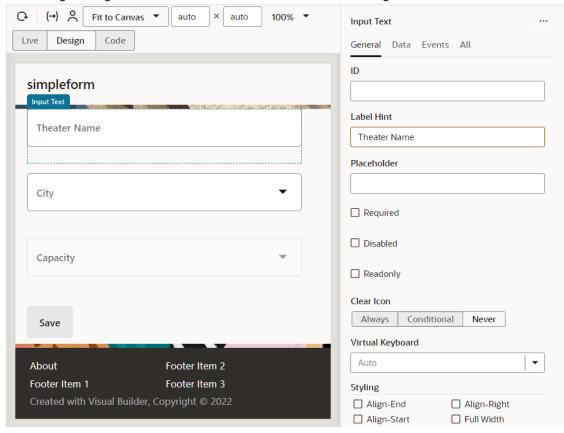
After the user selects a city, the Capacity field becomes active so the user can select a value:



When you want something to happen on a page based on the behavior of another component, either on that page or elsewhere, it almost always involves an *action chain*. Let's focus on how action chains work as we go through this scenario, so that you can apply your understanding to future use cases as well.

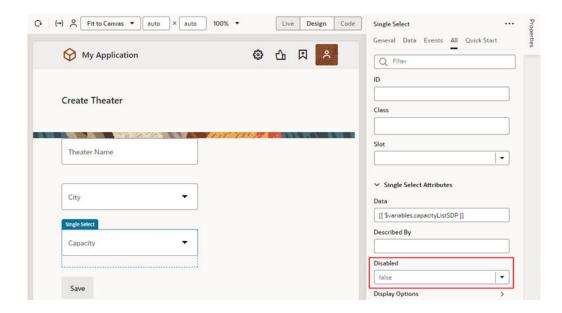


If you're working with a *dynamic form*, you'll need to work with its *rule set* to achieve the effect you want. See Selectively Display a Field to get an idea of how to do that.



In the Page Designer, let's assume that the form looks something like this:

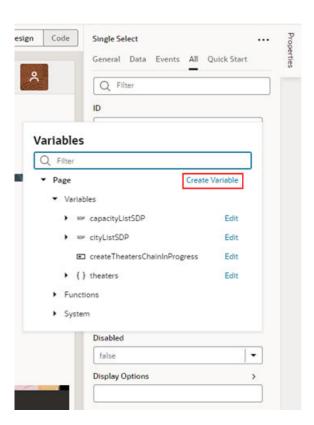
- 1. We'll start by disabling the component for the Capacity menu.
 - a. Select the Capacity field on the canvas, which is provided by a Single Select component (commonly used for LOVs).
 - **b.** In the Properties pane, click **All**, then locate the **Disabled** property:



The Disabled property is a Boolean, so the value must be either true or false. The default value is false, which means the component is *enabled* on the page by default.

But we want the field *disabled* by default, and enabled only when the user selects a city. The best way to implement this selective behavior is to change the Disabled field's value to a *variable*, rather than a static value. We can then use that variable in an action chain, which we'll get to later.

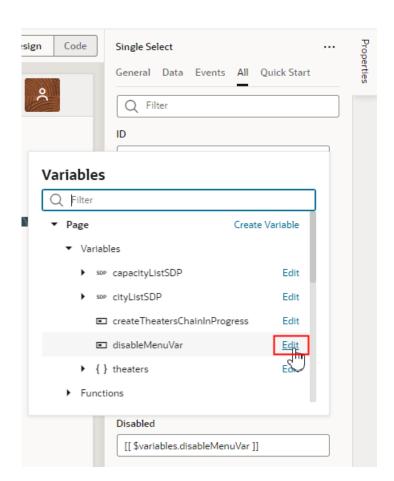
- 2. To create a variable for the Disabled property:
 - a. Click (x) on the Disabled property to open the Variables picker, then click **Create**:



b. Specify a name for the variable, perhaps disableMenuVar, and set the type to Boolean. Click Create:

Create Variable
Variable
ID *
disableMenuVar
Type *
Boolean ▼
☐ Input Parameter
Cancel Create

The new variable becomes the source of the Disabled property's value. Now let's set the value of this variable to true, meaning the field will be disabled by default.



c. Open the Variables picker for the Disabled property again, and click Edit next to the new variable:

d. On the Variables tab, set the new variable's Default Value to true.

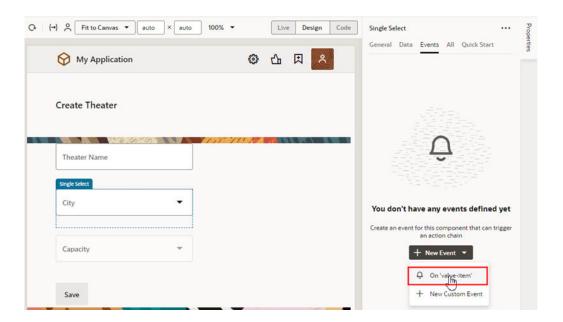
The Capacity field is now disabled by default. The next step is to set things up so the value of this variable changes when the user selects a city from the City field. We'll use an action chain to do that.

- To set up the action chain:
 - a. Click the Page Designer tab.
 - Click the City field.

Notice that we're setting up this action chain on the *City* field, because that's the field that impacts the behavior of another field; in our case, the Capacity field. An action chain always requires an *event* to serve as a triggering mechanism, so let's create one now.

c. In the Properties pane, click Events, then click New Event. Select On 'value-item' when prompted:

An On 'value-item' type of event essentially says, "When the value of this field changes (in this case, the City field), kick off the associated action chain."



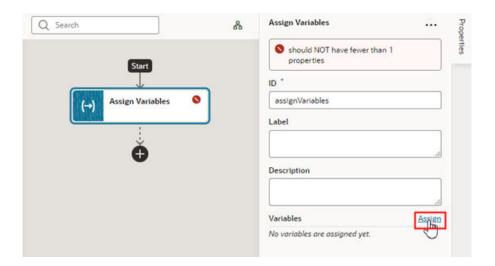
The Action Chains editor opens automatically.

When you create an On 'value-item' event, VB Studio automatically creates a new action chain for you called SelectValueItemChangeChain, as well as an event listener that listens for the On 'value-item' event. When the event listener "hears" the event, it starts the sequence of actions defined in the action chain.

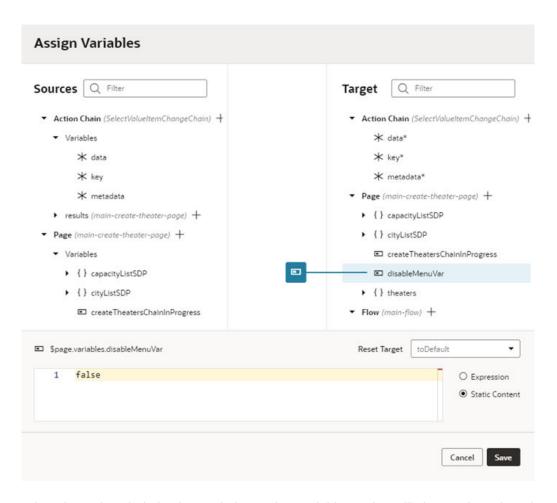
d. In the Action Chains editor, drag an **Assign Variables** action from the Actions palette and drop it onto the plus sign under Start.

As the name implies, the Assign Variables action assigns a value to a variable. The value could be an expression, but in this case we want to set the value of our disableMenuVar variable simply to false, a static value.

e. In the Properties pane, click **Assign** to select the variable we want to modify and define its new value:



f. In the Assign Variables window, select disableMenuVar in the Target pane, then type false at the bottom of the window. Make sure that Static Content is selected, then click Save:



When the action chain is triggered, the Assign Variables action will change the value of the disableMenuVar variable to false. This will change the value of the Capacity menu component's Disabled property, so that the component will become enabled on the page.

You can now Preview the page to check your work. Notice that the Capacity field is disabled by default. Select a city, and see that the Capacity field is enabled. Success!

Let's do a quick re-cap of what's happening behind the scenes:

- The user selects a value from the City field.
- The event listener detects that an On 'value-item' event has occurred, and fires off the SelectValueItemChangeChain action chain.
- The action chain has one action, which is to assign a value of false to the disableMenuVar variable.
- Because the Disabled property for the Capacity field is controlled by the disableMenuVar variable, and that value is now false, the Capacity field is enabled.

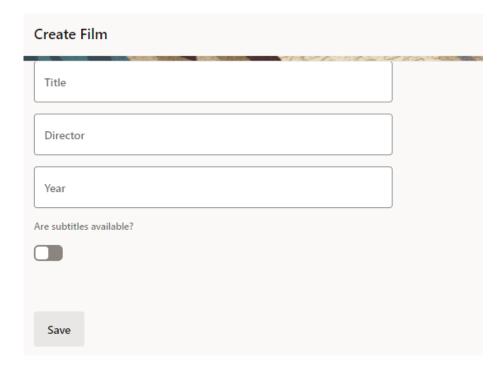
Selectively Display a Field

Within a dynamic form, you can hide a field by default and make it visible only after something else happens in the page.

For example, suppose you've created a dynamic form for entering data about films with these fields:

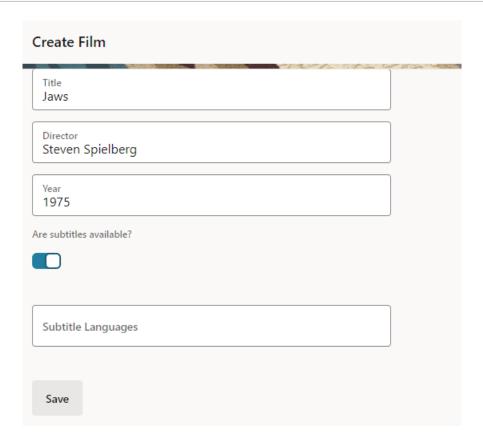
- Film title, which is an input field
- Director, also an input field
- Year of release, input field
- A toggle switch, which indicates whether subtitles are available
- An input field for entering the available subtitle languages.

In this scenario, you want the field for entering subtitle languages to be hidden when no subtitles are available, like this:



When the user changes the **Are subtitles available?** toggle to On, the field for entering subtitle languages becomes visible:

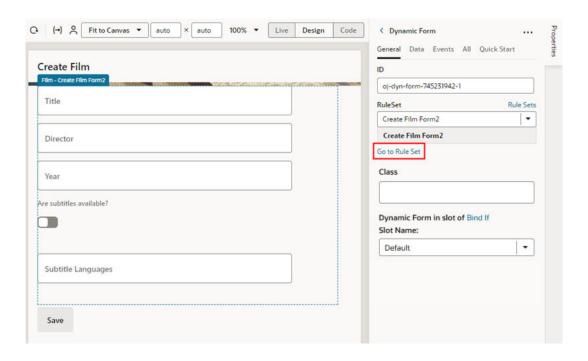




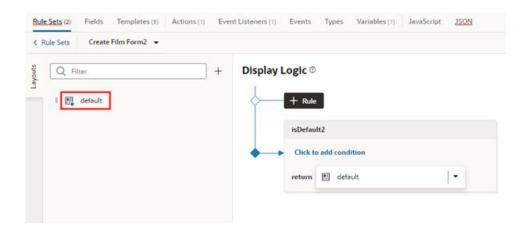
Dynamic forms are controlled by the display logic you specify in a *rule set*. A rule specifies both a *condition*—in this case, whether the toggle switch is set to On—as well as a *layout* to apply to the dynamic form at runtime. In this case, that layout will show the <code>Subtitle Languages</code> field in the form.

After reading through this topic, you should have a basic understanding of how rule sets and layouts work, which you can apply to real-world tasks you may have at your site.

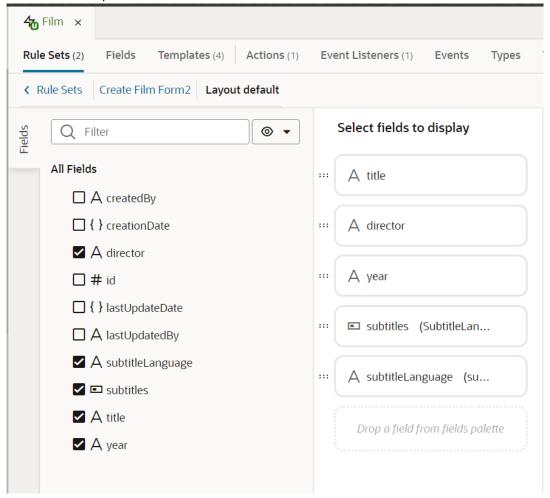
1. In the Page Designer, click anywhere in the form to display its properties in the Properties pane, then click **Go to Rule Set**.



2. In the form's rule set, click **default** in the Layouts pane to open the layout:



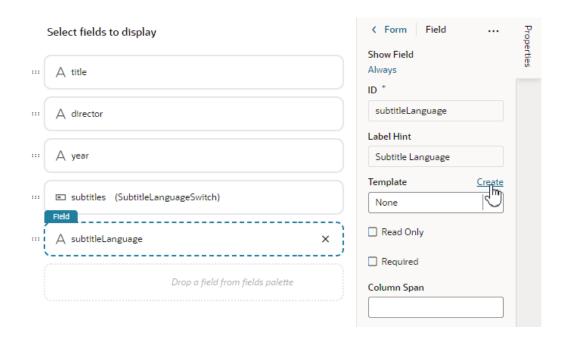
The Fields tab opens:



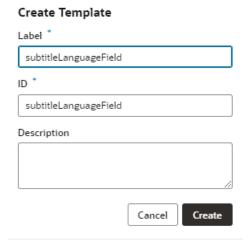
In this tab, you can see all the fields that *could* be added to this dynamic form on the left, as well as the fields that are included in the default layout: title, director, year, subtitles, and subtitleLanguage.

Only a few field properties can be set directly in a dynamic form's layout. When you need full control over how fields are rendered in a form, you need to use a *field template* instead.

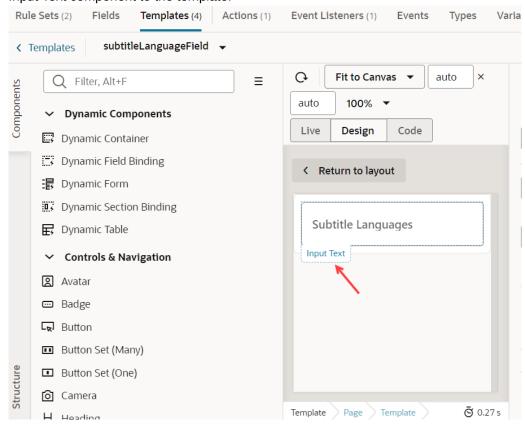
- Let's create a field template for the subtitleLanguage field to give it the hide/show behavior we're after.
 - a. In the rule set layout, click **subtitleLanguage** in the center pane,then click **Create** next to **Template** in the Properties pane:



b. In the Create Template dialog box, enter a label for the new template, like subtitleLanguageField, then click **Create**:

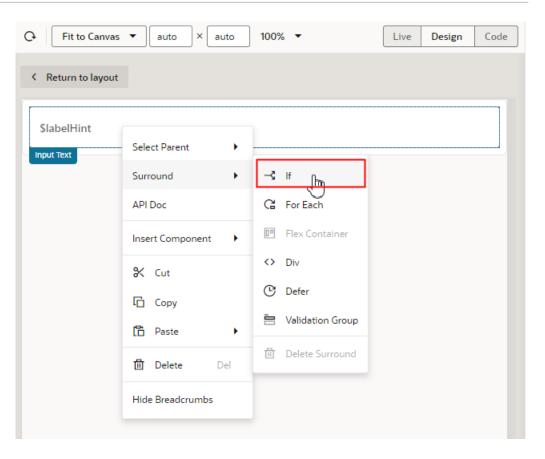


The new template opens in the template editor.



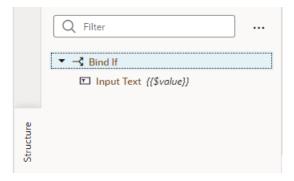
subtitleLanguageField is a text field, so the first thing VB Studio does is to add an Input Text component to the template:

- 4. Now let's edit the template to hide subtitleLanguageField when the switch is set to Off.
 - a. In the template editor, right-click the Input Text component, then select Surround > If in the pop-up menu:



This wraps the Input Text component in a Bind If component, which is better suited for controlling the visibility of a component. Bind If comes with a Test property; any content surrounded by Bind If is displayed only when the Test property is *true*. Let's set up the Test property next so we can see how this works.

- b. Click the **Structure** tab so you can work with the Bind If component more easily.
- c. Click Bind If in the Structure view:

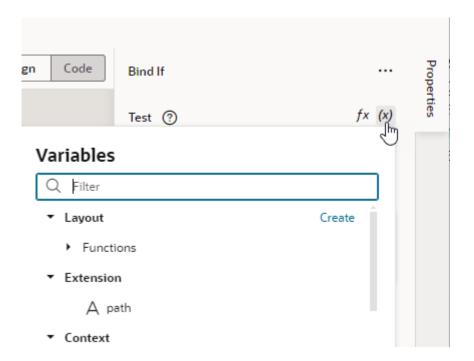


d. In the Properties pane, hover to the right of the **Test** property and click (x) to open the Variables picker.

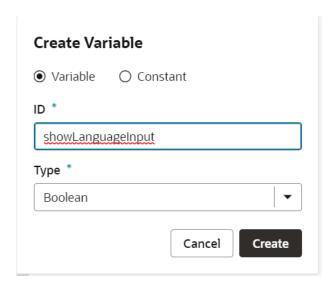
The Test property is used to hide or display the content surrounded by a Bind If component—which, in this case, is the Input Text field for selecting languages. The Test property type is Boolean, so the value must be either *true* (which displays the contents of the Bind If component) or *false* (which means the contents are not rendered). *True* is the default.

How are we going to hook up the user's action—that is, switching the Subtitles toggle from On to Off—to actually hiding or showing this field as a result? By defining a variable to represent the toggle's current state, then using that variable in an *action chain*. When you want something to happen on a page based on the behavior of another component, either on that page or elsewhere, it almost always involves an action chain.

