

# Oracle® Cloud

## Using Visual Builder Studio with Classic Application Extensions



Release 24.01

F86708-01

December 2023

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Cloud Using Visual Builder Studio with Classic Application Extensions, Release 24.01

F86708-01

Copyright © 2020, 2023, 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, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

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

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

# Contents

## Preface

---

Audience	vi
Documentation Accessibility	vi
Diversity and Inclusion	vi
Related Resources	vi
Conventions	vii

## 1 How Do I Use Visual Builder Studio to Extend Oracle Cloud Applications?

---

Which Extension Guide Should I Use?	1-1
Migrate Your Extension to the New Version for 22C	1-3
What Is Oracle Visual Builder Studio?	1-4
How Can Visual Builder Studio Help Me Extend My Oracle Cloud Application?	1-5
How Do Visual Builder Studio and Oracle Cloud Applications Work Together?	1-5
What's an Extension?	1-7
What Is a Workspace and Why Do I Need One?	1-8
Use Your Oracle Cloud Application Sandbox	1-8
What Parts of the UI Can I Modify?	1-10
What are Dynamic Components?	1-10
Understand the Designer	1-11
Video: Work With Dynamic Components	1-18
How Do I Customize and Publish an App Extension?	1-18
Upgrade Runtime Dependencies	1-20

## 2 Get Started

---

Open Your Extension	2-1
Create a Workspace	2-2
Create a Workspace Using an Existing Repository	2-2
Create a Workspace by Importing an App Extension Archive	2-4
What is a Scratch Repository?	2-5
Push Your Scratch Repository to the Remote Repository	2-6

Set Up a Build Pipeline for a Scratch Repository	2-6
Use Branches to Isolate Changes	2-9

### 3 Customize Dynamic Tables and Forms

---

Create a Layout for a Dynamic Form or Table	3-1
Edit a Field's Properties in a Layout	3-4
Use Conditions to Hide and Show Fields in a Layout	3-6
Set How User Assistance is Rendered in a Layout	3-9
Configure How Columns Are Rendered in a Dynamic Table Layout	3-11
Set a Field to Display as a Text Area in a Form	3-12
Work with Polymorphic Objects in a Layout	3-15
Determine What's Displayed at Runtime	3-19
Responsive App Display Logic Example	3-23
Use Application and Page Parameters in a Layout	3-25
Preview Different Dynamic Layouts	3-28
Using Field and Form Templates	3-29
Control How a Field is Rendered with Field Templates	3-29
Apply a Template to a Field	3-32
Set a Default Field Template for a Rule Set	3-35
Start an Action Chain from a Field	3-38
Video: Customize the Fields in Your Dynamic Layout	3-40
Control How a Form Layout is Rendered	3-40
Apply a Template to a Form	3-45
Add and Group Fields in Dynamic Form Layouts	3-47
Create Fields For a Layout	3-49
Create a Virtual Field	3-54
Add Converters and Validators to Fields	3-56

### 4 Display Your Own Content

---

How Do Cases Work?	4-2
Configure a Dynamic Container	4-2
Re-Ordering a Container's Content	4-7
Guidelines for Working with Container Sections	4-8
Add Content From a Custom Object to Your Page	4-9

### 5 Advanced Techniques for Customizing Your App Extension

---

Advanced Layout and Dynamic Component Editors	5-1
Customize Your App Extension with Variables and Constants	5-1
Change the Default Value of an Extendable Constant	5-3

Create or Edit Variables and Constants	5-6
Define the Behavior of Your App Extension with Action Chains	5-8
Define Events in Your App Extension	5-9
Create Event Listeners for Events	5-9
Start an Action Chain From a Component	5-12
Start an Action Chain When a Variable Changes	5-14
Start an Action Chain From a Lifecycle Event	5-15
Start an Action Chain From an Action Chain	5-16
Choose How Your Custom Events Call Event Listeners	5-17
Use Events Defined in the Base App	5-21
Call Custom JavaScript Functions	5-22
Add Custom Resources	5-24
Add Pages To an Oracle Cloud Application	5-26

## 6 Preview and Share Your Application Extension

---

Preview Your Application	6-1
Preview in Debug Mode	6-2
Share Your Application Extension	6-2
Export Your Workspace as an Archive	6-3

## 7 Publish the Application Extension

---

Review the Process	7-1
Publish the Application Extension	7-2
Publish Changes Without Creating a Review	7-4
Publish Changes that Require Review	7-5
Video: Overview of Your Git Repositories	7-7
How Do I Add My Changes to a Remote Branch?	7-7
Merge Remote Branches	7-10
Keep Your Local and Remote Branches in Sync to Avoid Conflicts	7-12
View Your App Extension Deployments	7-13

## 8 Troubleshooting

---

Resolving the error "Page cannot be previewed"	8-1
--	-----

# Preface

*Using Visual Builder Studio with Classic Application Extensions* describes how to use a web-based visual development tool to extend Oracle Cloud Applications.

## Audience

*Using Visual Builder Studio with Classic Application Extensions* is intended for Oracle Cloud Applications users who want to create, edit and publish classic 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.

## 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>
- What is Oracle Visual Builder Studio in *Using Visual Builder Studio*

- Set Up Oracle Visual Builder Studio for Extending Fusion Applications in *Administering Visual Builder Studio*

## Conventions

The following text conventions are used in this document.

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

# 1

## How Do I Use Visual Builder Studio to Extend Oracle Cloud Applications?

Your Oracle Cloud Applications are built using Oracle Visual Builder Studio and Oracle JET components. You can use these same tools to customize Oracle Cloud Applications with [application extensions](#).

### Which Extension Guide Should I Use?

You may have noticed that the Visual Builder Studio library contains two extensions guides: the one you're currently viewing, as well as *Extending Oracle Cloud Applications with Visual Builder Studio*. Determining which book you need depends on if you are extending an older or newer version of your Oracle Cloud Application.

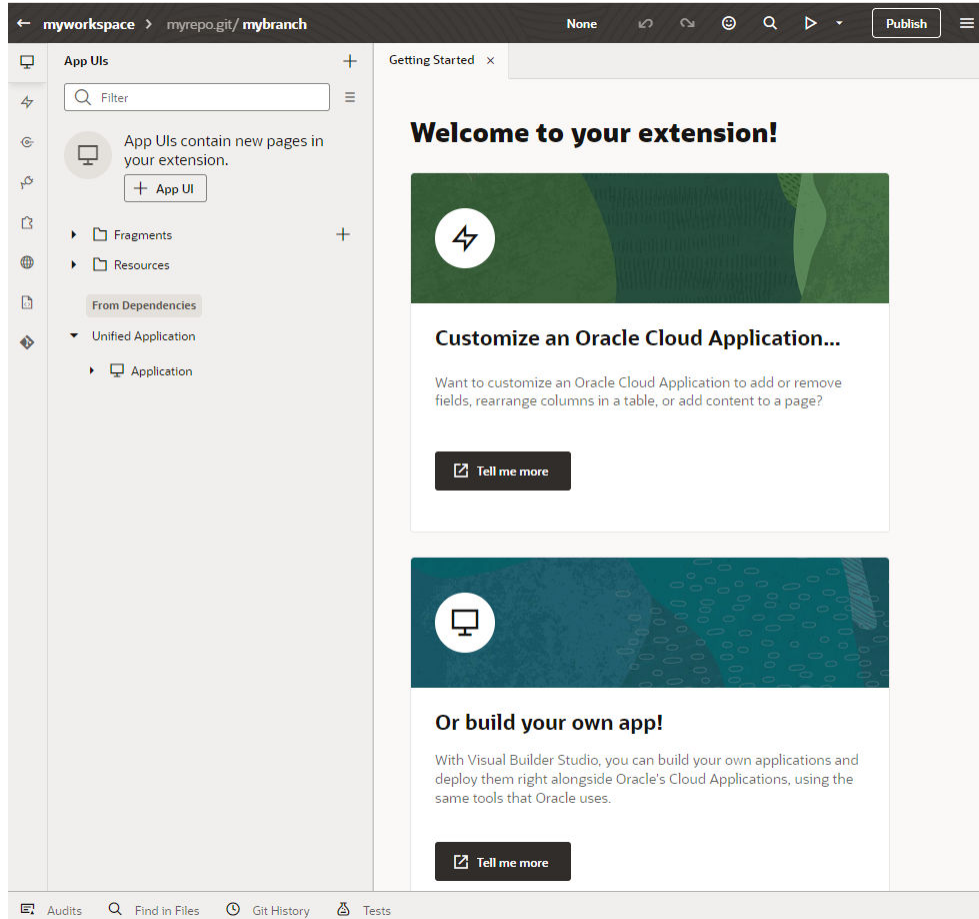
This guide is for you if you are still using an older version of an Oracle Cloud Application (22B and earlier), and still use "classic" application extensions to extend it. Classic application extensions are not supported on newer versions of Oracle Cloud Applications. To edit a classic application extension, you'll need to log into VB Studio and then open the workspace for the extension. For details on how to start editing a classic application extension, see [Open Your Extension](#).

If you are using a newer version of an Oracle Cloud Application (22C and newer), you need to use the *Extending Oracle Cloud Applications with Visual Builder Studio* guide. A good place to start is The Basics chapter.

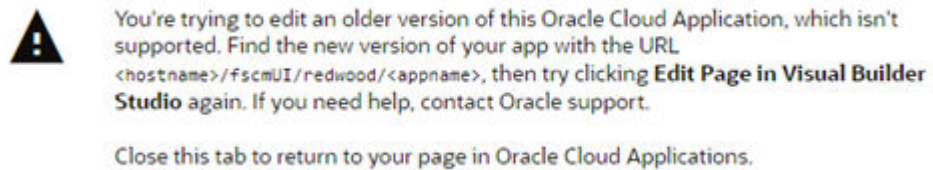
To determine if you are using a newer version, view your Oracle Cloud Applications page, click the **Settings and Actions** menu in the upper right corner, and then click **Edit Page in Visual Builder Studio**.



- If VB Studio opens in a new tab in your browser and your view of the Designer looks something like this, then you're using a newer version of a Cloud Application:



- If a new tab opens and you see a message like this, you can no longer use the older version of your Cloud Application, and you'll need to open the newer version of your Cloud Application:



After opening the newer version in your browser, try clicking **Edit Page in Visual Builder Studio** again to open the extension in VB Studio.

 **Note:**

If you don't see the **Edit Page in Visual Builder Studio** option in your **Settings and Actions** menu, that means your Oracle Cloud Application was not built using Visual Builder Studio and Oracle JET (JavaScript Extension Toolkit). In that case, you should use [Configuring and Extending Applications](#) to help you both add data to your data model, as well as to surface those changes in the user interface.

## Migrate Your Extension to the New Version for 22C

Oracle Cloud Applications that have been updated to 22C use a new structure for developing and registering extensions. App extensions built for version 22B and earlier won't work after your Oracle Cloud Application base app has been updated to 22C.

To use an extension after your Oracle Cloud Application has been updated 22C, you'll need to use Visual Builder's migration tool to convert your app extension to an extension that uses the new structure. After your extension is converted, you can re-deploy it to your 22C Oracle Cloud Application. For more on creating, developing and publishing extensions for 22C Oracle Cloud Applications, see The Basics chapter in *Extending Oracle Cloud Applications with Visual Builder Studio*.

Let's take a look at the basic workflow for migrating an extension. If you've created an extension for your 22B Digital Sales app, you most likely have published it to the TEST instance, and possibly the PROD instance, of your Oracle Cloud Application. When upgrading instances to 22C, Oracle will start by updating your TEST instance. When your TEST instance is updated, Visual Builder will prompt you to migrate your extension to an App UI extension, and you'll have a few weeks to convert and test the migrated extension to make sure everything works on your TEST instance. This way, when your PROD instance is updated to 22C, you'll have an App UI version of your extension ready to publish to your PROD instance.

The Visual Builder migration tool will migrate your app extension to an App UI extension for you, but there are a few things you'll need to do to help make sure the process runs smoothly.

1. Log into Visual Builder Studio and open your workspace.  
If the workspace contains an extension, you'll know your Oracle Cloud Application base application has been updated to 22C if:
  - a. You see a Migrate notification at the top of your workspace, and
  - b. You can't see your extension in the Designer (because it cannot be rendered in the updated base app).

If your base application has been updated to 22C, step through the following steps to start the migration process for your extension.

2. Create a new branch for your migrated extension.
  - If there are active branches, you'll need to migrate them, although without publishing. You can also choose to create a new Git repository for the new extension.
  - Select the Git repository that has your app extension. We'll assume you are using the main branch in your repository.
  - Create a branch off main to create a v1 branch (you can use this for any hotfixes you might need to make to your classic extension in the production while testing the new extension).

- Create a new branch (for example, mymigration) off main for the migrated extension.
- 3. Switch to your new mymigration branch in your workspace.
- 4. Click **Migrate** in the Migrate notification.
- 5. Open the Workspaces page, then click **Clone from Git** to create a new workspace.  
In the Clone from Git dialog box, select the branch with your migrated extension.  
Click **Create**.
- 6. Open the new workspace.  
You can now see the migrated extension in the Designer.

When you are ready, you can publish the extension. Publishing will merge your changes in the branch back to main.

## What Is Oracle Visual Builder Studio?

Oracle Visual Builder Studio (VB Studio) is a robust application development platform that helps your team effectively plan and manage your work throughout all stages of the app dev lifecycle: design, build, test, and deploy. It makes it easy for your entire team to develop the artifacts they need, including:

- Oracle Cloud Applications developers, who need to extend their applications with business-specific customizations;
- Low-code developers, who want to create applications using a visual designer;
- Experienced programmers, who want to modify the source code for applications created by others, or to develop bespoke apps using the web programming language of their choice.

With VB Studio you get:

- A rich visual designer integrated with source control (Git) so that developers can manage changes, apply version control best practices, and collaborate with their teammates to develop applications
- The ability to build and display different flavors of the UI to meet the needs of discrete users of certain Oracle Cloud Applications (those built with VB Studio and Oracle JavaScript Extension Toolkit (Oracle JET)), also within a Git framework
- Built-in repositories for hosting code in Git and for hosting binaries, such as Maven dependencies
- A continuous integration service so you can automate your build and test systems
- A continuous delivery service that tightly integrates with Oracle Cloud Applications
- Agile boards and an issue tracking system for tracking sprints, tasks, defects, and features

VB Studio enables developers to easily deploy their applications to their preferred target, whether it's a staging or production instance of Oracle Cloud Applications or an Oracle Cloud Infrastructure (OCI) service instance.