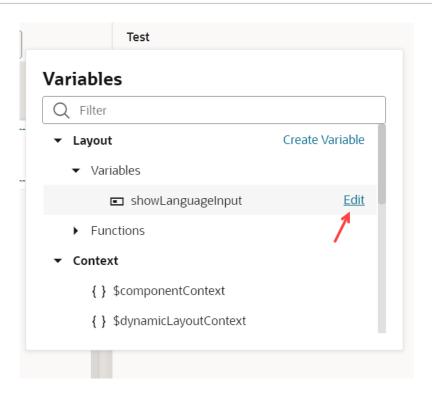
e. Click Create:



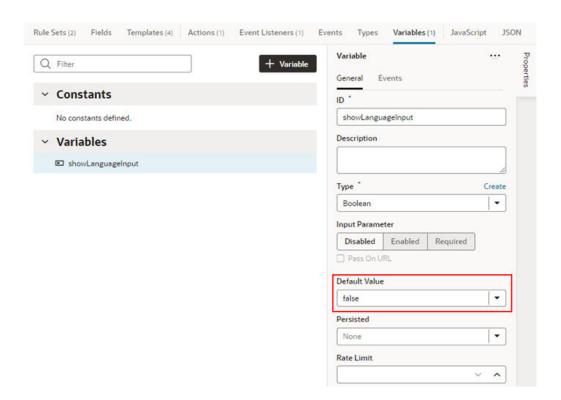
f. Specify a name for the variable, for example, showLanguageInput, set the type to Boolean, then click Create:



g. Open the Variables picker again and click **Edit** next to the showLanguageInput variable to open the Variables tab in the Designer:



h. In the Variables tab, set showLanguageInput's default value from *true* to *false*, so that the Bind If's component (the Select Languages field) is hidden by default:

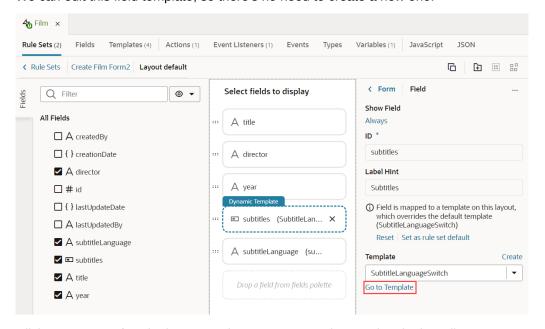


Now we're going to use this value in an action chain, to change its value when the toggle moves from Off to On, which in turns sets the display value from *true* to *false*.

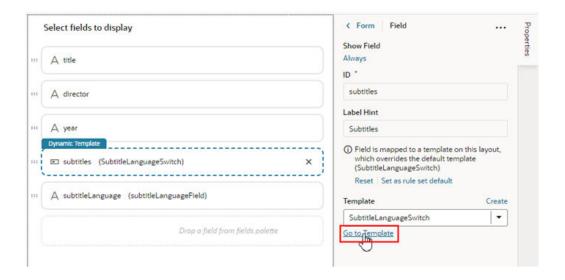
5. Let's create the action chain.

- a. Click the Rule Sets tab in the Designer to go back to the default layout.
- b. Select the subtitles toggle field in the center pane.

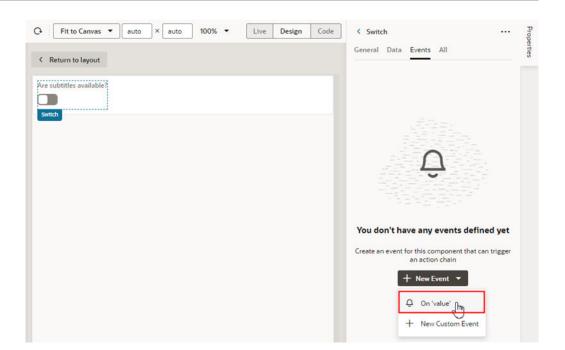
Notice that a developer has already created a field template, called (SubtitleLanguageSwitch), and applied it to the toggle field. The template name appears both in parentheses after the field name in the center pane, as well as in the drop-down list in the **Template** section of the Properties pane. We can edit this field template, so there's no need to create a new one.



c. Click **Go to Template** in the Properties pane to open its template in the editor:



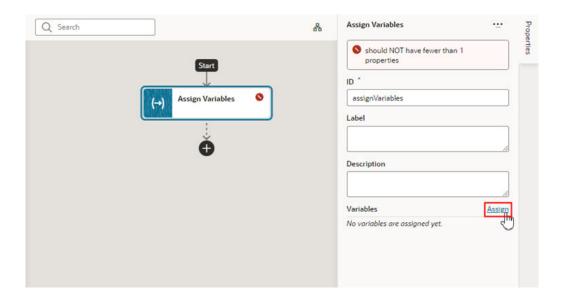
- d. In the template editor, select the Switch component on the canvas, then click the Events tab in the component's Properties pane.
- e. Click New Event, and then select On 'value' when prompted:



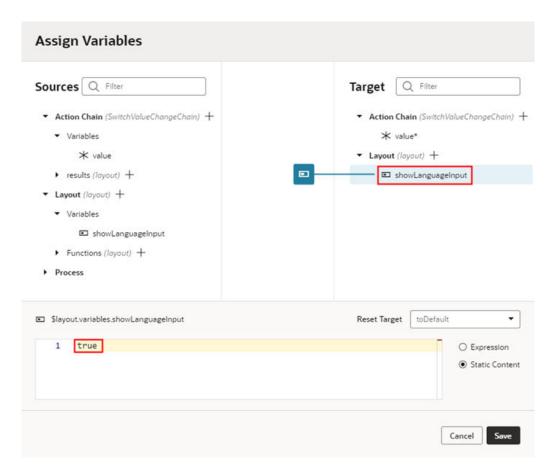
When you create the <code>On 'value'</code> event, VB Studio automatically creates a new action chain for you (in this example, it's called <code>SwitchValueChangeChain</code>), as well as an event listener that listens for the <code>On 'value'</code> event. When the event listener 'hears' the event, it will start a sequence of actions defined in the action chain.

In this example, the <code>On 'value'</code> event happens when a user uses the switch component to change the <code>subtitle</code> field's default value.

- f. In the Action Chain editor, drag an Assign Variables action from the Actions palette and drop it onto the plus sign under Start.
 - The Assign Variables action assigns a value to a variable. The value could be an expression, but in this example we want to set the value of our <code>showLanguageInput</code> variable to *true*, which is a static value.
- g. Select the Assign Variables action in the editor, then click **Assign** in the Properties pane to open the Assign Variables window:



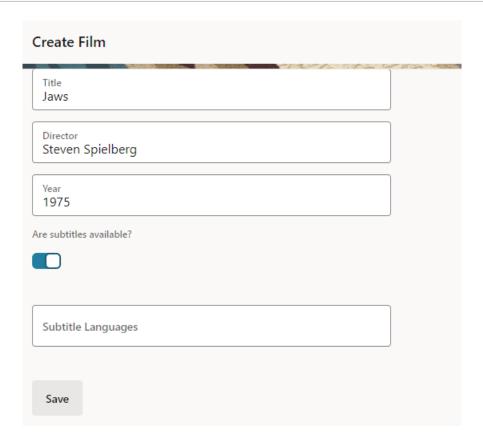
h. In the Assign Variables window, select showLanguageInput in the Target pane, then type true at the bottom of the window. Make sure that Static Content is selected, then click Save:



In the Properties pane, the Variables section shows that the showLanguageInput variable is now mapped to a value:



You can now view the page in Live view to check your work. Notice that the Subtitle Languages field is hidden by default. If you toggle the Are subtitles available? switch to On, the Subtitle Languages field is displayed:



Let's do a quick re-cap of what's happening behind the scenes:

- The user toggles the Are subtitles available? switch to On.
- The event listener detects that an On 'value-item' event has occurred, and fires off the SwitchValueChangeChain action chain.
- The action chain has one action, which is to assign a value of *true* to the showLanguageInput variable.
- Because the Bind If component's Test property is set to the value of showLanguageInput, and that value is now true, the Subtitle Languages field (which is wrapped in the Bind If component) can now be displayed.

Add Offline Support for Your App UI

Your App UI can function even when it's disconnected from a network. To do this, you use the Oracle Offline Persistence Toolkit which enables your App UI to cache data on the client and serve it back from the cache when your device doesn't have access to the server.

The Offline Persistence Toolkit is a client-side JavaScript library that enables data caching and offline support at the HTTP request layer. This support is transparent to the user and is done through the Fetch API and an XHR adapter. HTTP requests made while the client or client device is offline are captured for replay when the connection to the server is restored.

Using the toolkit, you can configure your App UI to:

 Download content for offline reading where connectivity isn't available. For example, an App UI 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. The salesperson, for example, could
 visit a site with no Internet access and enter an order for some number of product items.
 When connectivity returns, the App UI 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 App UI can configure
 a message asking the salesperson to cancel the order or place the item on back order.

✓ Note:

It's important to implement offline caching with an understanding of the risks involved. For example, cached data can get stale and out-of-sync with what's on the server. So while caching can improve performance, you might want to use it for data that doesn't change often. Also because data is cached on the client, anybody with access to the client would have access to the data as well. So make sure you're not caching sensitive data.

To use the toolkit in your App UI, you update the App UI's app.js file to include an OfflineHandler() function that determines the scope of data to cache, what type of caching strategy to use, and so on.

Note:

Adding offline capabilities to your App UI requires JavaScript knowledge and an understanding of the toolkit, so proceed carefully. Also, responses from REST services to your App UI 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 properly. Work with administrators of the REST services, so that values in the Cache-Control HTTP header are configured appropriately.

The following commented <code>app.js</code> file demonstrates one scenario of how you might implement caching for offline capabilities. It also demonstrates how you enable the toolkit's logging capabilities while you develop the App UI that uses the toolkit. Enabling this type of logging during the development phase will help you understand what data the toolkit caches in your App UI. Disable the logging functionality when you are ready to publish your App UI 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 offline persistence toolkit as an open-source project. For more information, see the toolkit's README.md and Wiki on Github at https://github.com/oracle/offline-persistence-toolkit. You can also access the API documentation directly at https://oracle.github.io/offline-persistence-toolkit/index.html.

Abort Pending REST Calls in VB Studio

When a REST call to your App UI 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 your App UI. 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/publicWorkers/{publicWorkers_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.



The AbortController API is *not* supported for App UIs that are configured for offline capabilities.

- 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 App UI's pages.
 - Select your App UI node, then click JavaScript to open the app-level JavaScript editor
 - **b.** Add this JavaScript snippet to the editor:

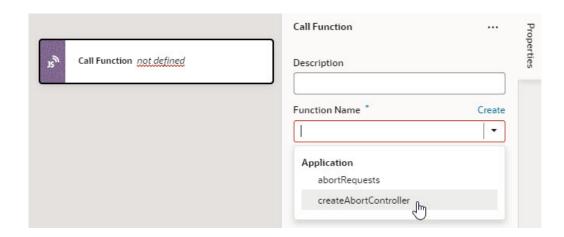
```
/**
  * Method to create an instance of AbortController.
  */
createAbortController() {
  return new AbortController();
}

/**
  * Method to invoke the abort method on the AbortController instance.
  */
abortRequests(abortController) {
  abortController.abort();
}
```

Your JS editor might look something like this:

```
🖵 demoappui 🛛
         Action Chains
                       Event Listeners
                                       Events
                                                Types
                                                       Variables
                                                                  JavaScript
                                                                             JSON
      define([], () => {
 1
 2
         'use strict';
 3
        class AppModule {
 4
 5
          * Method to create an instance of AbortController.
 6
 7
 8
          createAbortController() {
 9
            return new AbortController();
10
11
12
            * Method to invoke the abort method on the AbortController instance.
13
14
          abortRequests(abortController) {
15
            abortController.abort();
16
17
18
19
        return AppModule;
20
21
      });
22
```

- 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 wbEnter event for the application.
 - Click Event Listeners at the app level.
 - b. Click + Event Listener.
 - c. Select vbEnter under Lifecycle events and click Next.
 - 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.



- Now drag and drop an Assign Variable action to follow the Call Function action.
- In the Assign Variable action's Properties pane, select abortController under Application from the Variable list.



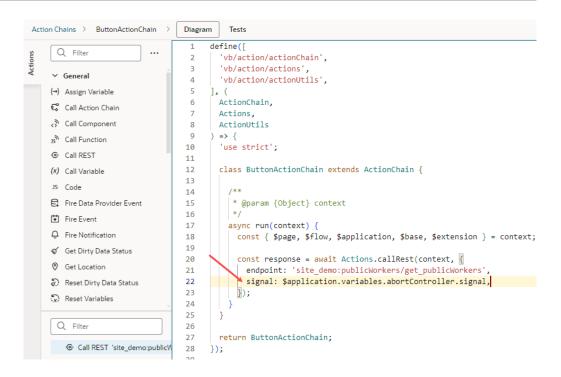
- j. Hover over the **Value** property and click (x) to open the Variables picker, then select createAbortController under Local to assign the value returned by the createAbortController function to your abortController variable.
- 4. Associate the AbortController with the REST call you want to abort. 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/publicWorkers/{publicWorkers_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:

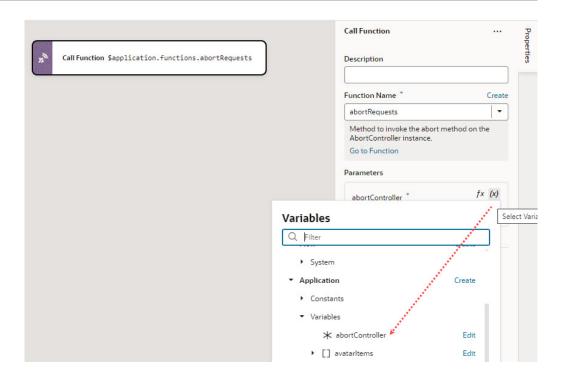
signal: \$application.variables.abortController.signal,

In this example, the AbortSignal is accessed via \$application.variables.abortController.signal and attached to the get_publicWorkers REST request:

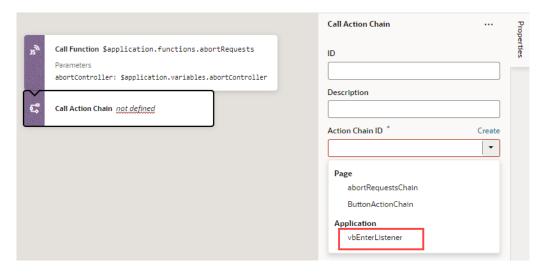




- 5. To abort REST requests with the AbortSignal attached, build an action chain to call the abort method on the AbortController instance. Because our example assumes a Cancel button that users click to abort the Get/publicWorkers/{publicWorkers_Id} request on a page, we define the abort action chain at the page level (but you can also define it at the app level).
 - a. Click Action Chains to open the Action Chains editor.
 - 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.

Part V

Troubleshooting

These topics cover some common issues and how to address them.

Topics

Troubleshooting and FAQs



Troubleshooting and FAQs

These topics cover some common issues and how to address them.

Resolving the Error "Page cannot be previewed"

If you're having trouble seeing the preview of your app extension page in the Page Designer, you might need to check your browser settings to make sure that the browser isn't blocking your VB Studio instance from accessing your Oracle Cloud Application instance. This can happen when the instances are not in the same host domain and your Chrome browser's settings are set to the default.

You might see something like this in the Page Designer instead of seeing the app extension page:





Page cannot be previewed

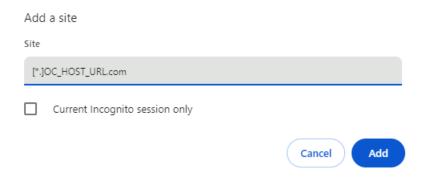


To resolve this page preview error, you'll need to edit some of your Chrome browser settings.

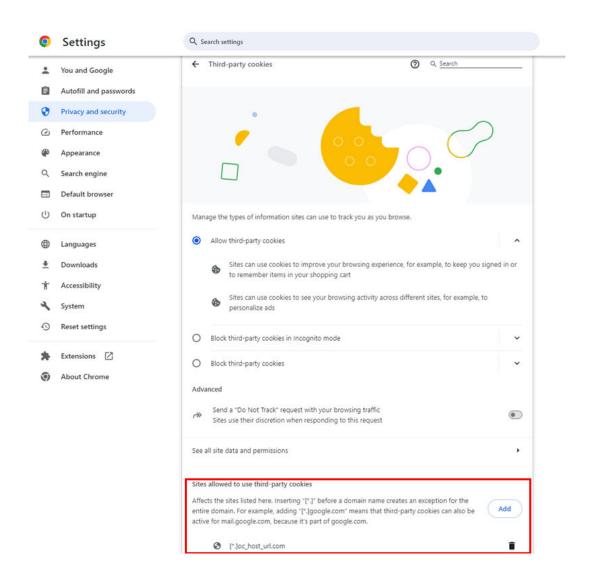
- Note the host domain of your Oracle Cloud Application instance.
 - For example, if the URL of your Oracle Cloud Application is something like os1.mysubdomain.OC_HOST_URL.com, the host domain is OC_HOST_URL.com.
- Open the browser's Cookies settings page (chrome://settings/cookies) and disable Block third-party cookies, if it's not already disabled.
 - Selecting **Allow third-party cookies** disables the "Block third-party cookies" and "Block third-party cookies in incognito mode" options.

Click Add next to the "Sites allowed to use third-party cookies" option, then type your Oracle Cloud Application host domain in the Add a site dialog box. Click Add.

Use the syntax $[*.]OC_HOST_URL.com$ for your host domain in the dialog box to allow cookies from your host's subdomains.

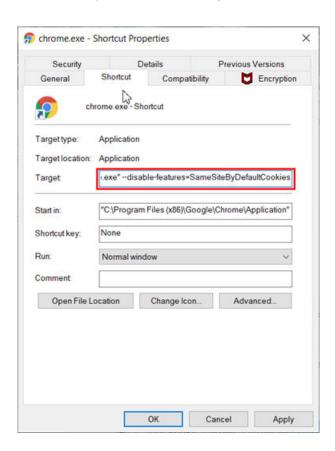


Confirm that your host domain is included in the list of sites that can use third-party cookies.



4. Disable the "SameSite by default cookies" flag option by adding --disable-features=SameSiteByDefaultCookies to the browser's startup parameters.

If you're using Chrome on a Windows machine, you can create a shortcut to Chrome and then add the parameter to the Target field in the shortcut's Properties window.



If you're using Chrome on MacOS, you can launch Chrome with the parameters from a Terminal window. To launch Chrome, open a Terminal window and type '/Applications/Google Chrome.app/Contents/MacOS/Google Chrome' --disable-features=SameSiteByDefaultCookies.

If you're starting Chrome from the command line, you can add the parameter when you launch the application (for example, /opt/google/chrome/chrome --disable-features=SameSiteByDefaultCookies).

5. Restart your browser and open the page again in the Page Designer.

After restarting your browser, you can open <code>chrome://version/</code> in your browser and check the Command Line properties to confirm that "SameSite by default cookies" is disabled.





How Do I Clear My Extension's Resource Cache?

Oracle Support might sometimes ask you to clear your extension'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 Recover My Extension After Deleting the Project?

Even after deleting a project, you can still recover your deployed extension's source files, if needed. Use the script provided below to retrieve the source files, which you can then import into a new workspace for continued testing and configuration work.

Before running the script, make sure you have these details:

- The deployed extension ID and extension version (If you need help finding this information, file a service request and include as many details about your deployed extension as possible, such as the base application you're configuring and deployment date.)
- The URL of the environment where the extension was deployed
- Credentials of a user who can access the Oracle Cloud Applications instance

You'll also need to be able to edit and run a Python script, or use curl to duplicate what the Python script does. If running this script in Windows, you must have curl installed.

Copy this Python script and save it as get extension files.py:

```
Usage: python3 get_extension_files.py
What is this? A python3 script to retrieve customer extension source code files for a specific extension identified by its extensionID and extensionVersion.

How to configure this script before using it:
Set these variables: userid, pw, extensionID, extensionVersion, envURL
To run this script in Windows
- curl must be installed
```

```
- set the Windows variable to True
import json
import requests
from requests.auth import HTTPBasicAuth
import os
userid=""
pw=""
envURL=""
extensionID=""
extensionVersion=""
Windows=False # set to True if running this script in Windows, else set it
to False
curlfile="curlfile.sh"
if Windows:
  curlfile="curlfile.bat"
#get extension file list
response = requests.get(envURL+"/fscmRestApi/vx/v2/
extensions/"+extensionID+"/"+extensionVersion, auth=(userid, pw))
print ( response )
response dict = response.json()
print(json.dumps(response dict, indent=4)) #pretty print the json received
through the above GET cmd
#write a shell script file containing the curl commands to download the
necessary extension files, while recreating the exact directory structure
of the deployed extension
if os.path.exists(curlfile): os.remove(curlfile)
baseURL=response dict["baseUrl"]
for x in response dict["files"]:
  # ignore files in the extension-digest folder and bundles folder, and
ignore build-info.json.
  if not (("bundle" in x) or ("build-info.json" in x) or ("extension-
digest" in x)):
    with open(curlfile, 'a', encoding="utf-8") as f:
        if Windows:
            newX=x.replace("/","\\")
            f.write("curl -u \"" +userid+ ":" +pw+ "\" --create-dirs -o "
+response dict["id"]+ "\\" +response dict["version"]+ "\\sources\\" +newX+
" " +baseURL+ "/" +x+ "\n\n")
        else:
            f.write("curl -u \"" +userid+ ":" +pw+ "\" --create-dirs -o "
+response dict["id"]+ "/" +response dict["version"]+ "/sources/" +x+ " "
+baseURL+ "/" +x+ "\n\n")
#run the shell script
if Windows:
  os.system(curlfile)
else:
  os.system("sh " +curlfile)
```

- 2. In the script, define the **userid**, **pw**, **envURL**, **extensionID**, and **extensionVersion** variables using your extension-specific values.
- 3. Set the Windows variable.

4. Run the script (recommended version is Python 3). For example:

```
python3 get extension files.py
```

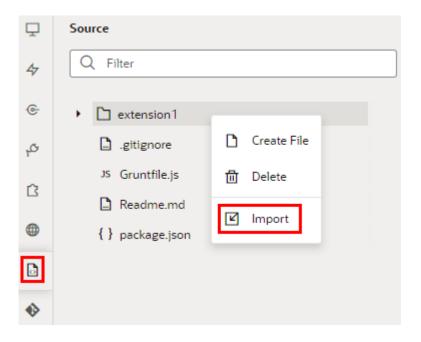
The script creates a directory structure for the extension and extracts the source files into a sources subdirectory.

5. Create a .zip file of the sources subdirectory, only.



On a Mac, don't use the Finder to do this because it will create an archive that's incompatible with VB Studio.

- In VB Studio, create a workspace with a scratch Git repository. We're using a scratch repository in this example, but you can use any Git repository that fits your own best practices.
- Open the Source tab, right-click extension1, and click Import to import the sources.zip file into your workspace.

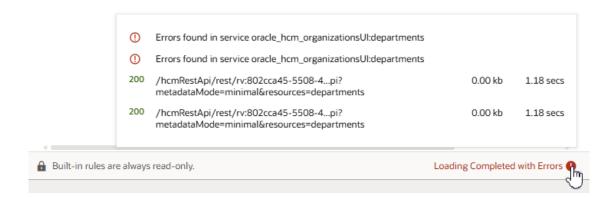


The extension is now available in VB Studio for testing and validation.

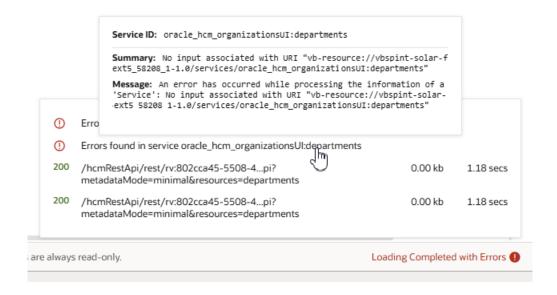
Resolve the "Loading Completed with Errors" Message

When you open a page in the Business Rules editor, VB Studio loads all the services that furnish the page's data in the background. If an error is encountered during the loading process, you'll see the message "Loading completed with errors" at the bottom of the page.

If you see this message, click "!" in the status bar to display more detailed information about the services affected:



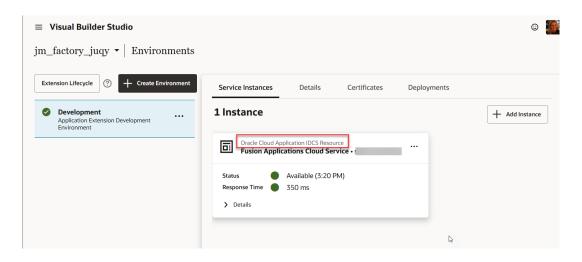
The list might also include information about data fetches that are unusually large, which might cause the editor to load slowly. By hovering over a particular error in the list, you can drill down even further:



It's important to note that even if there were problems retrieving data from one or more services, the Business Rules editor will still show data from the services that are loaded successfully. In other words, your page might be missing regions or fields from some services, while still displaying content from healthy services.

If a service can't be loaded, it's usually because there are issues either in your environment or in CORS. To troubleshoot:

 Check the environment associated with your workpace. (If you're in the Designer, click the arrow at the top left, then click **Environments** in the left navigator.)
 Make sure your status indicator is green, and that your Oracle Cloud Applications instance was added as an IDCS resource (as opposed to OAuth or Basic Auth):



If both conditions are true, continue with the next step. If not:

- If your status is anything other than green AND your instance is an IDCS resource, there may be a more complex underlying issue. File an SR for further assistance.
- If the instance was added with OAuth or Basic Auth, it's best to create a new environment using the Identity Domain option, then switch your workspace's environment to the new one.
- 2. Make sure there aren't any CORS issues:
 - a. Open Chrome Developer Tools, then click the **Network** tab.
 - b. Open your workspace.
 - c. In the Network tab, look for the REST call: GET fscmRestApi/applcoreApi/v1/ applicationManifest/faStandalone?appRelativePath=fscmUI/redwood. You might also see an OPTIONS call for the same endpoint, which you can ignore.
 - d. If the call is successful, move to step f. If not, continue with the next step.
 - e. If the call is not successful:
 - Click Console in Chrome Developer Tools and find the full description of the CORS "Access to fetch" error:
 - If you see: Access to fetch at '<Oracle Fusion Cloud Applications endpoint>' from origin '<Visual Builder Studio>' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled., that means the Visual Builder Studio domain has not been added to Oracle Cloud Applications' CORS setting. See the row "Access-Control-Allow-Origin" in Configure Cross-Origin Resource Sharing for guidance. Also check to make sure there aren't any typos.
 - If you see: Access to fetch at '<Oracle Fusion Cloud Applications endpoint>' from origin '<Visual Builder Studio>' has been blocked by CORS policy: Request header field <some-specific-header> is not allowed by Access-Control-Allow-Headers in preflight response., that means a specific header field hasn't been added to Oracle Fusion Cloud Applications' CORS Allow Headers profile. See the row "Access-Control-Allow-Headers" in Configure Cross-Origin Resource Sharing for guidance. Also check to make sure there aren't any typos.



- ii. Make sure there aren't any typos in the CORS profile values, as header fields can't contain spaces. For example, in CORS_ACCESS_CONTROL_ALLOW_HEADERS, X-HTTP-Method-Override, and X-Requested-By is incorrect, due to the space before and after "and". Instead, you would specify ,X-Requested-By.
- f. If there aren't any CORS errors but fscmRestApi/applcoreApi/v1/ applicationManifest/faStandalone?appRelativePath=fscmUI/redwood fails with a 401, there may be a problem with your wiring to IDCS. Log an SR for further assistance.
- 3. If there aren't any problems with fscmRestApi/applcoreApi/v1/applicationManifest/faStandalone?appRelativePath=fscmUI/redwood, but there is a problem with one particular REST describe call, log an SR on the Oracle Cloud Application you're using for further assistance.



Part VI

Appendix

Work With Services (Limited Availability)

Here's a guick test you can do to see if this appendix applies to you:

- 1. In your browser's address bar, enter https://<oracle fusion base url>/api/ rwdinfra/trap, as in: https://abcd-test.fa.oraclecloud.com/api/ rwdinfra/trap.
- 2. If you get this response: {"2":{"links":[{"href":"2/"}]}}, you can use this appendix to set up your backends and service connections. If you don't get this response, your instance doesn't yet have this capability, so you should use Work With Services instead.

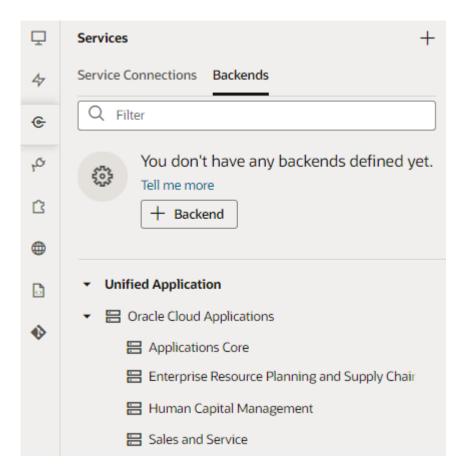
Many of the UI components in your App UI are bound to data that comes from REST APIs, which in turn are provided by *services*, like Oracle Cloud Applications. These services are made available to your extension through backends and service connections.

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 extension level, which means that all the App UIs in your extension can use them. In addition, you can use service connections provided through a dependent extension as a source of data for other App UIs in your extension, whether you're creating the App UI from scratch or configuring an existing App UI.

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. The Unified Application provides a backend called "Oracle Cloud Applications for all services", which points to the servers hosting the Application Core API, the Sales and Services API, and more:





While you can't modify the service connections or backends that come from dependent extensions (including the Unified Application), you can create your own and reuse them across all the App UIs in your extension.

Important: When creating your own service connection or backend, you must mark it as "Accessible to application extensions" if you want to allow their use by dependent extensions.

Manage Backends

Your extension has automatic access to some out-of-the-box backends, like Oracle Cloud Applications and its child backends. You can also create your own backends, as well as add another extension as a dependency and gain access to its backends.

What Are Backends?

Simply put, a *backend* describes the server that hosts a system you want to connect to. For example, the "Oracle Cloud Applications" backend represents the details of the Oracle Cloud Applications system, including the URL, authentication method, and so on, all of which are needed to gain access to that system's REST APIs.

All new service connections require a backend. If a backend doesn't exist, you'll be prompted to create one. You'll know when services are "derived" from a backend because you'll see something like this:

1 This service is based on the crmRestApi backend.



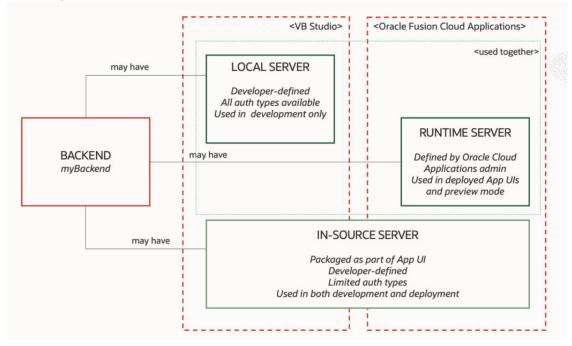
Click the backend link to open the Backends editor, where you can view the backend's details.

You can create your own backends to represent the systems you want to connect to. Creating backends for commonly used systems helps keep the information in one place, rather than duplicating it across various service connections. For example, instead of having multiple service connections for your company's LinkedIn account, each with the same URL, authentication method, and so on, you could define the details in one backend, perhaps called the "LinkedIn Backend".

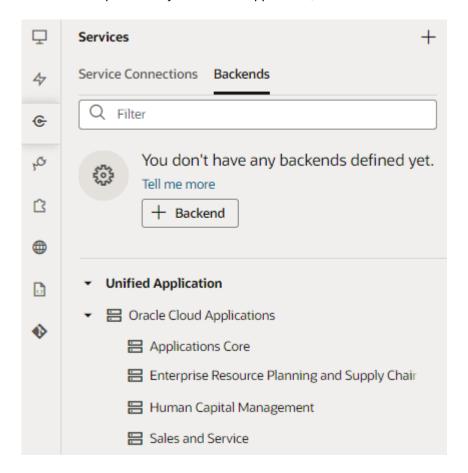
A backend server can have two types of definitions, which reflect the URL and authentication method you want to use:

- An *in-source* server definition provides a backend's details for use during both development and deployment. Its details live in an extension's source code (specifically in the catalog.json file) and is therefore packaged up as part of the extension itself. This means that the backend always points to the same URL and authentication method, irrespective of where you deploy the extension.
- A local server definition provides a backend's details for use during development only. Because its details aren't stored in an extension's source code, you can go beyond the limitations of an in-source server by defining the URL and authentication method for the particular instance you're deploying to. Typically, a local server is used in conjunction with a runtime configuration, specified in the Oracle Cloud Applications instance, which provides the backend's details for the deployed extension (see Provide a Backend's Runtime Server Details). You can also use a local server with an in-source server to override its details during development, although this is far less common.
 In VB Studio, a local server's details are used by the Page Designer's Live mode, while the server details provided by the runtime configuration and in-source server are used by the Preview action.

For example, suppose that during the development phase you want to use a mock URL. While testing you want to use a different URL, located in the test instance, and when you finally deploy you want to use the actual URL. You can achieve this using local servers, but not with in-source servers. The caveat is that local servers must be supplemented with a runtime configuration on the Oracle Cloud Application instance you are deploying to. The relationship between potential servers is shown here:



In addition to defining your own backends, you can also add another extension as a *dependency* and gain access to its backends as well (so long as those extensions have been marked as accessible to other extensions). Your extension also has automatic access to some out-of-the-box backends, like Oracle Cloud Applications and its child backends. These backends are provided by the Unified Application, as shown here:



Note:

At the moment, you can't add an extension as a dependency with the intent to use the extension's backends to create service connections. Until this is resolved, you must create service connections in the same extension as their associated backends.

These backends are provided so that you can use them to quickly create service connections to the REST APIs exposed by Oracle Cloud Applications. You'll need these service connections so you'll have data for things like configuring a dynamic container, or adding other components to the page that require a data source.

Note:

If you don't see the Unified Application as a dependency, contact your administrator to confirm that you have the proper credentials and that your user role is authorized.

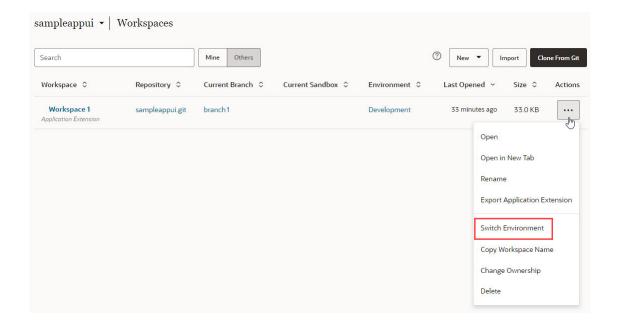
You can manage backends on the **Backends** tab, which you access from the **Services** tab in the Navigator. When you click a backend, you'll see four tabs on the right, each focusing on a different set of details:

Tab	Description	
Overview	Displays the name and type of the backend, which can be 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 the services connections that were defined from the backend. 	
	If you have the backend configured with a local server, you might also see a section that shows the runtime configuration on the Oracle Cloud Application instance you're deploying to. See What Are Local Servers? for more on this.	
Servers	Displays the server, including the instance URL, associated with the backend. You can display details for a backend server that comes from a dependency, but you can't edit it. You can edit the details for a backend server that you defined for your extension. The server can be either an in-source server, which is packaged within the extension, or a local server. Local servers are indicated by a Local server definition tag.	
Headers	Add and edit static headers, which pass information to the service, for the backend at the server level.	
Source	Displays the backend's description, which is stored in the <code>catalog.json</code> file of the extension or dependency that the backend is defined in. For backends that you define, the location is <code>services/self/catalog.json</code> , and the entries are editable. Local server details are not stored in the source code.	

The artifact that stores all the backend definitions for the extension is an OpenAPI-compliant file called *catalog.json*, which contains the list of backends and their details. If you want a backend to be usable by other extensions (that is, extensions who have added your extension as a dependency), the backend **must** be marked as "Accessible to application extensions" on the backend's **Overview** tab. The complete service catalog for your extension is a combination of all the backends that you have defined in your extension, plus any backends exposed by dependencies.

If your App UI needs access to a particular Oracle Cloud Applications instance—say, one that's running a different version of Human Capital Management—use your project's Environment's tab to add that instance to your Oracle Cloud Applications environment, then use **Switch Environment** to point your workspace to that environment (or just create a new workspace):





What Are In-Source Servers?

As the name implies, an *in-source* server definition lives in an extension's source code (specifically in the <code>catalog.json</code> file) and is therefore packaged up as part of the extension itself.