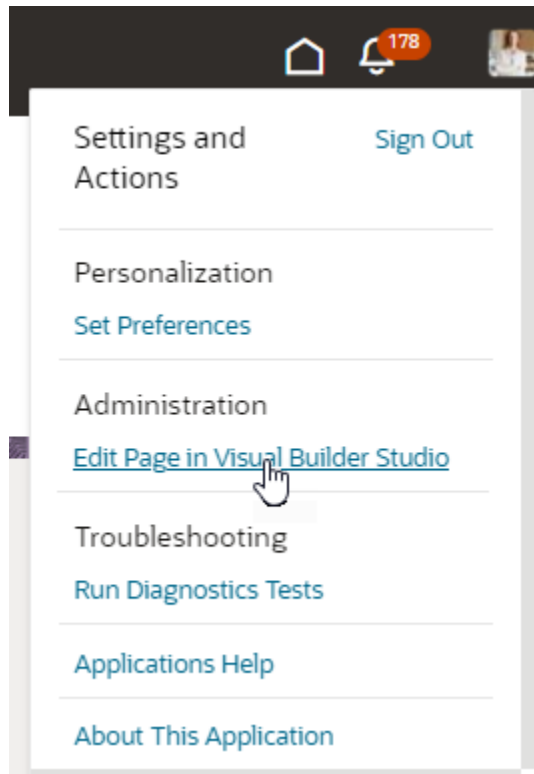
## How Can Visual Builder Studio Help Me Extend My Oracle Cloud Application?

VB Studio provides tools to help you work with your team members to develop an application extension and manage the entire development lifecycle, from creation to publication. VB Studio can help you and your team members:

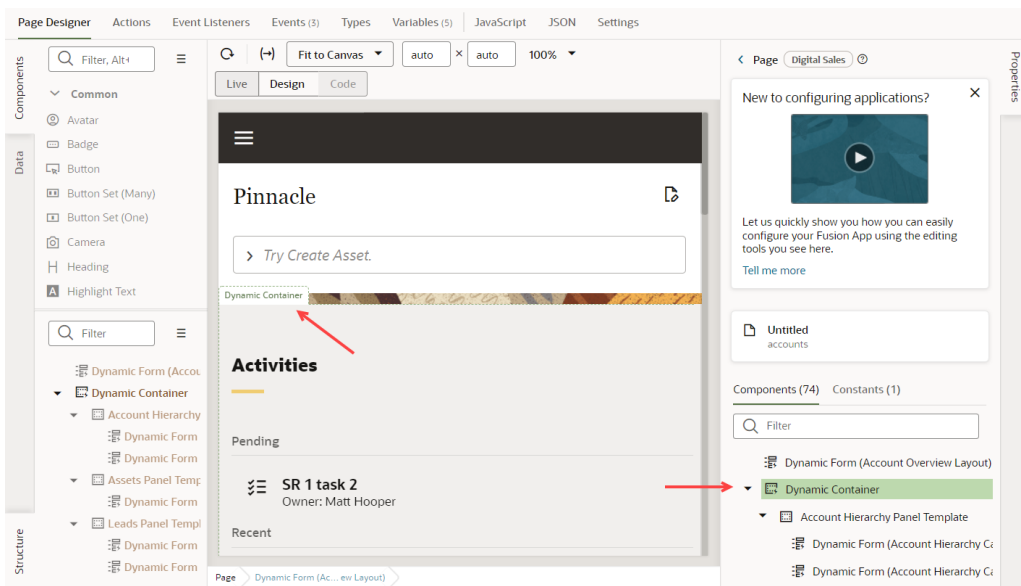
- Create new application extensions and edit existing ones.
- Edit page layouts and display logic using a visual editor.
- Version the changes made to an extension so you can revert to previous versions, if needed.
- Collaborate on an application extension, so that both experienced and novice developers can contribute.
- Participate in the code review process.
- Test the changes on your Oracle Cloud Application development environment.
- Share your changes during development and provide feedback.
- Publish the changes to the application's production instance.

## How Do Visual Builder Studio and Oracle Cloud Applications Work Together?

[Configuring and Extending Applications](#) describes how you can use Oracle Cloud Application tools to customize **most** Oracle Cloud Applications to meet your business needs. However, if you check the Settings and Actions menu while in a "Redwood"-style Oracle Cloud Application (Redwood is Oracle's next generation user experience design system) and see an option for "Edit Pages in Visual Builder", that's your signal that you need to use VB Studio to make your changes to the app's user interface:



If you require any underlying data model changes, you should use Application Composer to make them *first*, then use the **Edit Pages in Visual Builder Studio** option to expose those changes in the UI with VB Studio. For example, you may want to add a new field to a table, or rearrange the fields in a form so that those you care about the most are featured. You can do this using the Designer in VB Studio, by manipulating the dynamic forms and tables that appear on your Oracle Cloud Application's pages.

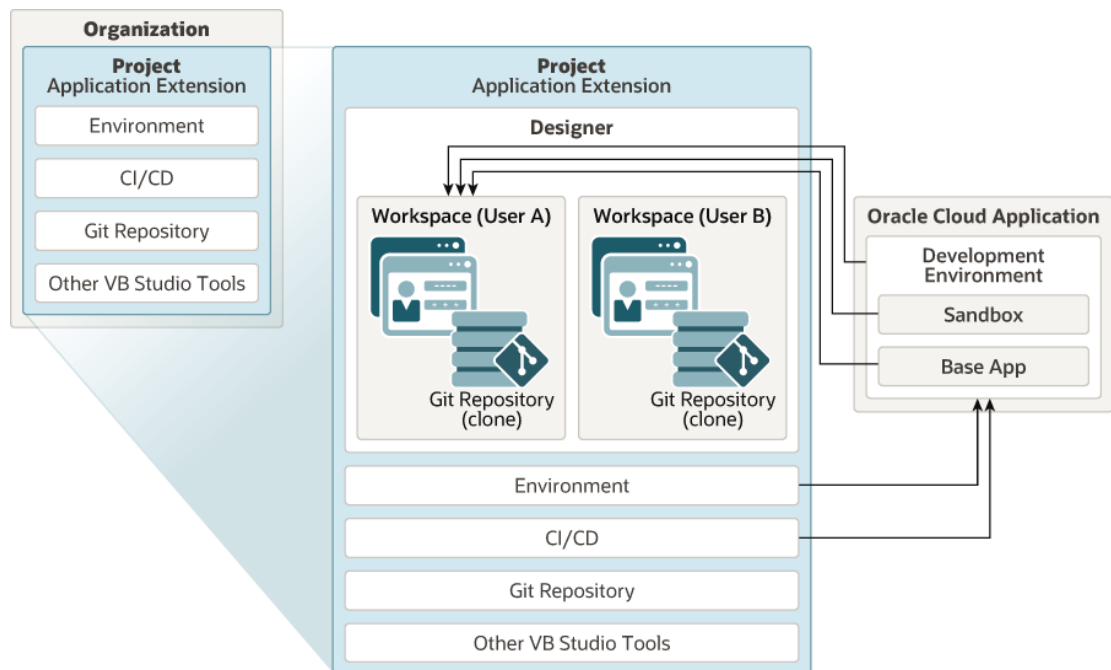


The Designer's rule set editor lets you create and edit the rules that determine what is displayed in a dynamic form or table. For example, you may want to display one set of options for a form when the end user has the "manager" user role, but another set of options when the user has a different role. You can create a different layout for each case, then set a rule to apply the correct layout based on the end user's role.

You can even include other components in your page—for example, images or links—and then define their behavior by creating action chains.

## What's an Extension?

The changes you make to your Oracle Cloud Application in VB Studio are stored in an artifact called an *extension*. Physically, the source files associated with the extension are stored in a Git repository. When working on an extension, the best practice is to have only one extension for the *base app* (the Oracle Cloud Application you're extending) in the project, and to store the source files for the extension in the same Git repository in the project. Multiple developers can work together to develop the extension, but they should all be working from the same repository.