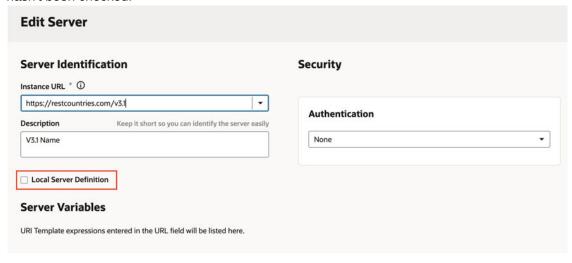
This means that the URL for the server is constant; you cannot change it during different phases of the development cycle as you can with local servers. In addition, the authentication methods available to in-source servers are limited to those that don't rely on credentials (like None, Oracle Cloud Account, and OAuth 2.0 User Assertion).

How can you tell if a server is an in-source server? For starters, it won't have the **Local server definition** tag on the Servers tab:





Additionally, if you click **Edit Server**, you'll see that the **Local Server Definition** checkbox hasn't been checked:

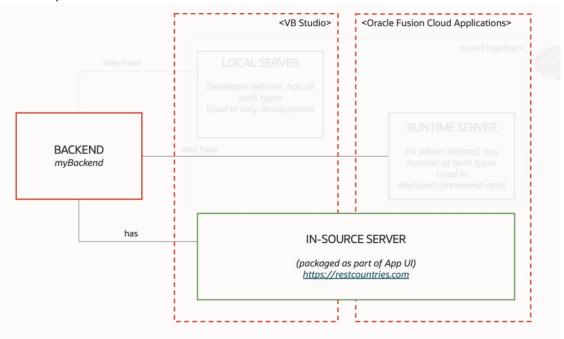


If you navigate to the Source tab in the Designer, you can see the server definition, including the URL and authentication type (if any):

Because an in-source server is packaged with an App UI's source code, it can support only very limited use cases, where there's no need to differentiate the URL based on the deployment instance. Publicly available REST APIs that don't require authentication, like https://restcountries.com, are good candidates for in-source servers. As shown in this diagram, the myBackend backend resolves to https://restcountries.com with no authentication at all times (in VB Studio, plus all Dev/Test/Prod Oracle Cloud Application



instances):



If you change the details of an in-source server (like the URL) after deploying your extension, you'll need to redeploy it so that the changes will take effect. (Changes to local servers and runtime configurations, on the other hand, take effect immediately; redeploying is not required.)

What Are Local Servers?

A *local server* lets you override an in-source server's details, so that you can provide a URL and authentication type that are more applicable during development, without those details going into the extension's source code.



To use local servers in conjunction with runtime server configurations, make sure your Oracle Cloud Applications instance is on 25A and that you've completed the check described in Work With Services (Limited Availability).

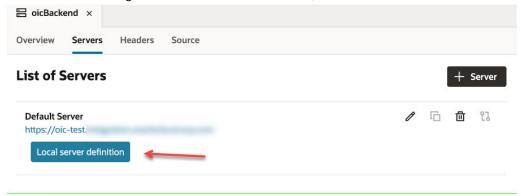
If a service connection's backend has a local server defined, it will be used when interacting with the Designer during development, like when you use the layout editor or the Page Designer's Live/Design mode. When you deploy or preview your App UI, however, the local server is **not** used. Instead, the URL and authentication method defined in the runtime server configuration are used. More about this later.

Note:

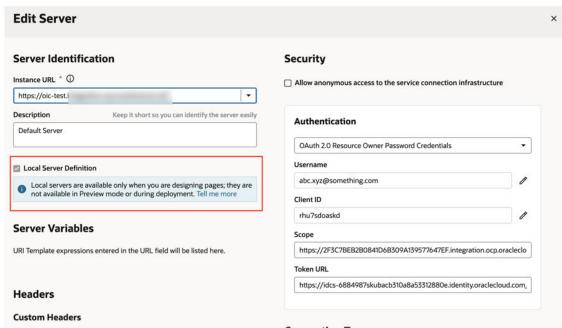
Service connection endpoints based on local servers currently only work in the Page Designer if they are GET-based and have the connection type **Always Use Proxy** in the local server definition.



You can tell that a local server has been defined for a backend because it has the **Local server definition** tag on the backend's **Servers** tab, like this:



You'll also see that the **Local server definition** check box has been checked if you click the Edit Server icon:



If you navigate to the Source tab in the Designer, you won't see any important details about the server because they aren't stored as part of the extension:

```
"oicBackend": {
"description": "Backend representing OIC",
"servers": [
]
}
```

Local servers are stored within a given VB Studio environment. So, if you switch your workspace to a new environment, you may have to define the local server again for the new environment.

When a backend has a local server, you must also supply a *runtime server configuration* on the Oracle Cloud Application instance to which you plan to deploy or preview the extension. You can do this on the **Manage Backends in Visual Builder** screen, available from the **Setup and**

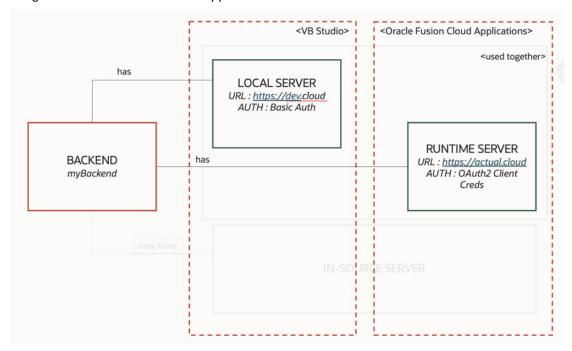


Maintenance menu on the relevant Oracle Cloud Applications instance. This completes the entire definition for the server: you have the URL and authentication method for the server while it's being used in development (as set on the VB Studio Create/Edit Server screen), and a different URL and authentication method when the extension is deployed or previewed (as set on Manage Backends in Visual Builder).

Note:

After you enter your credentials on **Manage Backends in Visual Builder**, it can take up to five minutes for them to take effect. Keep this in mind if you try to preview or deploy your extension before this time period has elapsed.

This diagram shows how a local server in VB Studio works in tandem with the runtime server configuration on an Oracle Cloud Applications:



In this image, myBackend has a local server. Within VB Studio, then, myBackend resolves to the URL https://devcloud with Basic Auth credentials. But when the App UI is served from Oracle Cloud Applications, either through the Preview action or after it is deployed, myBackend will resolve to https://actual cloud and OAuth 2.0 Client credentials.

Note:

You can't use runtime server configurations in government installations, although this support is being introduced gradually beginning in 2025.

If the runtime server configuration is missing for a backend, you'll see a warning message on the backend's **Overview** tab. Assuming you have the necessary privileges, you can click **Edit Configuration** to supply the runtime details on the Oracle Cloud Applications instance. If you don't have the right privileges, you'll have to ask an administrator to do this for you. This



process must be completed for **each** Oracle Cloud Application instance to which you plan to deploy. See Provide a Backend's Runtime Server Details.

A local server is a good option if you need to:

- Change the URL and authentication method for a backend for different environments (VB Studio, an Oracle Cloud Application test or prod instance, and so on).
- Use mock REST APIs with a different set of credentials only during the development cycle, switching to different set during deployment.
- Override an in-source server during development. While less common than the first two scenarios, this situation can arise occasionally.

To see an example of using a local server and a runtime server to access third-party data (in this case, Oracle Process Automation), see Connect to OCI Process Automation Services.

If you change the details of a local server or runtime configuration after deploying your extension, your changes take effect immediately; redeploying is not required. (Changes to insource servers, on the other hand, require you to redeploy the extension in order to implement the update.)

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 App UI 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* (relevant to local servers only) 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:

- Manage the credentials for accessing the service (if credentials are required).
- Manage identity propagation of the end user logged into the App UI (if the service supports the standard IDCS OAuth flows).
- Manage how the extension connects to a service to obtain code grant type token.
- Manage how your application connects to the service (via proxy or via Direct call).

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 App UI 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 three types of authentication methods that can be used for a backend: identity propagation, code grant type, and fixed credentials. (Fixed credentials are never available for in-source servers.)

To provide flexibility in how service connections are established, you can use different authentication methods for your extension's development cycle, as well as for each Oracle Cloud Application instance to which your extension is deployed. If needed, you can use methods of different types between development phases.



Note:

Before you connect to a service, an administrator may need to configure the VB Studio environment settings for backend services, or the settings for the external service or identity provider.

For certain authentication types, your REST APIs might need CORS support for both VB Studio and Oracle Cloud Applications domains (as reflected in the tables shown in the next few topics.) If you don't have this setting in place, you'll see a CORS Preflight error when the REST API is called. The backends that are inherited from the Unified Application must be pre-configured for CORS to allow access to VB Studio.

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 VB Studio and extract the user (or subject) from it. VB Studio 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 <heathermal / header.body.signature > and looks like this:

eyJhbGciOiJIUzINNiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY30DkwIiwibmFtZSI6IkpvaG4
gRG91IiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJ
SMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c

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.

Identity propagation authentication methods are available for both *local* and *in-source* servers. Use local servers if you need to vary the URL or scope between different instances, like Test and Production. You can see an example of this in Connect to OCI Process Automation Services.

Here are the authentication methods that use identity propagation:

Authentication method	Description	Available for:
Oracle Cloud Account	This method is used in extensions to communicate with Oracle Cloud Applications. Since you will always have an out-of-the-box backend (available through the Unified Application) representing Oracle Cloud Applications and its child instances, you will probably not need to use this authentication type.	In-source servers Local servers



Authentication method	Description	Available for:
OAuth 2.0 User Assertion	Select this method to call an external system's REST API that can accept a token from the IDCS or OCI IAM identity domain attached to VB Studio. Such external services are represented in IDCS or OCI IAM as a resource application with a particular scope. You must be signed in as a user that is present in the relevant IDCS or OCI IAM identity domain.	In-source servers Local servers Runtime server configurations
	If you have an Oracle Process Automation or Oracle Integration instance in the same IDCS stripe/OCI IAM domain, you can use this authentication type. See Connect to OCI Process Automation Services for details.	
	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 <i>own</i> scope, rather than using the service's URL.	

How Does the Fixed Credentials Authentication Method Work?



To use local servers in conjunction with runtime server configurations, make sure your Oracle Cloud Applications instance is on 25A and that you've completed the check described in Work With Services (Limited Availability).

Authentication methods like Basic Auth are called *fixed credentials* because they ignore the signed-in user's identity and instead apply only the credentials that are configured (fixed) for the backend.

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	Valid for:
None	Select this for services that don't need authentication and don't accept Authorization headers.	In-source servers Local servers
		Runtime server configurations



Authentication method	Description	Valid for:	
Basic	Select this for services that require a fixed user name and password for authentication. The signed-in user's credentials aren't used for authentication. This option uses the VB Studio authentication proxy, irrespective of the connection type you choose. Due to the limitations of basic authentication, it's recommended that you use this method only during development. Here's why: Suppose you set basic authentication with a particular user name 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.	Local servers Runtime server configurations	
	While Basic authentication is an option for service connections in VB Studio Gen 2 in government realms, it is a relatively weak security layer. If your service connections use Basic authentication to connect to Oracle Cloud Application services, it's recommended that you use one of the other available authentication mechanisms (for example, OAuth 2.0 Resource Owner Password), if possible. Consult your organization's security guidelines if you still plan to use Basic authentication, especially in production applications.		
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.	Local servers Runtime server configurations	
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 taken LIBL fields.	Local servers Runtime server configurations	

for the Client ID, Client Secret and token URL fields.

Authentication method	Description Valid for:	
OCI Signature Authentication	This method uses a signature method to create an Application Local servers on ID flow using a single Oracle Cloud Infrastructure (OCI) user to connect to OCI endpoints. All requests go through a proxy	ly
	because of the requirement to sign the outgoing message.	

Note:

Although OCI Signature authentication appears as an option, it is actually not supported for extensions at the moment. If you configure OCI Signature authentication for a backend and use it in an App UI, the service connection call will fail with a 500 error when you try to preview or deploy the extension.

To use this authentication in VB Studio, 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 VB Studio as follows:

- Select Oracle Cloud Infrastructure API Signature 1.0 as the authentication method.
- 2. Click Enter the API Key and Private Key ().
- Construct the API key in the following format, then copy and paste it as the API Key: tenancy-ocid/user-ocid/fingerprint
- Paste the PEM file contents as the Private Key. Copy the entire file, starting with ----BEGIN PUBLIC KEY----right up to ----END PUBLIC KEY----.
- 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 App UI's JavaScript or from the server proxy. You should make this decision based on the CORS support your REST API has. In extensions, this attribute is only available if you have a local server. If you choose an in-source server, your REST API should already have support for CORS.

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.

How Does the OAuth 2.0 Authorization Code Grant Authentication Method Work?

The OAuth 2.0 Authorization Code Grant authentication method provides a token to the user of your extension, which allows them to access a service. This is the most common OAuth authentication method and is used by many service providers, like Google and Dropbox.

Services that support the Authorization Code Grant method have an authorization endpoint, a token endpoint, a client ID and secret, and a scope. See Connect to Google Drive Using the OAuth 2.0 Authorization Code Grant for descriptions of these properties.

During authentication, the extension opens the authorization endpoint in a new tab or window, which prompts the user to authenticate and, optionally, provide consent for the app to connect to the service. After the user has done this, an auth code is created, which is passed on to the token endpoint to get an access token.

In this way, the Authorization Code Grant method differs from the identity propagation and fixed credential authentication methods. With identity propagation, the user's identity is converted into a token and passed to the REST API. With fixed credential authentication, the identity that the REST API sees is the one that is configured for the REST API regardless of who logs in. Unlike these methods, the identity of the user logging into the system using the Authorization Code Grant method might be different than the identity of the user being passed to the REST API.

The Authorization Code Grant method is available for local servers and runtime server configurations.

Refer to Authorization Code Grant in RFC #6749: The OAuth 2.0 Authorization Framework.

There are currently some known issues for this authentication method:

- This authentication method currently does not support storing refresh tokens.
- Proof Key for Code Exchange (PKCE) is not supported in this release. PKCE is a variation
 of the OAuth Authorization Code Grant flow that doesn't require a client secret. Refer to
 Proof Key for Code Exchange by OAuth Public Clients.



• Service connections configured with this authentication method cannot be tested using the Test tab of the Create Service Connection wizard, because the Test tab cannot open the Google login flow in a new tab or child window. Instead, test your service connection from the Request and Response tabs by providing request and response examples as needed. See step 2 in Connect to Google Drive Using the OAuth 2.0 Authorization Code Grant for a sample response for the Get method.

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.

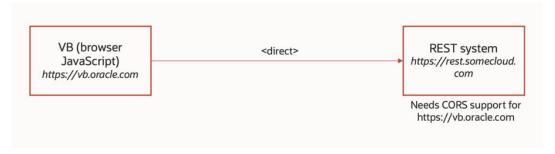
To allow such calls, the server hosting the APIs has to allow the domain of the web page to call by enabling CORS for the web page domain. The rules for CORS are built within the browser and are used to secure the user from arbitrary hosted scripts calling any APIs. See this blog for more details. If 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.

Decide Between Direct Calls and Using a Proxy

An extension can call external REST services either directly or through a proxy. Which method you use depends in part on which authentication method you've chosen.

Let's take a look a the two primary connection types:

Direct calls: When using direct calls, the REST API 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, as illustrated here:

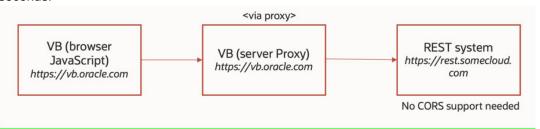


When using direct calls, the browser determines when to timeout the call to the REST API, if the REST API doesn't respond.

• Proxy: If you're using a local server, you can use the proxy flow by setting the Connection Type attribute to Always Use Proxy or Dynamic - Service doesn't support CORS. The proxy is a VB Studio trusted server-side component that calls external REST services hosted on external domains on behalf of your App UI. One benefit of this method is that there is no 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 proxy. Some authentication types, like Basic and OCI Signature Authentication, automatically use the proxy to avoid security issues, regardless of what's specified on the Connection Type attribute. The proxy times out if the REST API doesn't respond within 234



seconds.



When you are previewing or running a deployed App UI that calls an external REST API, you may need to add the Oracle Cloud Applications domain to the CORS allow list of the external REST API, depending on which backend configuration you have chosen.

This table describes what kind of CORS support you need for your external REST API, depending on which type of server/configuration you're using:

	In-source Server	Local Server	Oracle Cloud Application Runtime Configuration
Where is it applicable?	If a backend has an insource server, the server details are used for any call to the backend from VB Studio (where the extension is being developed) or from the Oracle Cloud Applications instance (where the extension is deployed).	If the backend has a local server, the server details are used for any call to the backend from VB Studio only (where the extension is being developed).	If the backend has a runtime configuration, the server details are used for any call to the backend from the Oracle Cloud Applications instance (where the extension is deployed).
What kind of CORS support does your REST API need?	There is no Connection Type attribute available to circumvent CORS. The external REST system must provide CORS support irrespective of the authentication type used.	If the external REST system doesn't have CORS support, you can change the Connection Type attribute to "Always use proxy" or "Dynamic, Service doesn't support CORS". Authentication types— Basic and OCI Signature—always use the proxy and are therefore not subject to CORS.	There is no Connection Type attribute available to circumvent CORS for individual backends. However, authentication types—Basic, OAuth 2.0 Client Credentials, and OAuth 2.0 Resource Owner Password— always use the proxy and therefore are not subject to CORS.

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 issuing system, Oracle Cloud Applications (service) infrastructure ensures that the token is reissued when the current one expires, provided the user is still logged into Oracle Cloud Applications and the calling REST API complies with the OAuth specification. If the user is no longer logged into Oracle Cloud Applications, any attempts to issue a new token will redirect the user to the login page.



Create a Backend

By default, your extension has access to Oracle Cloud Applications and its child backends. You can't override or modify these, but you can use them to create service connections based on these backends. You can also create custom backends in your extension.

To create a service connection to your backend, click + Service Connection. See Create a Service Connection.