Here are the key components of the VB Studio ecosystem as it pertains to extensions:

- Within a single VB Studio instance, you and your team members who use that instance are considered an *organization*. Within your organization, you will likely belong to one or more *projects*, each of which is devoted to a discrete software effort. For example, you might have a project for building extensions for an Oracle Cloud Application, like Digital Sales, and another project for building a bespoke application your own department will use. A project brings together all the tools you need to create those artifacts, such as a Git repository for storing your source code, a pipeline to provide continuous integration and delivery, an issue tracking system, team wikis, and more. For details on how to create projects in VB Studio, see Set Up Oracle Visual Builder Studio for Extending Oracle Cloud Applications in *Administering Visual Builder Studio*.

- The best practice is to have a project dedicated to a single Oracle Cloud Application, so that all work for that App is stored in the same Git repository. When setting up a workspace, developers working on the extension should clone the project's Git repository and begin working from there.
- When you work on an extension, you do so within the editors provided in the VB Studio Designer. Depending on which aspect of the page you're customizing, VB Studio invokes the proper editor to provide the experience you need.
- 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.

## What Is a Workspace and Why Do I Need One?

A workspace defines the resources available to you when you open the Designer. You can think of a workspace as your editing context while you're working with the Designer.

Before you can use the Designer to edit an application extension, you must first select or create a workspace to identify the source files you'll be editing. A workspace defines the repository—and the branch—containing the source files you want to use, the application's development environment or test environment where you plan to publish your app, and, in certain cases, a sandbox, which contains data model changes made to the app which haven't been published yet. The Git repository and development environment must already be set up and defined in the application extension project in VB Studio before you can create a workspace. If the development environment isn't defined, you won't be able to create a workspace.

You can publish your app to an environment manually from your workspace, or wire it up to a pipeline to do it automatically, such as when a developer on your project merges his or her branch to the main branch.

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) deploy them. You can have multiple workspaces, each with a different branch and sandbox, or you can use one workspace and switch to a different branch and sandbox when you are in the Designer.

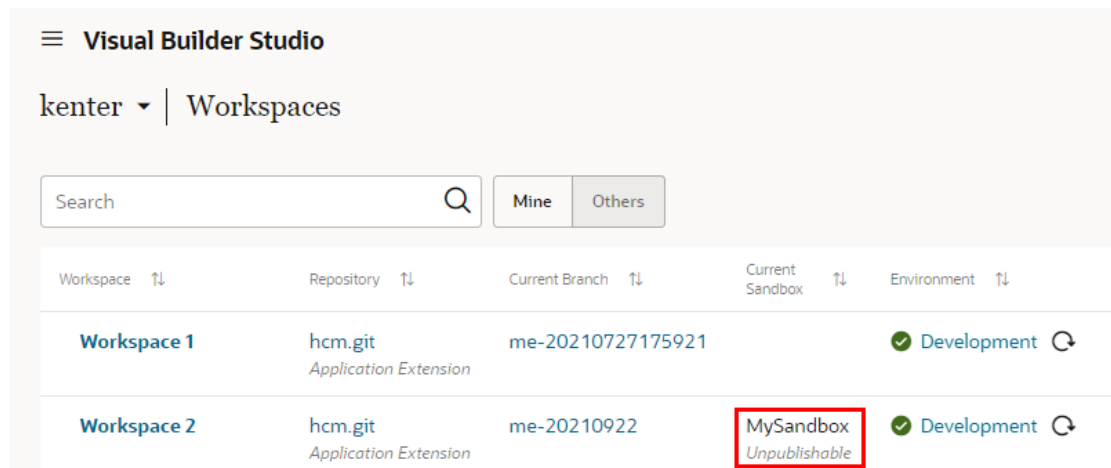
## Use Your Oracle Cloud Application Sandbox

VB Studio can access the sandboxes in your Oracle Cloud Application 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 app extension, you can associate the sandbox with your workspace so you can access it while you're working on the app extension. You'll want to do all your work in a branch that is associated with the sandbox with those changes, and you'll want to continue to use the same branch with that sandbox until you are finished. If you need to edit the app 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. If it helps, 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.

When selecting a sandbox for a new workspace, or when switching the sandbox used with a workspace, VB Studio will indicate if any of the available sandboxes created in App Composer are "unpublishable". If a sandbox is unpublishable, you won't be able to publish it when you publish your app extension. Using an unpublishable sandbox in your workspace increases the risk that your app extension might not work correctly when you publish it because the sandbox might be out of sync with your Oracle Cloud Application's data model.

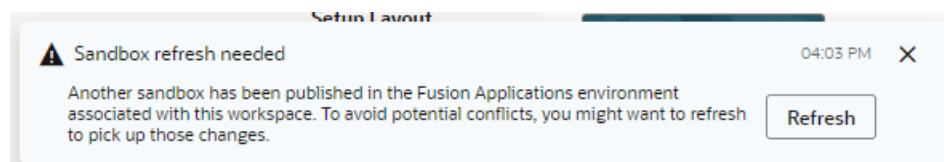
In your Workspaces page, you'll see a note in the Current Sandbox column if any of the sandboxes you're using with your workspaces are unpublishable.



If you open a workspace that uses an unpublishable sandbox, you'll see a lock icon in the header next to the sandbox's name. Also, in the dropdown list of your workspaces, if any of your workspaces use a sandbox that is unpublishable you'll see a lock next to the sandbox name.

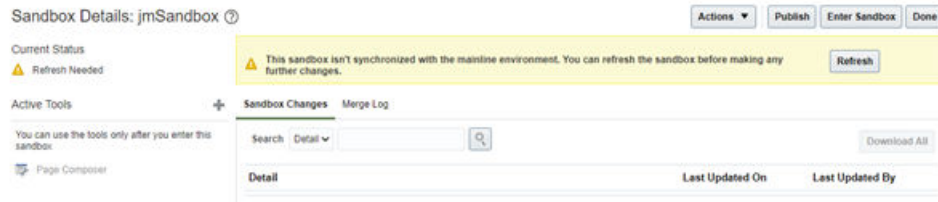


VB Studio will display a notification if the sandbox you're using with your workspace has changed (for example, it was published or deleted). The notification will contain a description of the change, and suggest what you should do. In this image, the notification displays a message that the sandbox associated with the workspace should be refreshed because of changes in the application's environment.



In this case, the notification box contains a Refresh button you can click to refresh your sandbox. For more about working with sandboxes in Oracle Cloud Applications, see the [Sandboxes](#) section in *Configuring and Extending Applications*.





When you are ready to publish your app extension to your Oracle Cloud Application environment, you should publish the associated sandbox at the same time or shortly before to minimize the risk that the data model and app extension will be out of sync.

## What Parts of the UI Can I Modify?

The base application developer (at Oracle) who created your Oracle Cloud Application made certain elements of the app's UI eligible for customization. You customize these elements in your app extension by modifying the *dynamic components* and the UI components and elements the dynamic components contain. Dynamic components are clearly indicated in the VB Studio UI, so you know what you can extend and what you can't.

There are other elements used in dynamic components you might also need to modify when extending them. For example, adding a button to a dynamic component might involve:

- Modifying a *rule set* to control when the button is displayed;
- Modifying a *template* to control how the button is rendered;
- Creating an *action chain* to define what the button does;
- Creating *variables* used in the action chain.



### Note:

You can't add new pages to your Oracle Cloud Application using classic application extensions. However, if your Oracle Cloud Applications instance is version 22B or newer, you *can* add pages by creating an App UI. To do this, follow the steps to [Create an Extension](#) and [Create an App UI](#) in *Extending Oracle Cloud Applications with Visual Builder Studio*.

After extending your application with the App UI, you can create links to the new pages from your classic application extension.

## What are Dynamic Components?

A dynamic component is an extendable UI component, such as a form, table, or container, that does not render a static set of fields or content. Instead, a dynamic component uses *display logic* to determine what the component displays; for example, what fields are displayed in a table and how they are rendered. 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) to determine what is displayed in the component. The dynamic components that appear in your app are determined by the base app developer at Oracle—you can't add them yourself.