Add Server Variables for Backends

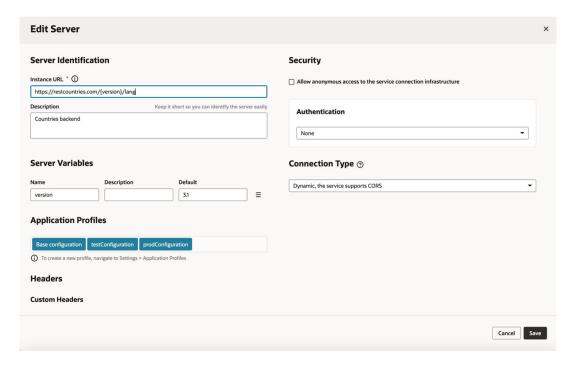
When creating or editing a backend, you can add server variables to the instance URL on the Servers tab for an in-source server. Server variables are not available for local servers.

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:

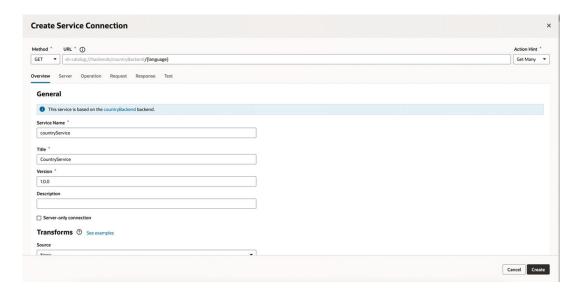
1. The server instance URL can be represented as https://restcountries.com/
{version}/lang.

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**:

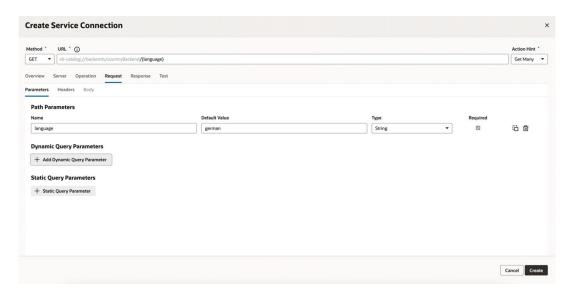


2. The endpoint is /{language}.
We define the service connection on the countryBackend by defining the endpoint and adding /{language} to the URL path.





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 \bigcirc or \bigcirc 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** \square 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

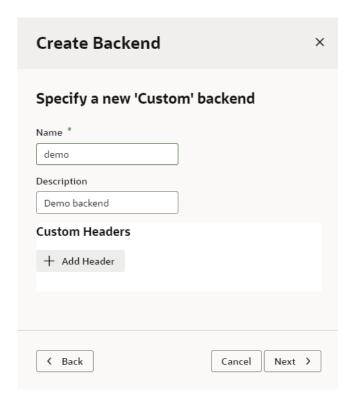
You can create your own backend to map to a custom server other than the built-in Oracle Cloud Applications backend.

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.





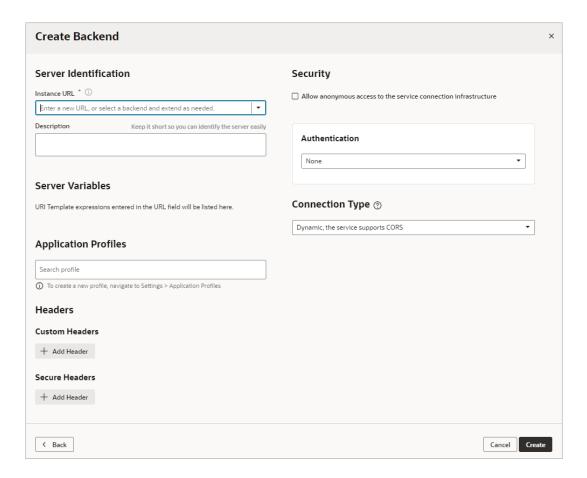
Optionally, click + Add Header under Custom Headers to add a static header that is
passed from the browser to the service. Enter a name for the header and its value, then
click OK.

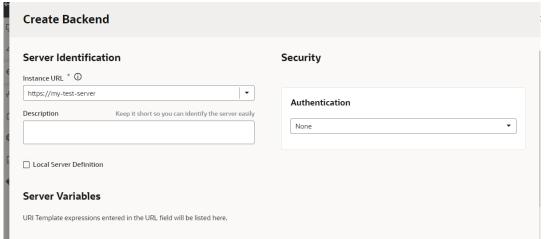
Headers can exist at both the backend level and at the server level. The difference is that backend headers are guaranteed to be applied to REST API calls, whereas server headers are applied to the call only if that particular server is in play. For this reason, using headers at the backend level is recommended.

Custom headers become available to you from the browser's Developer Tools console.

6. Click **Next** to display the second page in the wizard:







- Determine whether this backend needs a local server (see What Are Local Servers?) or insource server (see What Are In-Source Servers?). If you need a local server, click the Local Server Definition checkbox. (Most extensions will need a local server.)
- 8. Enter the instance URL and other information that your extension 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.

- 9. If you need more information about the options in the Authentication drop-down, see Set the Backend's Authentication Method and Connection Type.
- If you're using an in-source server and need help in determining server variables, see Add Server Variables for Backends.

11. 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.

If this backend was created with a local server, you'll need to provide the corresponding runtime server details in Oracle Cloud Applications so that the backend will work for the Preview action. See Provide a Backend's Runtime Server Details.

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.

Provide a Backend's Runtime Server Details

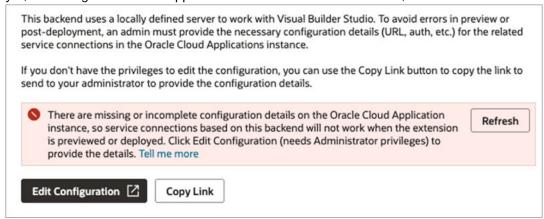
Each VB Studio backend that has a local server, but no in-source server, needs a corresponding runtime server configuration defined in Oracle Cloud Applications. This server will be used for both the Preview action, as well as when the extension is deployed to production.

Note:

- To use this feature, your version of Oracle Cloud Applications must be 24D or later.
- Runtime server configurations are not yet available in Oracle Government Cloud instances.



If a backend's runtime server hasn't been set in the Oracle Cloud Application's environment yet, a warning to this effect appears on the backend's Overview tab, like this:



To provide the backend's runtime server definition in Oracle Cloud Applications:

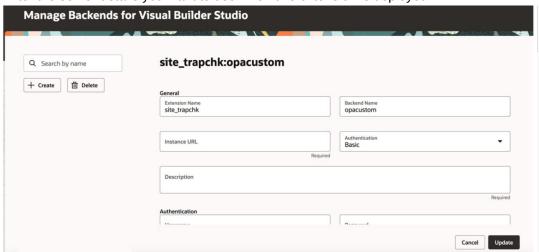
 Click Edit Configuration to go to the "Manage Backends for Visual Builder Studio" administrative page.

If you don't have the required administrative privilege on this instance, use the **Copy Link** button to copy the URL and send it to an administrator with a request to supply the necessary details. If you'd like the privilege to edit the configuration yourself, contact your security administrator and request the Create and Edit Backends for Visual Builder Studio (ORA_FND_TRAP_PRIV) privilege.



In 24D, users who have the <code>ORA_FND_TRAP_PRIV</code> privilege can view the "Manage Backends for Visual Builder Studio" page, but not add new entries to it. To edit this page, users must have either the <code>ORA_FND_ADMINISTRATOR_JOB</code> or <code>ORA_FND_APPLICATION_DEVELOPER_JOB</code> Oracle Cloud Application role.

2. Enter the server details you want to use when this extension is deployed:



This process must be completed for each Oracle Cloud Application instance to which you plan to deploy.



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 provided by the Unified Application.

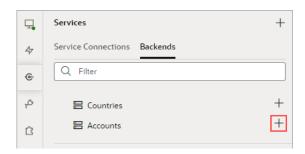
A child backend inherits the parent backend's definition, which you can override as required. Its server URL is derived from the top-level backend, with vb-catalog://backends/ as the base URL. For example, the Sales and Service backend, which is a child of the Oracle Cloud Applications backend, has the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



- 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.
- Enter a name and description for the child backend. Optionally, click + Add Header to add custom static headers.
- 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 (supplied by the Unified Application) is slightly different, as the parent and the child backends are in different extensions.

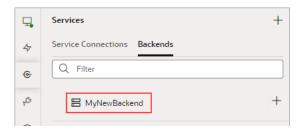
To create a child backend for the Oracle Cloud Applications backend, which is inherited from the Unified Application dependency:

1. Open the Backends tab from the Services tab in the Navigator.



- 2. Click the + sign.
- 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.
- Enter a name and description for the child backend. Optionally, to add custom static headers, click + Add Header.
- Click Next.
- 6. Click the URL text box to see the URLs for all the backends you can use, both in the current extension and those available through dependencies.
- 7. Select the Oracle Cloud Application backend, then 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.
- 8. Enter other settings, as needed.
- Click Create.

Notice that the child backend is not shown under the Unified Application, but rather where backends created for your extension appear:



Edit a Backend

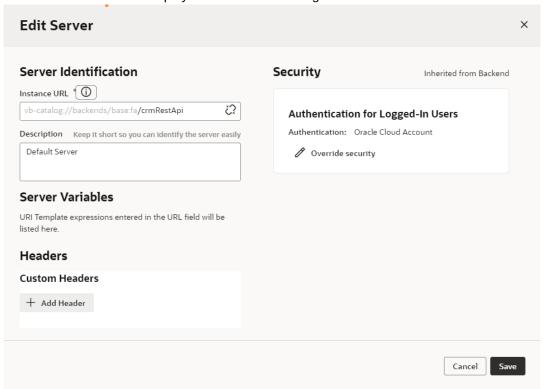
To edit a backend:

- Open the Backends tab from the Services tab in the Navigator.
- 2. Select the backend you want to modify to display the **Overview** tab.
- Change the description, if desired, or change the transform attached to the backend (see Add Transforms for more information.)
- On the Headers tab, use + Add Header to add static headers that are passed from the browser to the service.
- On the Servers tab, select an in-source or local server to modify. Note that local servers are flagged with an extra "Local server definition" tag.





6. Click the **Edit** icon of to display the Edit Server dialog:



Modify the backend server details. These options are the same as those you configure when you define a backend. Depending on whether it's a local or in-source server, you'll see slightly different fields:

- In Server Identification, under Instance URL, change the modifiable portion (the part
 that isn't grayed out) of the instance URL or the description. If this is an in-source
 server, you can also change the server variables (described in Add Server Variables
 for Backends).
- In Headers, under Custom Headers, use + Add Header to add static headers that are
 passed from the browser to the service (although it's preferable to use the backend's
 headers rather than those on this tab—see step 4 in this procedure).
- In Security, you can change the authentication type used to connect to the backend. If this is a local server, you can also change the connection type (see Set the Backend's Authentication Method and Connection Type).
- 7. Click **Save** when you're done making changes.
- 8. Click the **Source** tab to view the contents of the services/self/catalog.json at the extension level. Edit the file if needed.

You can also access this file under services in the Navigator's Source View tab.

Remember, any changes you make here for a local server will **not** be reflected in the server's source code.

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

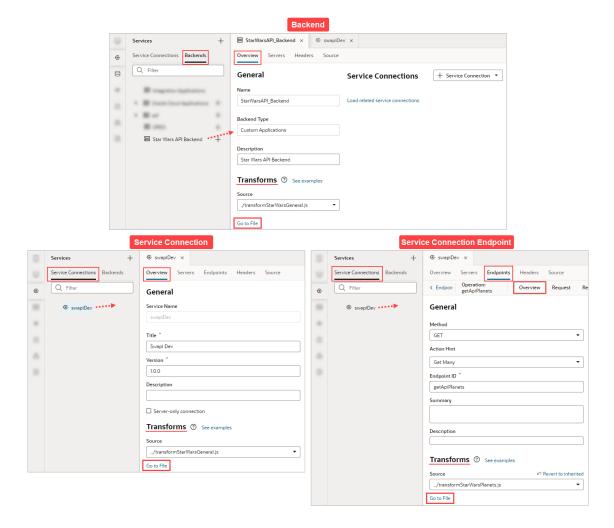
Service Level Transforms

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**.



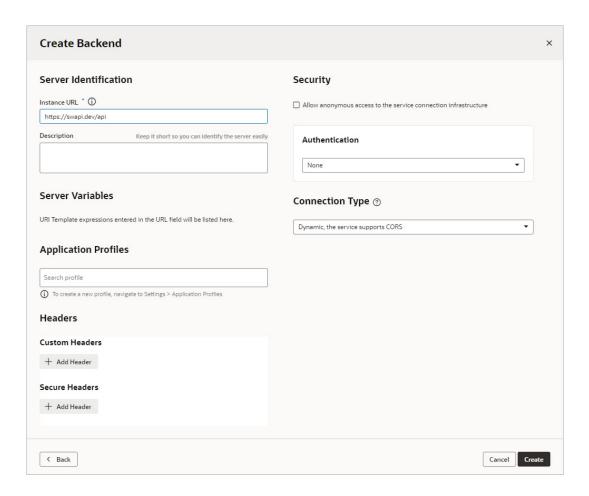


Example of Creating Service Level Transforms

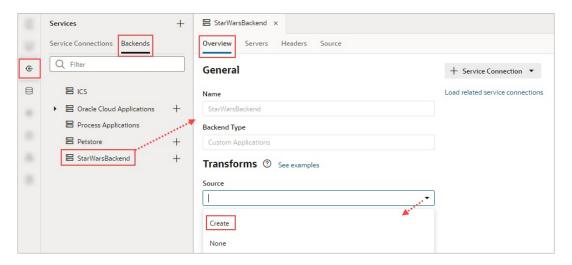
These steps show how to create a backend for service connections and how to create the file with the template transform functions for your new backend and its service connection endpoints:

- 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.



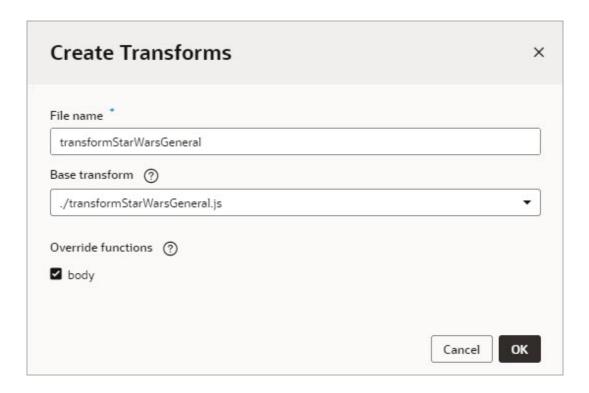


5. You can now create the file with the template transform functions for your new backend. On the Overview tab, in the Transforms section, click the Source field's down-arrow and select Create:

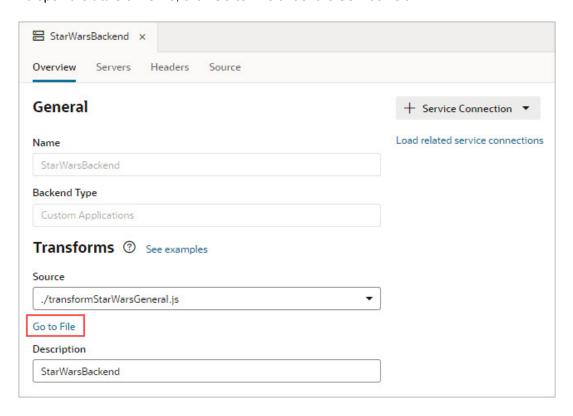


6. Provide the file name for the transforms file:





- 7. For the **Base transform** drop-down list, 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.

```
JS TransformStarWarsGeneral.j ×
          class Response {
236
               * @typedef {Object} PaginateResponse
237
               * \mbox{\it Mproperty} (number) totalSize optional what the totalSize of the result is (the total count of the records in
238
239
               * the service endpoint).
240
               * @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
              * @property {String} pagingState optional. This can be used to store any paging state specific to the paging
243
244
              * capability supported by the endpoint. This property can be used in the response paginate transform function
245
               * to set an additional paging state. This will then be passed as is to the request paginate transform function
               * for the next fetch call.
246
247
248
249
              * paginate is called with the response so this function can process it and return an object with
250
              * properties set.
251
252
               * @param {object} result
              * @param {TransformsContext} transformscontext
253
              * @return {PaginateResponse}
255
256
              /*paginate(result, transformscontext) {
257
258
                  const tr = {};
259
                  if (result && result.body) {
260
261
                      const cb = result.body:
                      if (cb.totalCount) {
262
                        tr.totalSize = cb.totalCount;
263
                      if (cb.totalCount >= 0) {
266
                          tr.hasMore = !!cb.hasMore;
                      } else {
267
268
                        tr.hasMore = false;
269
279
271
                  return tr:
272
273
274
               st body is called last, after all the other response transforms have been called. It is a hook for authors
275
276
              * to transform the response body or build an entirely new one.
277
               * @param {object} result
               * @param {TransformsContext} transformscontext
278
279
               * @return {object}
280
                                                      transform function for
281
              body(result) {
                                                      body of response
                  // TODO: Replace example code
282
                  let tr = {};
283
284
                  if (result.body) {
285
                    tr = result.body;
286
287
288
                  // as a example store some random aggregation data
                  tr.aggregation = { example: 4 };
290
291
                  return tr:
292
```

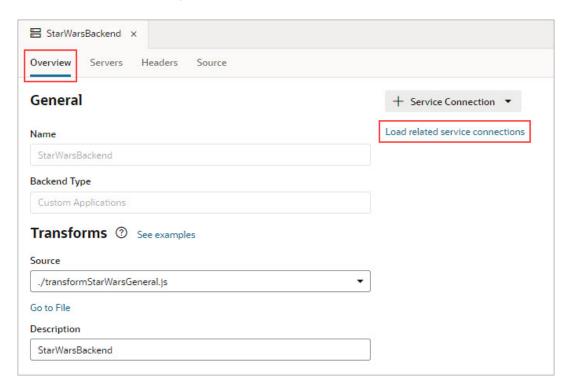
Here's an example for the body(result) transform function, which adds text to all of the mass and height fields:

```
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;
}</pre>
```

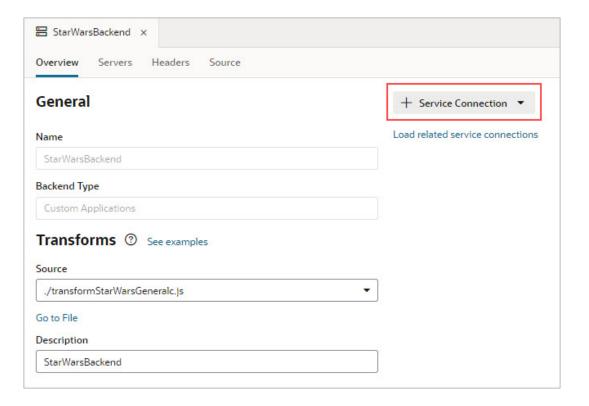
The created transforms file is inherited by its related service connections, which are created by using the **+ Service Connection** button on the **Overview** tab. To view the related service connections, click the **Load related service connections** link:



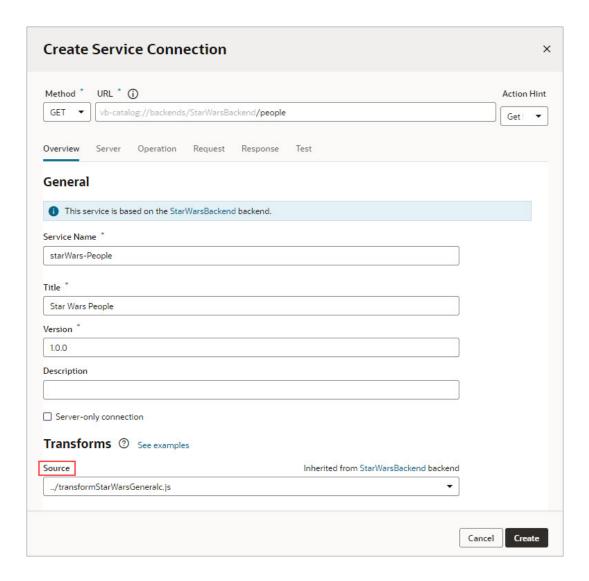
For details about creating a service connection from an endpoint, including how to get an example of the body of the response, see Create a Service Connection from an Endpoint.