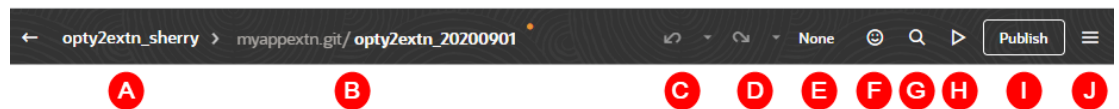
You have two main objectives when customizing a dynamic component: one, to configure the component's content the way you want it using *layouts* and *templates*, and two, to define the display logic that determines the layout and templates displayed in the component. In most cases you define the logic first, then configure the content that will be used in your logic.

There are three types of dynamic components that can be used in 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	Every dynamic table and form in your app extension is bound to a layout artifact that represents a data resource. 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 see how dynamic components work: <a href="#">Video: Work With Dynamic Components</a>
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.

## Understand the Designer

Let's take a look at the Designer so you can understand what it offers. In the header, you have:

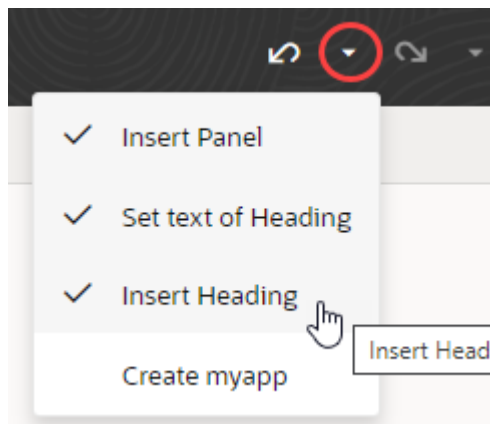


Here's what each element does:

Label	Element	Description
A	Workspace	The name of your current workspace, which defines all the resources you need to customize 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.
B	Git repository / branch	The name of the Git repository for the base app, and the name of the repository branch currently associated with your workspace. Click to open a menu of commands for working with your repository.

---

Label	Element	Description
C	Undo	Undo one or more of your changes. To undo your most recent change, click the <b>Undo</b> icon (hover your cursor over the icon to view the action that will be undone). To undo multiple changes, click the Undo drop-down list and select the actions you want to undo. For example, selecting the Insert Heading action in this image will remove the heading and undo other changes you made after adding the heading:

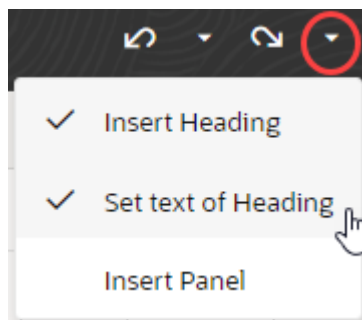


 **Tip:**

You can undo up to 10 of your changes at a time (your last 500 actions are stored in the browser and will be lost if you clear the browsing cache). To undo more than 10 actions, simply undo a few items, then open the drop-down list again.

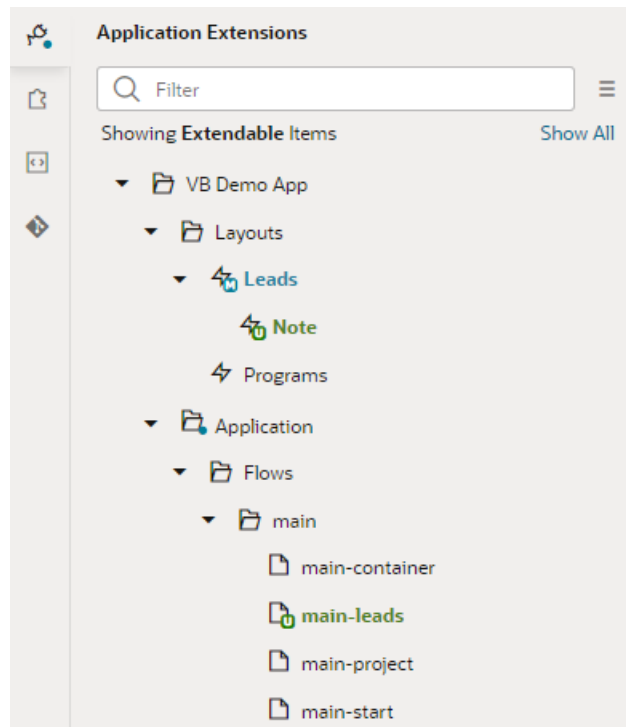
---

D	Redo	Redo one or more changes after undoing them. To redo your most recent change, click the <b>Redo</b> icon (hover your cursor over the icon to view the action that will be 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:
---	------	---



Label	Element	Description
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 <b>Edit Page in Visual Builder Studio</b> 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.
F	Feedback	Submit your feedback about Visual Builder Studio to Oracle.
G	Go to File	Search the Git repository by file name.
H	Preview / Debug Preview	Open a new tab in your browser to preview the current page in the default preview mode. You select the default preview mode in the menu. <ul style="list-style-type: none"> <li>Preview mode uses the optimized versions of the runtime and JET artifacts.</li> <li>Debug Preview mode uses the debug versions of the runtime and JET artifacts. You might want to use this mode when you want to debug the runtime and/or Oracle JET libraries.</li> </ul>
I	Publish	Commit changes <i>in the current branch</i> 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.)
J	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.

You use the Navigator on the left of the Designer to navigate the artifacts in your app extension.



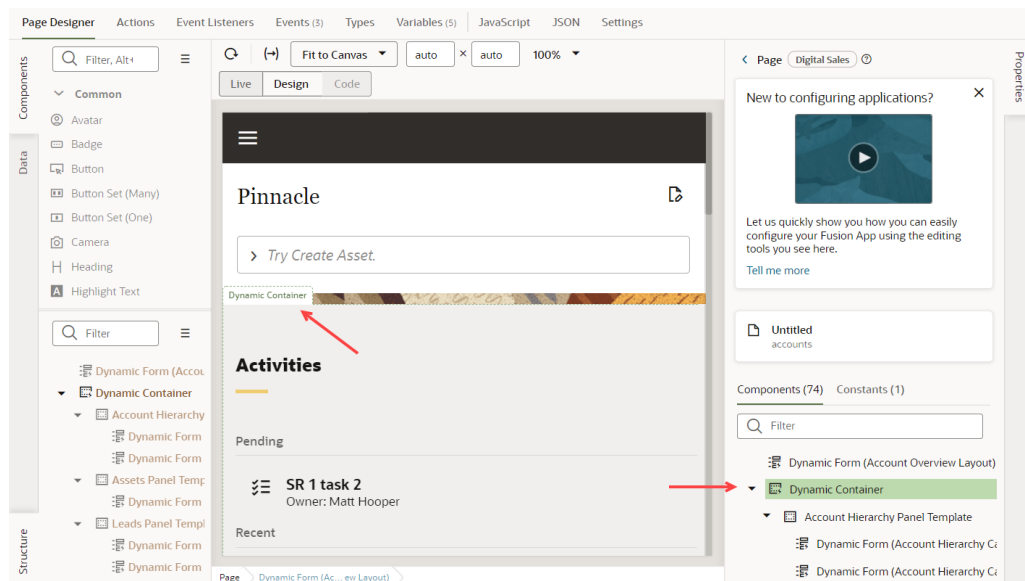
The Application Extensions pane in the Navigator contains two types of artifacts:

- A layout represents a set of data fields that can appear in one or more related dynamic table or form components. Which of these fields *actually* appear in a given component is determined at runtime by the layouts defined in a rule set. You can see all the layouts for the application in the Navigator under "Layouts". Clicking a layout opens a new window containing tabs for editing its artifacts, such as its rule sets, fields and templates.
- Beneath the Layouts node, the extendable pages are grouped by their page flows under "Application". When you select a page, a new window for the page opens on the right containing tabs for editing the page's artifacts, including the Page Designer where you can see how the page looks and select and edit page components.