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=${options.offset}`;
 c.url = newUrl;
 return c;
};
sort
 * Sort Transform Function Implementation
 * @param configuration
 ^{\star} @param options the JSON payload that defines the sortCriteria
 * @param context an object to store/retrieve any contextual information
 * current request lifecycle.
 * @returns {object} configuration object. the url looks like ?
orderBy=foo:asc
 * /
PageModule.prototype.sort = function (configuration, options, context) {
 const c = configuration;
  if (options && Array.isArray(options) && options.length > 0) {
    const firstItem = options[0];
    if (firstItem.name) {
     const dir = firstItem.direction === 'descending' ? 'desc' : 'asc'
     let newUrl = c.url;
     newUrl = `${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 App UI, you create connections to the services that provide access to these API endpoints.

What Are Service Connections?

A *service connection* provides basic information about a service, including connection details, properties, and the REST endpoints provided by the service.

You can create a service connection by:

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

Each of these mechanisms create a representation (metadata) of the external REST service, which is required for you to use the REST endpoints as data sources in your extension.

Let's take a look at how you might use service connections in your extension:

- Let's say you're configuring an Oracle-provided App UI, which has a dynamic container, and you want some of the content in the container to be sourced from a new custom object you defined in Oracle Cloud Applications. You create a service connection in your current extension, choose the Catalog option, select the relevant Oracle Cloud Application pillar, then select the custom object REST endpoint you want to use. Finally, after you've created and tested the service connection, you use it to source the content in your dynamic container.
- Suppose you're creating a brand new App UI which needs to show population data from a
 publicly-available countries API. In the extension containing your new App UI, you'd create
 a backend (with no authentication), as well as a service connection to the countries API.
 You'd then configure the UI elements on the App UI's page to show data from this service
 connection.
- Suppose you want to display data from a third-party system, like LinkedIn, in your App UI. To achieve this, you'd first create a backend to represent LinkedIn, using a *local server* configuration that uses the OAuth 2.0 Client Credentials for the authentication method. Next, you'd create a service connection to the LinkedIn REST API, using the "Define an Endpoint" option. Finally, for each Oracle Cloud Application instance to which you plan to deploy your extension, you'd need to configure the **Manage Backend for Visual Builder Studio** page, as described in Connect to OCI Process Automation Services.
- Suppose you want to create a companion application (that is, an App UI) for your
 HelpDesk app, which relies on a custom object you created in Application Composer. You
 could simply grab the URL for the REST Describe in Application Composer, then create a
 new service connection using the **Define by Specification** option in VB Studio. See
 Display SaaS Data In Your App UI for more details.



 Instead of creating a new service connection, you might add an extension containing the service connection you want to use as a *dependency*, assuming that the "accessible" flag has been set. You could also use any of the service connections or backends defined in the Unified Application.

When you define a service connection, you supply metadata for it that can include request and response payload structures in JSON, query and path parameters, headers, and so on. This metadata helps to shape the raw data that you get back from the REST call, to narrow the scope of the data that is returned.

When you create a new service connection, you must also create a new *backend*, unless you use an existing backend as part of the service connection. Backends provide a way to bundle common attributes across service connections into a single entity you can refer to later. For example, you could create a backend (perhaps called "myBackend") with the URL and the authentication information, then create two service connections based on "myBackend" for / employee and /department resources.

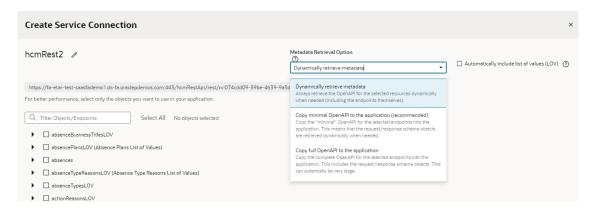
Service Connections: Static Versus Dynamic

Service connections in VB Studio 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 VB Studio.

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 App UI, because the service metadata is fetched every time the App UI 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 <code>openapi3.json</code> 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.
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) Available only for ADF Describe-based services like Oracle Cloud Applications	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

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 Manage Service Endpoints.

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, VB Studio 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:

```
de
Sales and Service vb-catalog://backends/base:crmRest
                          .oraclecorp.com:443/crmRestApi/rest/rv:8053295d-5425-47da-
https://
9d2c-3d59d603c2fe/en
Applications Core vb-catalog://backends/base:fndRestApi
                          .oraclecorp.com:443/fscmRestApi/applcoreApi
Enterprise Resource Planning and Supply Chain vb-catalog://backends/base:fscmRest
                          oraclecorp.com:443/fscmRestApi/rest/rv:8053295d-5425-47da-
https://
9d2c-3d59d603c2fe/en
Default Server vb-catalog://backends/Extension:hcmIndexSearch
                          .oraclecorp.com:443/hcmRestApi/indexSearch
Human Capital Management vb-catalog://backends/base:hcmRest
https://
                          oraclecorp.com:443/hcmRestApi/rest/rv:8053295d-5425-47da-
```

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.

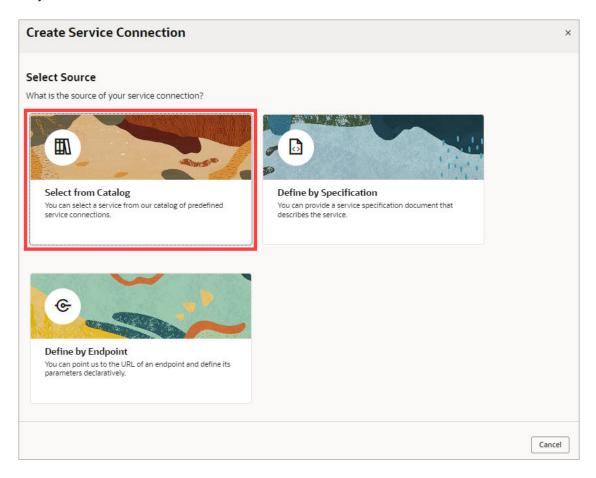




All URLs in backends and service connections should be HTTPS with a valid certificate.

Create a Service Connection from a Catalog

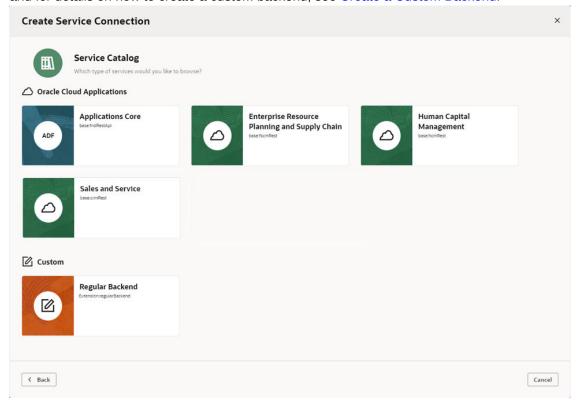
A catalog provides a list of services and their endpoints from an Oracle cloud service backend or a custom backend. Using a catalog to create a service connection saves you time, as common configuration tasks like authentication, connection type, and headers are predefined for you.



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:

If you don't see the Unified Application as a dependency, contact your administrator to confirm that you have the proper credentials and that your user role is authorized.

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.

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.

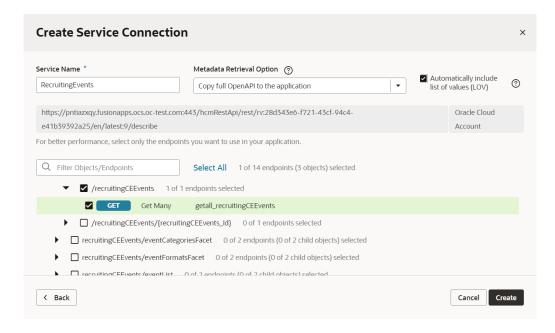
To create service connections from one of these catalogs:

- 1. Click **Services** in the Navigator.
- 2. In the Services pane, click the + sign and select Service Connection.
- 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 dropdown list, then select the endpoints that you require for your application. This way, you create a static service connection:

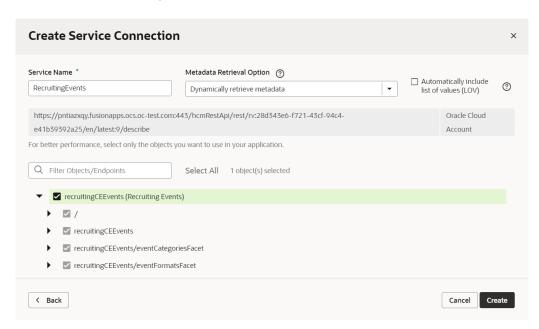


0

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.

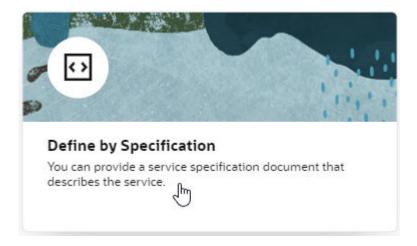
Create a Service Connection from a Service Specification

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.

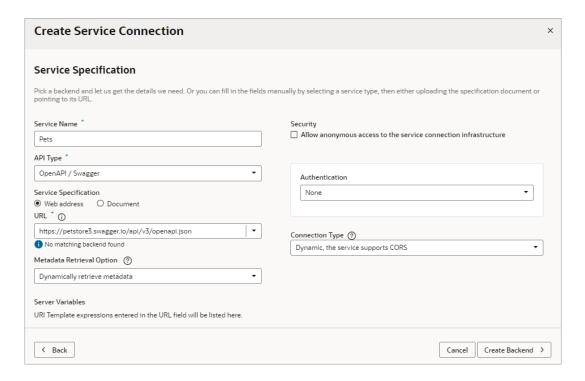
To create a connection from a service specification:

- 1. Click **Services** in the Navigator.
- 2. In the Services pane, click the + sign and select Service Connection.
- 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.

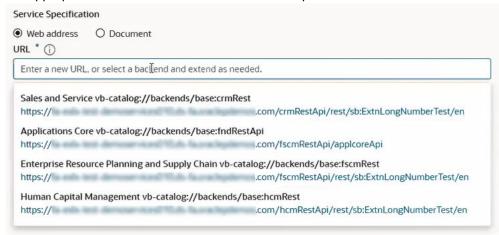


The name you specify will be the connection's display name in your application.

- Click the API Type dropdown and select the appropriate 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 (cache-enabled)** if you need to create a service connection to an Oracle Cloud Applications REST API that has a Describe URL.
 - Select ADF Describe for any other REST API that has an ADF Describe file.
- **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.



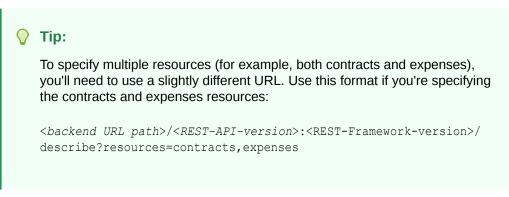
Then type in the rest of the Describe URL to the resource using this format:

<backend URL path>/<REST-API-version>:<REST-Framework-version>/
describe.openapi/<resource-name>

For example, for the HCM absences resource, you'd add /11.13.18.05:9/ describe.openapi/absences to the HCM URL you selected in the text box, where 11.13.18.05 is the REST API version and 9 is the REST Framework 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.



 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.

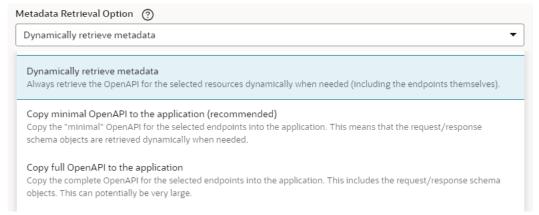


• If you have a document that contains the service specification, select **Document**, drag and drop the file that describes the service from your local system to the dialog or use the file browser to locate and select it on your system.

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.

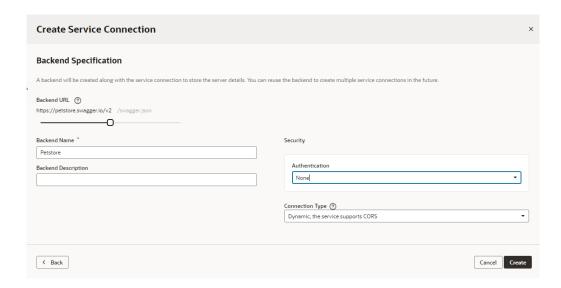


If you entered a web address, you can select **Dynamically retrieve metadata** to create a dynamic service connection that always pulls in the most up-to-date service definition for your application or you can select **Copy full OpenAPI to the application** to copy the complete service metadata during development and statically store it in your visual application's sources.

If your API type is ADF Describe or ADF Describe (cache-enabled), 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. If you are creating a backend that uses credentials (like Basic Authentication), or one that has different URLs for your dev/test/production instances, check the Local Server Definition checkbox. If your backend doesn't need authentication, leave Local Server Definition unchecked. You can see an example of how to work with different authentication methods here: Connect to OCI Process Automation Services.
 - d. In **Backend URL**, use the slider to specify which part of the URL you want to use for the backend.

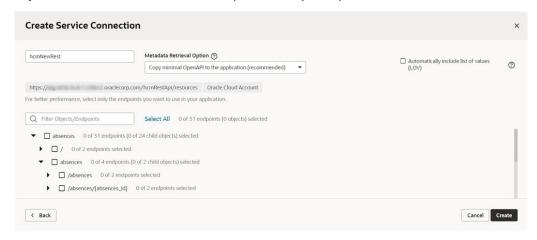




- Add a name and optional description for the backend.
- f. Update the **Security** and **Connection Type** settings for the backend as needed.
- Do one of the following:
 - If you have an ADF Describe or ADF Describe (cache-enabled) API type, continue to the next step.
 - If you have an OpenAPI/Swagger API type, click Create. The procedure is complete.
- 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.



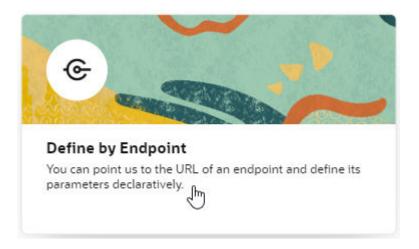
- Click Send Request.
- 4. Check the Send Request status. A status of 200 indicates success.

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**.
- 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.



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



Create Service Connection

Method

URL * ①

Action Hint *

Get Many

No matching backend found

Start by picking a backend and let us get the details we need.

Or by choosing the HTTP method and giving us the URL for your endpoint. It can optionally include query parameters.

To add path parameters include them in curly braces as shown in the example.

We can help you build apps faster if you select a basic action type for this endpoint (Get Many, Get One, Create, Delete, Update or Custom).

record, you would select POST as the method and Create as the Action Hint.

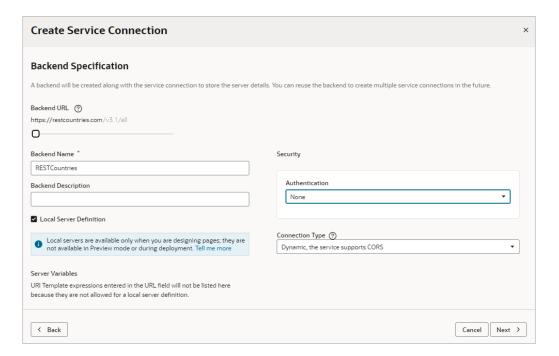
If an existing backend match is found, the URL will fill the field automatically; otherwise, you will need to create a new backend. See the following step for more details.

Cancel Create Backend >

When you're done, click Next.

< Back

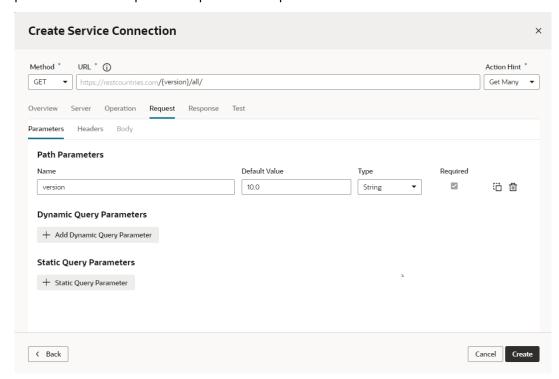
- If you did not have a backend to match your service specification URL, click Create Backend.
 - If you are creating a backend that uses credentials (like Basic Authentication), or one that has different URLs for your dev/test/production instances, check the Local Server Definition checkbox. If your backend doesn't need authentication, leave Local Server Definition unchecked. You can see an example of how to use different authentication methods here: Connect to OCI Process Automation Services.
 - b. In Backend URL, use the slider to specify which part of the URL you want to use for the backend.





- Add a name and optional description for the backend.
- d. (Optional) Change the **Security** and **Connection Type** settings for the backend.
- e. Click Next.
- In the Overview tab, add a Service name, and confirm the title and version. Optionally, enter a description.
- 7. 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:
 - 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.
- 8. Click the **Operation** tab to view the Endpoint ID that VB Studio will use to identify the REST API endpoint you specified at the start of this task.
- 9. 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.

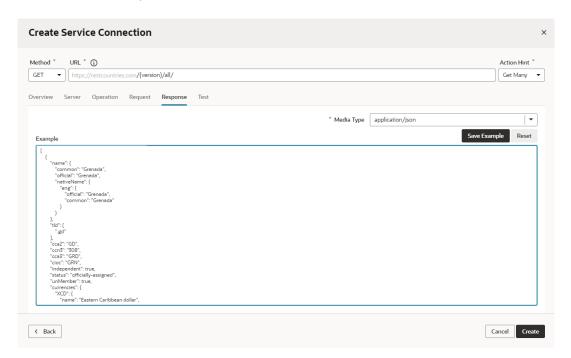


10. Click the **Response** tab and enter the response body for the endpoint.

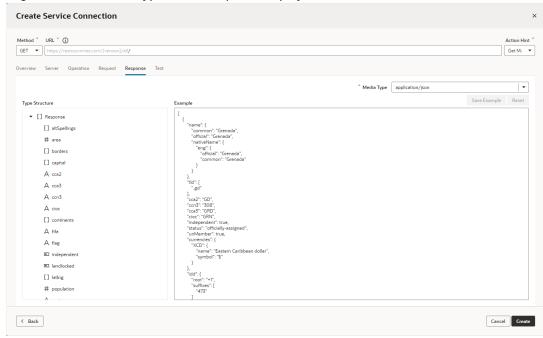
The **Response** tab displays the media type's OpenAPI3 metadata artifacts that can be represented: the example and the schema. You paste in an example of the body of the response into the text area and then click the **Save Example** button to commit your input



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.

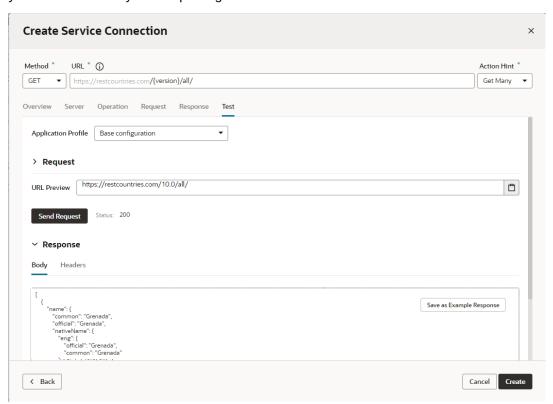


Note:

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.
- 11. 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.

12. 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

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.
	 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.

Manage Service Endpoints

You may want to add new endpoints to your static or dynamic service connections or modify the endpoints you already have.

See:

- Add Endpoints to a Static Service Connection
- Add Endpoints to a Dynamic Service Connection
- Edit a Static Service Connection's Endpoints

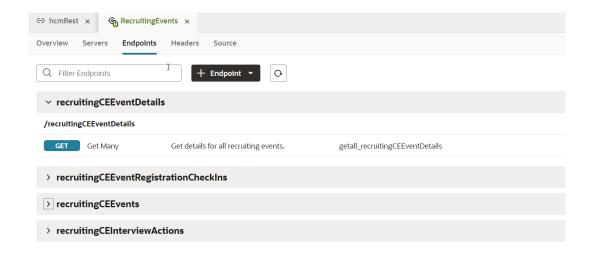
Add Endpoints to a Static Service Connection

After creating a static service connection, you can add endpoints from the same source or add custom endpoints.

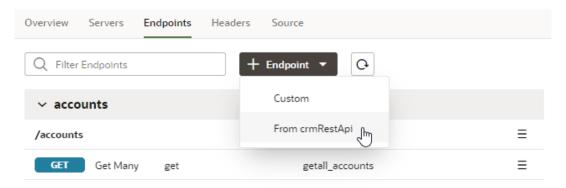
To add endpoints to a static service connection:

1. Open the static service connection's **Endpoints** tab:



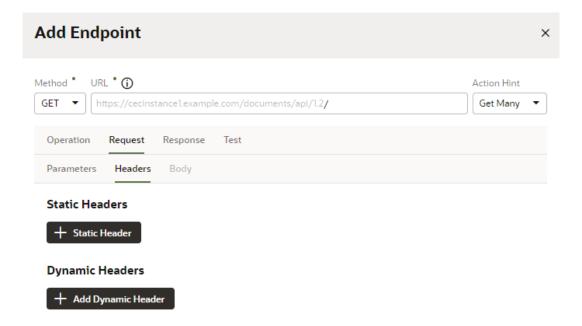


Click + Endpoint. If your service connection was created for a service from the catalog, select Custom or From original_service in the drop-down list:



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:



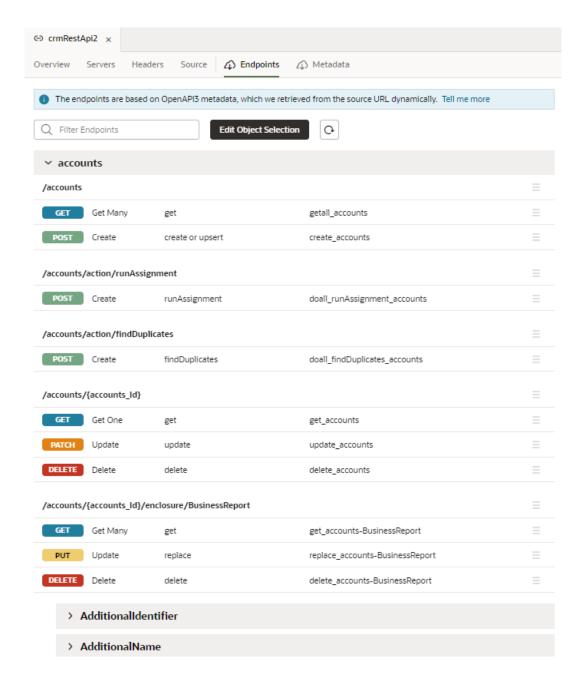
- 4. (Optional) If you chose to add an endpoint from the original service, you can change how your connection's service metadata is retrieved in the Metadata Retrieval Option dropdown. Select Dynamically retrieve metadata to convert your service connection to a dynamic one. For ADF Describe-based services, select Copy minimal OpenAPI to the application (recommended) to leverage both static and dynamic capabilities for your connection.
- 5. Click Save.

Add Endpoints to a Dynamic Service Connection

To add endpoints to a dynamic service connection:

1. Open the dynamic service connection's **Endpoints** tab:





- 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.
- Click Add.



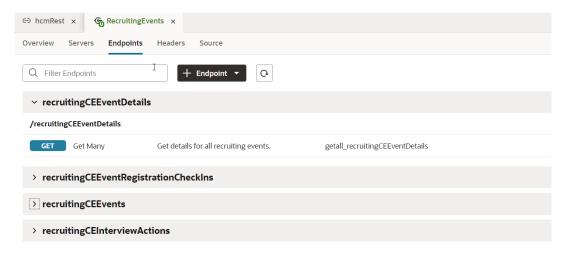
Edit a Static Service Connection's Endpoints

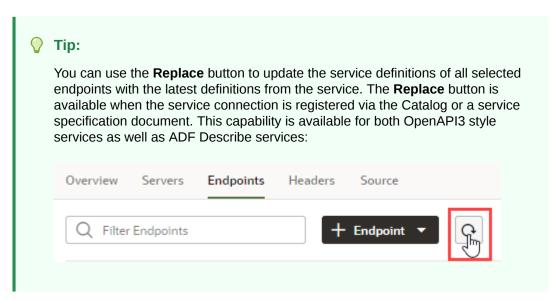
After you create a service connection and select the service endpoints, you can edit the endpoint settings, request parameters, and the response for each endpoint in the **Endpoints** tab. Endpoints can be edited only for static service connections (identified by the icon that appears in front of the service name).

If you edit an endpoint after you have created a type from it, you'll need to manually edit the type to use any of the changes to the endpoint. A type created from an endpoint isn't updated automatically when the endpoint is modified.

To edit a service endpoint:

Open the service connection's Endpoints tab.

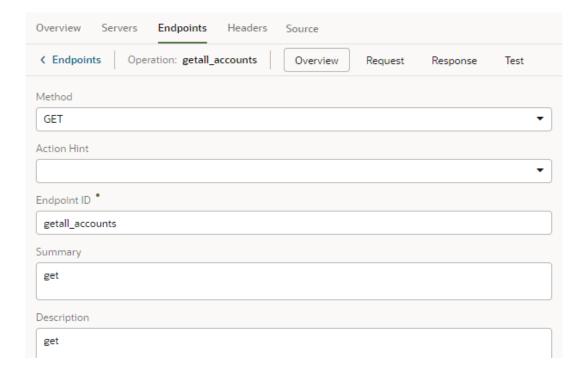




2. Click the endpoint you want to edit.

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, just 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.

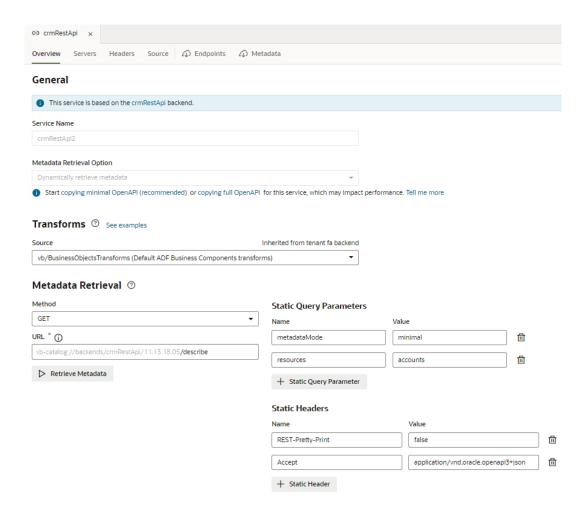
Retrieve Service Metadata for a Dynamic Service Connection

After creating a dynamic service connection, you can retrieve the service's most recent metadata, ensuring that your extension is using 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 for retrieving 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/base:crmRest/11.13.18.05:{REST-Framework-Version}/describe.openapi Server.

2. Click Retrieve Metadata.

The **Endpoints** tab displays a read-only view of the endpoints for the service connection's objects dynamically retrieved from the service's OpenAPI3 document corresponding to the source URL.

3. Click the **Metadata** tab.

The metadata file's contents, including endpoints and type definitions, are displayed.

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.



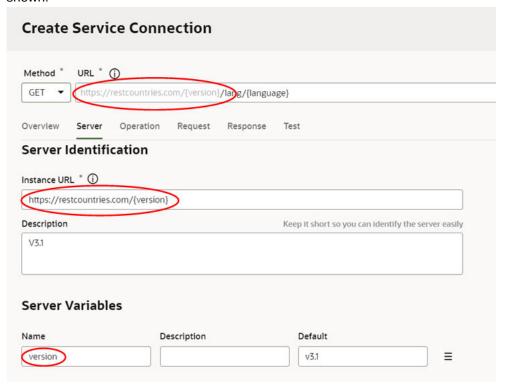
For service connections created after the 23.10 release, server variables must be updated from the backend associated with the service connection. For more information, see Add Server Variables for Backends.

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.
- 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 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.

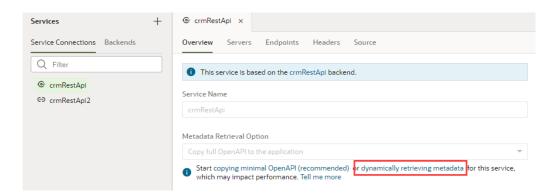
Typically, it's useful to create a dynamic service connection during an application's development stage, when endpoints defined in the service metadata are still evolving. A dynamic connection fetches the service's OpenAPI3 metadata from the source URL whenever the app is opened, enabling your application 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:



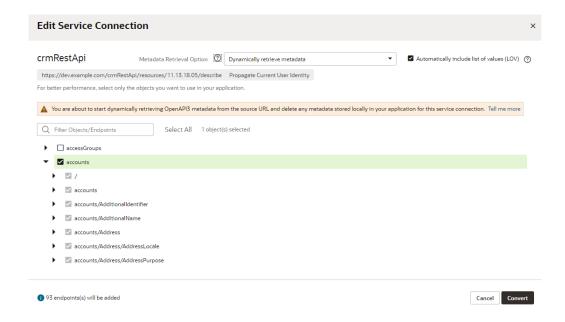
- To convert your service connection to dynamic:
 - Open the Overview tab of a static service connection, then click the dynamically retrieving metadata link.



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)**.

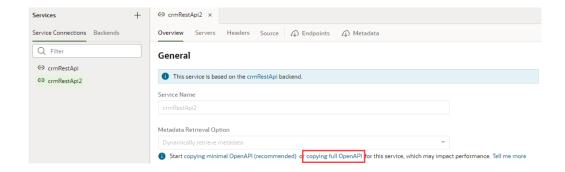




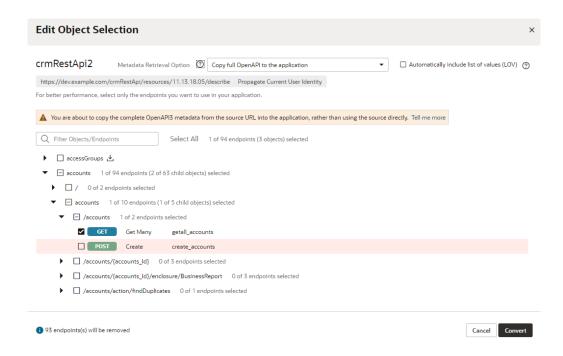
3. Click Convert.

The static service connection becomes dynamic.

- To convert your service connection to static:
 - 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.

Click Convert.

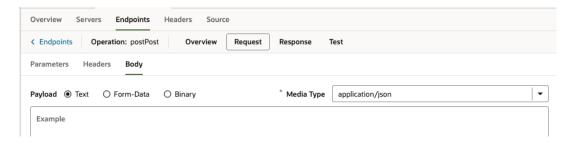
The dynamic service connection becomes static.

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.

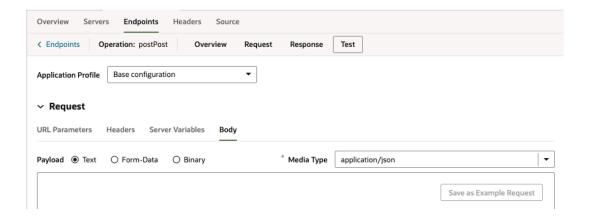
You can access this functionality from two places, where you can use radio buttons to select the payload type to test:

From the Body tab under the Endpoint editor's Request tab:



• From the Body panel under the **Test** tab:

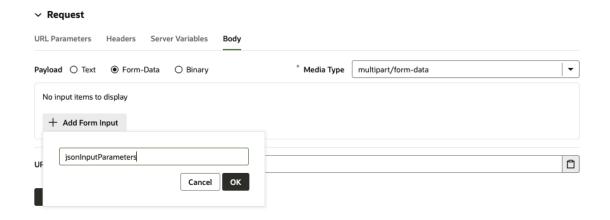




In both screenshots, the Text payload option is selected. The body input area beneath the radio buttons looks and operates the same way it has in all prior releases.

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:

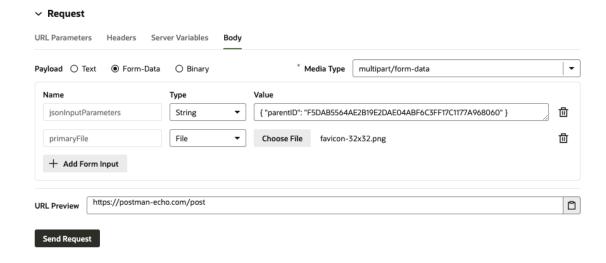


Two types of input elements can be created:

- String, for providing a user-entered text value.
- File, for selectiing a file to provide a binary data value.

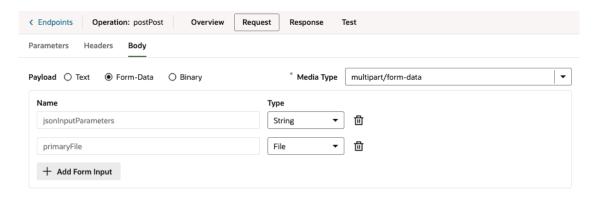
As elements are added to the form, corresponding properties will be added to the media type's schema in the OpenAPI metadata:





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

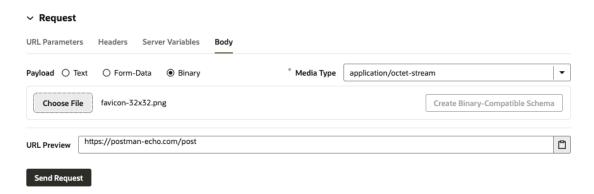
When you select the Binary payload option, the media type defaults to application/octetstream, although you could manually enter any binary content type you want:



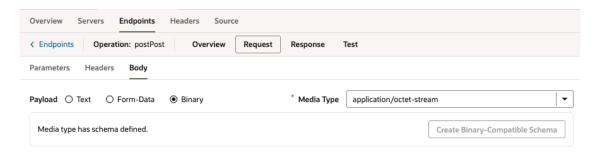


After you click the **Create Binary-Compatible Schema** button, the corresponding schema will be generated.