By default, the Application Extensions pane only displays artifacts that you can extend, but you can use the options menu next to the Filter field to select the filter: All Items, Extendable Items Only, or Extended Items Only.

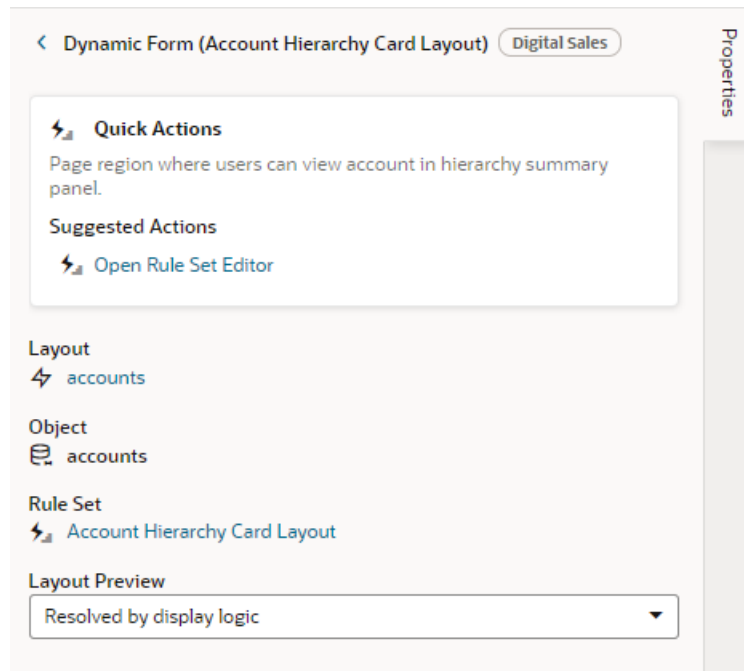
The Page Designer, which appears to the right of the Navigator when you open a page, contains the canvas that displays your app extension pages and a Properties pane. In this image, you can see an app extension page with several dynamic components. Because none of the components are selected, the Components tab in the Properties pane on the right lists all the dynamic components in the page. The Properties pane also has a Constants tab listing editable constants used in the page.

When you hover your cursor over a dynamic component on the canvans or in the Properties pane, it is outlined in green on the canvans and a tab is displayed at the top identifying the type of component. Here the cursor is hovering over a dynamic container in the page.

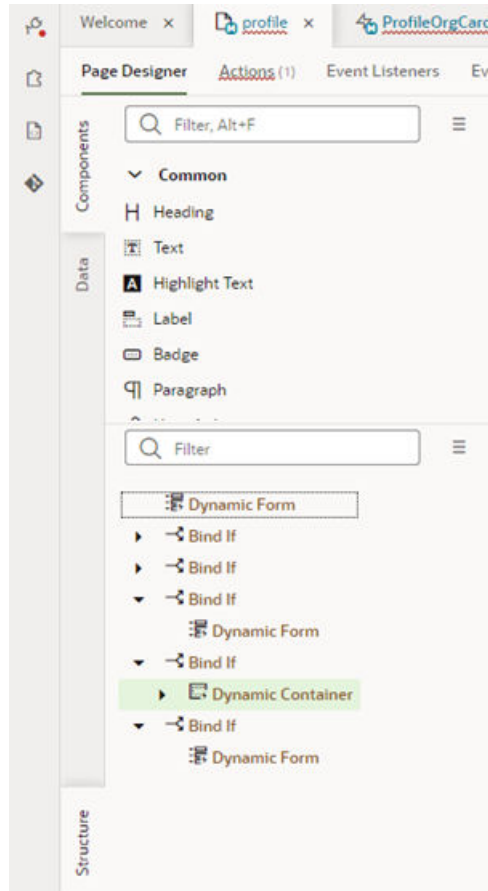


You select a component by clicking it on the canvans or in the Components tab. In this example, a dynamic form has been selected in the page. The form's Properties pane shows that it uses the `Account Hierarchy Card Layout` rule set in the `accounts` Layout, and that the active layout is determined by the display logic. You can edit some component properties directly in the Properties pane, but for others, like

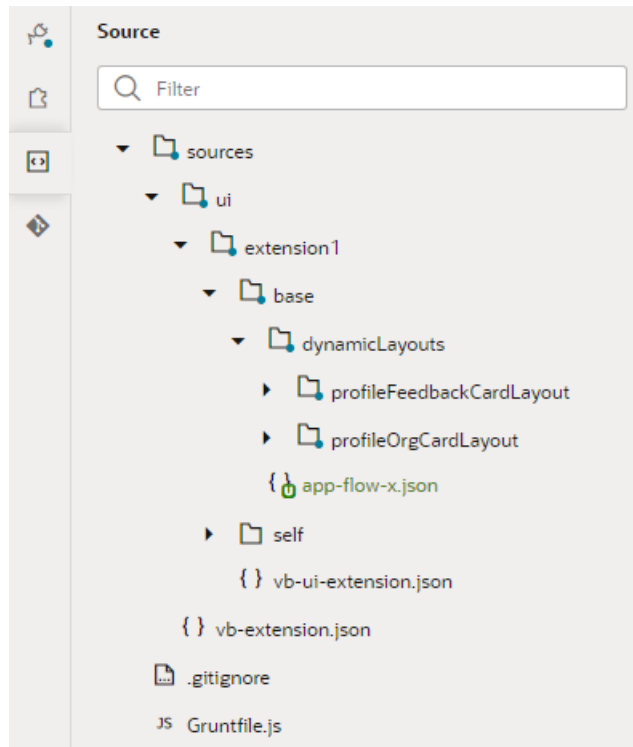
dynamic forms and tables, you'll need to click the link in the Properties pane to open a dedicated editor.



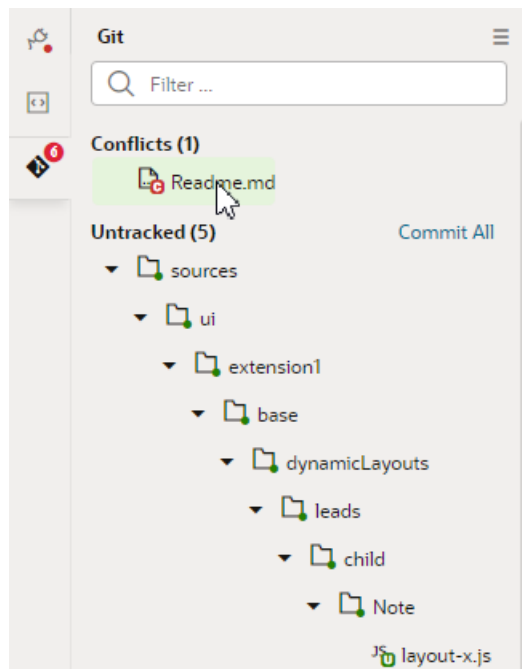
The left edge of the Page Designer has three collapsible panes that you can use to navigate and modify editable components in the canvas. The Components palette lists components that you can drag into editable components, such as dynamic containers. Services that have been marked as extendable in the base app are listed in the Data palette. When a service is listed in the Data palette, you can access them in your page by binding the service to dynamic components in dynamic containers. The Structure view provides a structural view of the components in the canvas you can modify. This can help you locate and select specific components in the page.



The Designer also lets you work in source code, if you prefer. In the Page Designer toolbar, you can click Code to switch from the default Design view to the code editor and see the page's source. You can also click Source in the Navigator, which opens a tree list view of the files in your application extension:



To see a list of files you've changed, but haven't yet committed to your branch, click Git in the Navigator. If you have merge conflicts in your branch you'll see them listed here. In the Git pane you can click files with conflicts to open them in the conflict resolver tool and resolve the conflicts.



At the bottom of the Designer, you have:



Element	Description
Audits	Scan the code in your app extension for places containing errors, warnings, info and to-dos. Your code is scanned when you open the Audits pane.
Find in Files	Search the code in your app extension for a text string.
Tests	View a list of all action chain tests. Action chain tests are not supported for app extensions, so this panel will not display any items.
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.

## Video: Work With Dynamic Components

This video shows how dynamic components work and how they influence the customization options in your app extension.



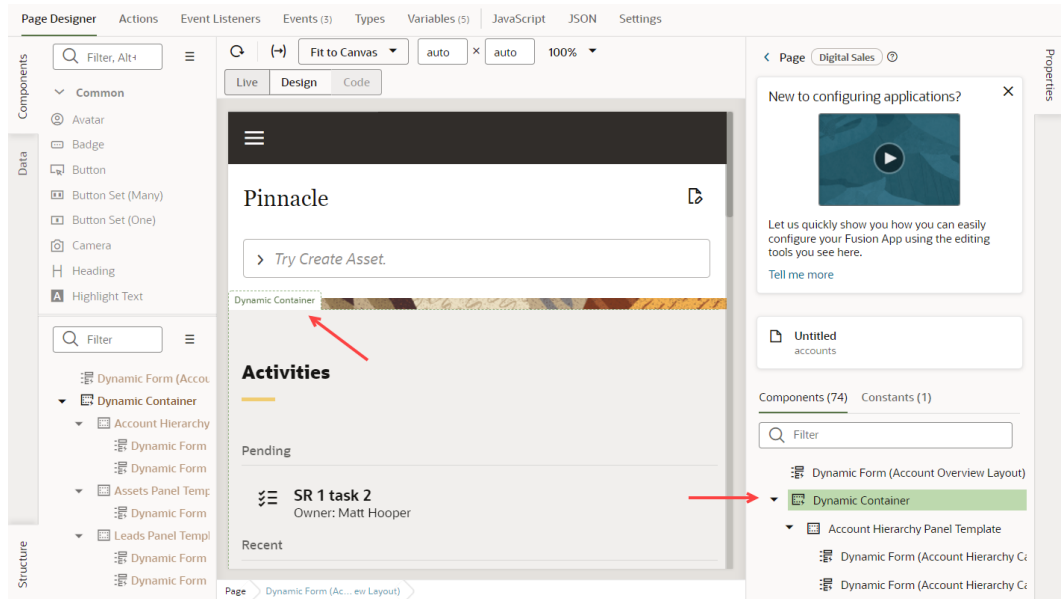
## How Do I Customize and Publish an App Extension?

VB Studio provides all the tools you need for managing the development lifecycle of your app extension.

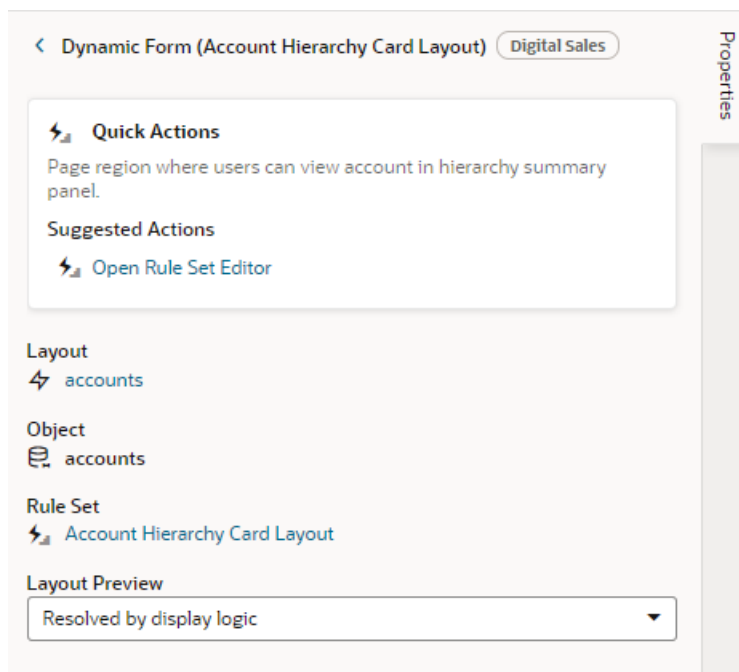
Here's the typical process for customizing and publishing an application extension:

1. Log in to VB Studio.
2. Open the Workspace for your application extension.
3. In the Application Extensions pane, click the page you want to customize to open it in the Page Designer.
4. Using the Designer's editors, make the customizations you want to be part of the application extension.

When you open a page in the Designer, it opens in a new window with several editor tabs. In the Page Designer tab you'll see the page you were just viewing, and you can select dynamic components on the canvas or from the list in the Components tab in the Properties pane on the right. The Constants tab in the Properties pane lists the extendable constants used in the page.



When you select a component (on the canvas, in the Properties pane or the Structure view), you'll see some of the component's properties in the Properties pane. You use the Properties pane to view and edit properties, and to navigate to other editors. For example, if you look at the properties of a dynamic form in the Properties pane, you can see some of its properties and suggestions on what to do next:



At the top is the Quick Actions panel, which holds a selection of suggested actions, dynamically chosen for each component. Most of the time, you'll want to do one of these. Sometimes you may need to change something that's not in the Quick Actions. You can find other component properties below the Quick Actions panel.

When you start editing a dynamic table or form, you'll configure the component's *rule set* to determine the conditions under which each layout is applied to the page. If none of the conditions apply, the base layout is used.

You'll see that each rule set has a default base layout that's read-only. You'll have to duplicate the rule set's default base layout or create a new one to make changes. In the layout you've made, you can move, edit and delete the fields displayed in the component, and add any field that's in your app's data model. You can also create custom fields that you can use in your layouts.

When editing dynamic containers, you'll use *display logic* to determine the sets of components that are displayed in the container.

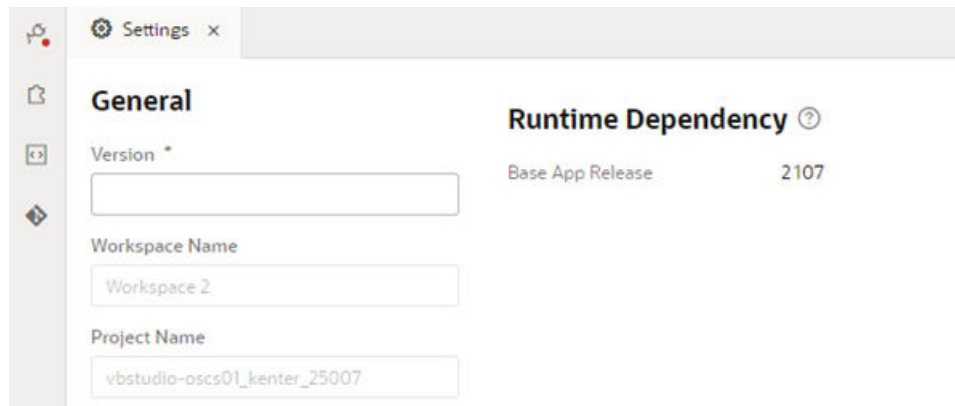
5. View your changes with the preview tools.  
Use the Layout Preview in the Properties pane when editing a page to check your layouts in the Page Designer, and Preview in the Designer header to see your changes as they will appear in the page in a browser.
6. Share your changes to get feedback.  
It's good practice to ask your team members to review the changes you make to an app extension before you publish it. You can use the Share tool to generate a URL for the app extension preview that you can share with team members so they can test it. 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.
7. Commit your changes to a branch.  
You use the Git commands in the menu to commit and push the changes in your workspace to the remote branch. The Publish action can automatically perform this step and the next two steps (create a merge request and merge the branch) for you.
8. 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.
9. Merge your branch into the main repository.  
When your changes have been reviewed and approved, you can merge your branch into the main repository. The administrator can configure the project so that merging changes to main will automatically trigger a build job. The build job will create the build artifact that is deployed to the Oracle Cloud Application test environment. After testing, a separate build job will publish the build artifacts to the live application.