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:

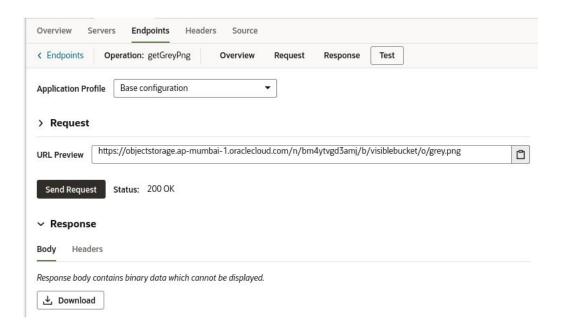


The file picker for the Binary payload type isn't shown in the Request tab:



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:

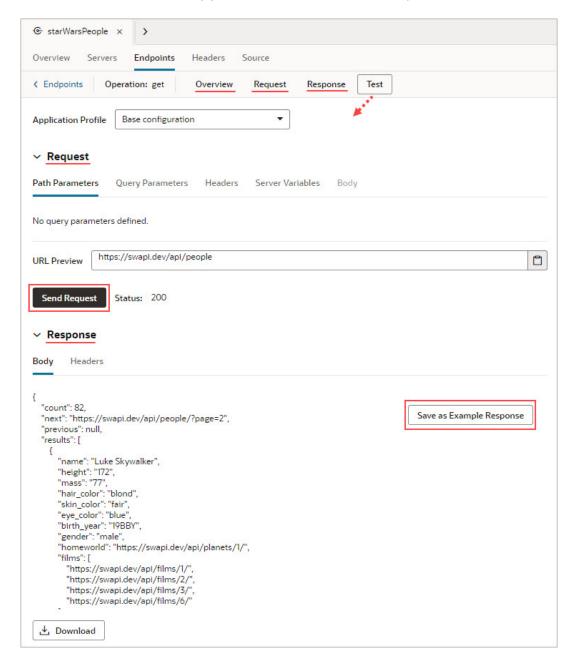


Update Schema of the Request or Response

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 Other Services

Provided here are topics on how to connect to services using various authentication methods.

Connect to OCI Process Automation Services

Here's an example of how to connect to Oracle Cloud Infrastructure Process Automation (also known as OCI Process Automation and Process Automation) services using two different authentication methods: identity propagation and fixed credentials (OAuth 2.0 Resource Owner Password). You'll also learn how to use different authentication for different phases of your extension's development, test, and deployment cycle.



This example applies to Oracle Cloud Applications 24D and later.

Example Scenario

In this example we'll look at two scenarios:

- Your Oracle Cloud Applications instance is configured with VB Studio, while Process Automation is provisioned in another IDCS/identity domain.
- Your Oracle Cloud Applications instance is configured with VB Studio, and Process Automation is provisioned in the same IDCS/identity domain.

You've created an extension that consumes a Process Automation REST API and are ready to begin testing. During the test phase, you want your extension to connect to the Process Automation Test instance that has a base URL of https://OPA-test-URL>, and a REST API service path of process/api/v1/tasks. When your extension is deployed to the Production instance, you want to connect to a different server; namely, the Process Automation Prod instance, which has a base URL of https://OPA-prod-URL>.

Prerequisites

The prerequisites differ a bit, depending on the authentication method you're planning to use:

For identity propagation, you'll need:	For OAuth 2.0 Resource Owner/Password, you'll need:
A Process Automation instance that is paired with a subscription to a Fusion-based Oracle Cloud Applications service. See this for more details.	You or your administrator need relevant Developer access to Process Automation
	Access to details from Process Automation's IDCS domain, such as client ID and secret, token URL, and scope. For this, Process Automation recommends creating a new confidential application with the Process Automation process scope. See here on how to set this up, and make sure the OAuth 2.0 Resource Owner grant type is available for this confidential application (others are optional for the purpose of this example).



Steps

Here are the steps for setting up this scenario:

- 1. Set up a local server as a backend for Process Automation
- Add service connections to the backend
- 3. Configure the backend for your Process Automation test instance
- 4. Preview and deploy to the test instance
- 5. Deploy to the production instance (or other instances)

Step 1: Set Up a Local Server as a Backend for Process Automation

In this step, you'll create a backend to represent the Process Automation instance, which is a one-time configuration for your extension. Since you're going to use one server for testing and a different one at runtime, you'll want to use a *local server* for this backend, which means it is used only during test and development and thus isn't stored in the source code.



For reusability purposes, it's a good idea to add all your frequently used backends to a common extension, deploy it, then use this extension as a dependency wherever you need it.

- In the Services panel, click the Backends tab, then click Add (+) to create a custom backend.
- 2. Name the backend opacustom, then click Next to provide the server details.
- 3. In the backend's Server tab, add these values:

	Indicates that these backend server details are not part of the extension itself, but will be used only during test and development.
URL	https:// <opa-test-url></opa-test-url>
	Either OAuth 2.0 User Assertion or OAuth 2.0 Resource Owner Password, depending on which type you want to use.
Scope	Corresponds to the Process Automation instance (typically https://
Token URL (for fixed credential authentication methods only)	Corresponds to the IDCS instance connected to Process Automation (typically https:// <idcs host="">/oauth2/v1/token)</idcs>

4. Click Create.

Step 2: Add a Service Connections to the Backend

Next, add the service connections that your extension requires to the backend you just created.

1. Select the custom backend's **Overview** tab, then click **+ Service Connection**.



- 2. Create the service connection by endpoint, selecting the **opacustom** backend. Add the rest of the endpoint's URL, /process/api/v1/tasks.
- 3. Navigate to the service connection's **Test** tab and test the service connection.
- 4. If the connection works, click **Create**.

Step 3: Configure the Backend for Your Test Instance

The Preview feature in the VB Studio Designer runs on your Oracle Cloud Application test pod. Since you want this instance to have access to the Process Automation test pod during the test phase, you need to provide these details explicitly to Oracle Cloud Applications; the local server definition itself is not used during Preview. To do this:

- Navigate to opacustom's Overview tab.
 You will see a warning there, which indicates that we haven't yet provided Oracle Cloud Applications with the information it needs.
- 2. Click Edit Configuration. If you don't have the required Administrative privileges, ask your administrator to complete the next steps. If you don't see this button at all, that means your instance has not yet been enabled to use this functionality and you won't be able to continue. Contact Oracle Support for more information.
- 3. In the Manage Backends on Visual Builder Studio screen, enter these details:

	For Identity Propagation	For Fixed Credentials
URL	https:// <opa-test-url></opa-test-url>	https:// <opa-test-url></opa-test-url>
Authenticati on	OAuth 2.0 User Assertion	OAuth 2.0 Resource Owner Password
Client id, Secret	N/A	Obtained during the prerequisite step
Username, Password	N/A	Obtained during the prerequisite step
Scope	Corresponds to the Process Automation instance (typically https:// <opa-test-url>/process</opa-test-url>	Corresponds to the Process Automation instance (typically https:// <opa-test-url>/process</opa-test-url>
Token URL	N/A	Corresponds to the IDCS instance connected to Process Automation (typically
		https:// <idcs host="">/oauth2/v1/token)</idcs>

Once this is done, you'll no longer see a warning on the opacustom backend's Overview tab.

Step 4: Preview and Deploy to the Test Pod

- 1. In VB Studio, click **Preview** to ensure that the extension is running as expected.
- Commit the sources and push them (or merge them) to the branch used to package your extension.
- If the build pipeline isn't configured for the extension, configure the build pipeline and deploy the extension on the Test pod.
- **4.** Test the extension to ensure everything works correctly.



Step 5: Deploy to Production and Other Pods

In this step you'll deploy your extension to an appropriate prod instance of Process Automation, just as you used a Process Automation test pod for testing. To do this, you'll need to supply new details about the <code>opacustom</code> backend on the Oracle Cloud Application Manage Backends for Visual Builder Studio screen. If you want to deploy to other instances, follow this procedure to configure the backend <code>opacustom</code> for <code>each</code> instance.

Note:

If you're using the identity propagation authentication method, make sure that you have a paired a Process Automation instance to whichever pod you are deploying to, so that the OAuth 2.0 User Assertion authentication will work.

- 1. Create the production build pipeline for the Prod instance according to this link.
- 2. Navigate to the opacustom backend's Overview tab.
- 3. To make things easier on the Oracle Cloud Applications side, click **Copy Link** to add the current URL value (your test instance) to your buffer.
- 4. Paste the copied URL into a text editor, then replace the base portion of the URL with the base URL for the prod Oracle Cloud Applications instance. For example, replace https://fa-test.oraclelcoud.com/withhttps://fa.oraclelcoud.com.
- 5. Use this link to go to the Manage Backends for Visual Builder Studio screen on the production pod you want to deploy to. If you don't have the required Administrative privileges, send the link to your administrator to complete the next step.
- 6. Use the same information you used in Step 3: Configure the Backend for Your Test Instance, except for **Scope**, use the URL for the Process Automation prod instance, typically https://copa-prod-URL>/process.
- 7. Run the production build pipeline, then test the deployed extension in the production pod.

Connect to Google Drive Using the OAuth 2.0 Authorization Code Grant

Here's an example of how to connect to an external REST service, like Google Drive, that supports the OAuth 2.0 Authorization Code Grant. The Authorization Code Grant method is supported by a wide variety of REST APIs.

In this example, we'll look at a scenario where you use the Authorization Code Grant method to connect your extension to Google Drive in order to fetch files from a given folder.

Note:

This example applies to Oracle Cloud Applications 25C and later.

Here are the steps for setting up this scenario:

- 1. Create Client Apps in Google
- 2. Create a Backend for a Local Server
- Add Service Connections to the Backend



- 4. Test the Service Connection
- **5.** Configure the Backend for Your Test Instance
- 6. Preview and Deploy to the Test Pod
- 7. Deploy to Production and Other Pods

Create Client Apps in Google

To connect to Google Drive using Authorization Code Grant authentication, your extension needs a client ID and secret. This client ID identifies your web app to Google's OAuth servers. To generate these, you'll create a client application from the Google console.

In this example, you'll need two client IDs: one for development and testing and another one for production. For this reason, you'll create two separate client apps in this step.



When you create a client app, Google generates a client ID and client secret. Make a note of these, as you'll need them when you create a custom backend.

Log in to the Google console at https://console.cloud.google.com and create your client apps with these capabilities and settings. Refer to the Google documentation here: https://developers.google.com/identity/protocols/oauth2/web-server:

- Data access for the scopes you require for Google Drive.
 Add the scopes you require, such as https://www.googleapis.com/auth/drive, in the Data access section of the client app. If you need additional scopes for greater granularity or to other Google services, refer to your Google documentation.
 - For example, if you just want to view and manage metadata for Google Drive files, you may decide to use https://www.googleapis.com/auth/drive.metadata instead.
- Appropriate values for Authorized redirect URLs and Authorized Java Script Origins as described in this table.

otion
the landing URL in VB Studio or in Oracle Applications:
ual Builder Studio: 'B Studio root url>/oauth/ llback
rexample, https://abc- st.developer.ocp.oracleclou com/oauth/callback
acle Cloud Applications: tps:// .oraclelcoud.com/api/ dinfra/trap/2/oauth/
1



Property	Description
Authorized Java Script Origins	This is only required if the REST API is being consumed directly from Visual Builder Studio (when the Connection Type is "Dynamic, Service supports CORS").
	Add the Visual Builder domains such as https://abc-test.developer.ocp.oraclecloud.com/



Step 1: Create a Backend for a Local Server

In this step, you'll create a backend to represent Google Drive. This is a one-time configuration for your extension. Since you're going to use one server for testing and a different one at runtime, you'll want to use a *local server* for this backend, which means it is used only during test and development and thus isn't stored in the source code.



Tip:

For reusability purposes, it's a good idea to add all your frequently used backends to a common extension, deploy it, then use this extension as a dependency wherever you need it.

To create a custom backend for Google Drive:

- In the Services panel, click the Backends tab, then click Add (+) to create a custom backend.
- 2. Provide a valid name for the backend, like googleOAuthAuthCode, then click Next.
- 3. In the backend's **Server** tab, add these values:

Local Server	Select this check box. This check box indicates that these backend server details are not part of the extension itself, but will be used only during development and test.
URL	Typically, this is https://www.googleapis.com. Refer to the Google Drive API documentation here to confirm this value: https://developers.google.com/workspace/drive/api/reference/rest/v3
Authenticati on	OAuth 2.0 Authorization Code Grant
Client ID and secret	Enter the client ID and secret you generated in Create Client Apps in Google for your development and testing environments.
Scope	Provide all the required scopes, such as: https://www.googleapis.com/auth/drive



Authorizatio n URL	Typically, https://accounts.google.com/o/oauth2/v2/auth
	Leave this property blank unless your extension requires specific authorization parameters. Refer to the Google documentation here to determine your needs: https://developers.google.com/identity/protocols/oauth2/web-server#creatingclient
Token URL	Typically, https://oauth2.googleapis.com/token
Open authenticatio n page as	Choose either New tab or New window for the Google login flow. At login, the user is presented with either a new tab or a new window which is then closed when the login process is complete.
Other fields	Leave this property blank.
Connection Type	Choose a connection type option: • Always Use Proxy: Requests to the REST API go through the VB Studio proxy. This is a trusted server-side component that calls external REST services hosted on external domains on behalf of your App UI. This property is recommended for testing.
	 Dynamic, Service supports CORS: Requests to the REST API go directly from the browser. This option requires extra configuration on the REST API side to allow CORS. For example, you must provide a value for Authorized JavaScript Origins in the client app.

4. Click Create.

Step 2: Add a Service Connection for the Backend

Next, add a service connection that your extension requires to the backend you just created.

 From the custom backend's Overview tab, click + Service Connection, then select Define by Endpoint.

VB Studio opens the Create Service Connection wizard. The URL field in the wizard shows the backend's URL in this format:

```
vb-catalog://backends/<backend name>/
```

- 2. Enter the rest of the endpoint's URL in the URL field: /drive/v3/files
- 3. Provide a valid service name, like googleDriveAPI.
- **4.** From the **Response** tab, provide a sample response for the REST API you are trying to configure. Here is a sample response for a GET method:



```
},
{
    "kind": "drive#file",
    "mimeType": "application/vnd.google-apps.folder",
    "id": "10plRQFYKN9DD6ifTGfTrSFm53XMaOx_w",
    "name": "test1"
}
]
```

Note:

You'll use the **Response** tab in this step rather then the **Test** tab here because the **Test** tab cannot open the Google login flow in a new tab or child window.

- 5. Click **Save Example** to generate a schema for this example.
- **6.** For methods with a request body, like a POST request, use the **Request** tab to provide a sample request body. As before, click **Save Example** to generate a schema.
- 7. When you have generated schemas for all required REST APIs, click **Create** to create your service connection.

Step 3: Test the Service Connection

In this step, you'll add a button to an App UI with an event that calls the newly created service connection, <code>googleDriveAPI</code>. You do this configure the button with an event that calls an action chain, then configure the action chain with a "RestAction" representing the service connection.

Note:

You can currently only test GET endpoints from the Page Designer using these steps. POST, PUT, and PATCH endpoints are currently blocked when sourced from a local server.

To test the service connection:

- 1. From the Page Designer, drag a button from the Components palette and drop it into position on the canvas.
- 2. From the Events tab on the button's Properties pane, add an oj-action event to the button. See Start an Action Chain From a Component.
- 3. From the Action Chain editor, add a "Call REST" action and provide the <code>googleDriveAPI</code> endpoint in the configuration of the action. See Add a Call REST Action.
- 4. Add a Fire Notification action with "info" on the success of the REST call, and another Fire Notification action on the failure of the REST call. See Add a Fire Notification Action.
- 5. Switch to Live mode and click the button.

 This action opens the Google login flow in a new tab or new window, depending on what you have configured. Once you log in, the new tab or window closes and you are returned to the main flow with the Google API results.



Step 4: Configure the Backend for Your Test Instance

In this step, you'll provide your backend with runtime server details in Oracle Cloud Applications itself. Each VB Studio backend that has a local server, but no in-source server, needs this configuration so you can use the Preview action in Step 5 to test your extension. Preview runs on your Oracle Cloud Application test instance. (For screen shots, see Provide a Backend's Runtime Server Details.)

To do this:

1. Navigate to the **Overview** tab for the backend (googleOAuthAuthCode). You will see a warning there, which indicates that we haven't yet provided Oracle Cloud Applications with the information it needs.

2. Click Edit Configuration.

If you don't have the required administrative privileges, ask your administrator to complete the next steps. If you don't see this button at all, that means your instance has not yet been enabled to use this functionality and you won't be able to continue. Contact Oracle Support for more information.

3. In the Manage Backends on Visual Builder Studio screen, enter these details:

URL	https://www.googleapis.com
Authenticati on	OAuth 2.0 Authorization Code Grant
Client ID and Secret	The relevant client ID and secret. You can provide a different client ID and secret than what was used during development.
Scope	https://www.googleapis.com/auth/drive
Authorizatio n URL	Typically, https://accounts.google.com/o/oauth2/v2/auth
Authorizatio n parameters	Leave this property blank unless your extension requires specific authorization parameters. Refer to the Google documentation here to determine your needs: https://developers.google.com/identity/protocols/oauth2/web-server#creatingclient
Token URL	Typically, https://oauth2.googleapis.com/token
Open authenticatio n page as	Choose either New tab or New window for the Google login flow. At login, the user is presented with either a new tab or a new window which is then closed when the login process is complete.
Other fields	Leave this field blank.

Note:

The **Connection Type** field is not available for Oracle Cloud Applications, as all REST API calls go through the proxy for this authentication type.

4. Click Create.

Once this is done, you'll no longer see a warning on the backend's **Overview** tab.

Step 5: Preview and Deploy to the Test Pod

- From the Designer, click Preview from the header to ensure that the extension is running as expected.
- Commit the sources and push them (or merge them) to the branch used to package your extension.
- 3. If the build pipeline isn't configured for the extension, configure the build pipeline and deploy the extension on the Test instance.
- 4. Test the extension to ensure everything works correctly.

Step 6: Deploy to Production and Other Instances

In this step, you'll deploy your extension to an appropriate production instance, and configure it to connect to a different Google client application configured for production use. To do this, you'll need to supply new details about the <code>googleOAuthAuthCode</code> backend on the Oracle Cloud Application Manage Backends for Visual Builder Studio screen.

If you want to deploy to other instances, follow this procedure to configure this backend for each instance.

- Create the production build pipeline for the production instance according to Create and Configure Production Build Jobs in Administering Visual Builder Studio.
- 2. Navigate to the googleOAuthAuthCode backend's Overview tab.
- Click Copy Link to copy the current URL for your test instance to the clipboard. You'll
 modify this URL to open the Manage Backends for Visual Builder Studio screen for the
 desired production instance.
- 4. Paste the copied URL into a text editor, then replace the base portion of the URL with the base URL for the Oracle Cloud Applications production instance. For example, replace https://fa-test.oraclelcoud.com/with/https://fa.oraclelcoud.com/
- 5. Use the URL you modified in the previous step to open the Manage Backends for Visual Builder Studio screen on the production instance you want to deploy to. If you don't have the required administrative privileges, send the link to your administrator to complete the next step.
- 6. Enter the same information you used in Step 4: Configure the Backend for Your Test Instance, but instead use the client id and secret you created for the production Google client app.
- Run the production build pipeline, then test the deployed extension in the production instance.