## Upgrade 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 application extension, like what JET components you can use. When your app extension was created, sets its runtime dependency version was automatically set to the Visual Builder runtime version of the base app. For more about runtime dependencies in Visual Builder, see *Manage Runtime Dependencies in Building Responsive Applications with Visual Builder Studio*.

In most cases, VB Studio will migrate the app extension's runtime dependency version in your workspace when the base app release is updated. To see the runtime

dependency version of your base app, click **Menu** in the upper right corner, then select **Settings**:



A notification will appear in the Designer prompting you to migrate your app extension's dependencies if they need to be migrated to a newer version and VB Studio hasn't migrated it automatically. It's recommended that you migrate the runtime dependencies when prompted to take advantage of improvements provided by newer releases.

 **Note:**

You can't migrate your app extension if you have any uncommitted changes. You'll need to either commit or revert your changes before you can migrate your app extension to the newer version.

When your app extension is migrated to a new release, VB Studio makes changes to your app extension to better align it with the upgraded release. We may, for example, address deprecated properties or move things from one file to another. You'll see details of all the changes made to your app extension during the migration, so you know exactly what happened behind the scenes. If your app has syntax errors, however, you'll need to fix those issues—conveniently flagged for action in the upgrade dialog—before you can upgrade. Here's an example of what you may see when a migration succeeds:

## Upgrade Runtime Dependency



### Migration successful!

Check below to see what we did.

- ✓ **Moved auto-wiring decorator metadata inside model object definitions**  
It is preferable to keep design-time metadata inside the model object definitions they annotate, e.g. for variables or action chains. This keeps the model definition cleaner and easier to maintain.
- ✓ **Remove the unused version attributes from application and layout json files**  
The attributes are not necessary anymore, as they have been replaced by the `source.version` in `visual-application.json` starting from 2010.
- ✓ **Migrate variable persisted property from "local" to "device"**  
"local" has been deprecated since 0.9.7 and replaced by "device"
- ✓ **Remove unnecessary service metadata which point to the default service locations**  
Service connections no longer use implicit/default entries in application and flow json files starting from 21.07

Close

Immediately after migration, if you decide you don't want the upgrade, you can use the **Undo** icon in the header to roll back all the changes, though upgrading is the recommended course of action.

# 2

## Get Started

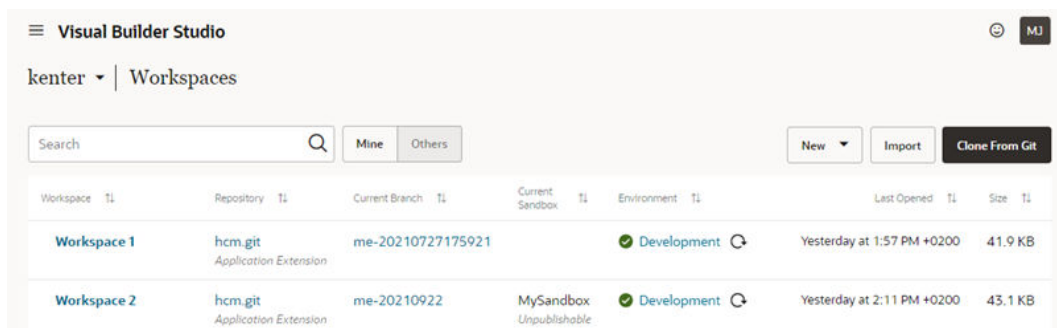
To work on your application extension, you need to have a workspace. There should already be a workspace for the extension, but if you need another workspace for some reason, you'll need to create the workspace manually.

If you're not sure why you need a workspace, see [What Is a Workspace and Why Do I Need One?](#)

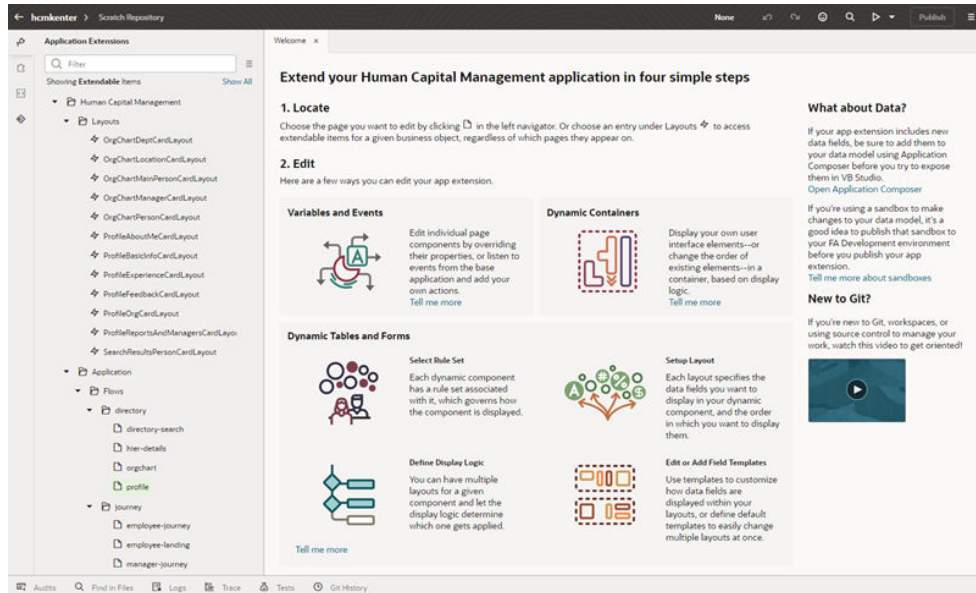
## Open Your Extension

If you're used to clicking **Edit Page in Visual Builder Studio** in your Oracle Cloud Application to open an extension, that option is no longer supported for classic application extensions. Instead, to work on an extension you must first open the workspace for the extension directly in VB Studio:

1. Log into VB Studio.
2. Select your project on the Organization page.
3. In the Workspaces pane, click the workspace you've been using for your app extension. If you want to create a workspace for an existing app extension, you can create one in the Workspaces page:



After opening your workspace, you should then see something like this:



At this point you can select an app extension page or a layout in the left navigator, after which you'll be placed in that artifact's editor. More on that later.

## Create a Workspace

A workspace defines the resources available to you when you open the Designer. You can think of a workspace as your editing context while you're working with the Designer.

Before you create a new workspace, make sure a Development environment is defined in your project and it points to an active Oracle Cloud Application instance. Your Project Home page lists your project's environments. Each of your workspaces will probably use the same development environment, because a project typically has only one. If an environment is not defined for your project you should contact an administrator or project owner to add one.

There are two ways to create a workspace.


### Create a Workspace Using an Existing Repository

If you need to work on an app extension that's already been started, even by another developer, you can clone the original Git repository when creating a new workspace.

When cloning an existing repository, you clone the branch whose 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 create a workspace that uses a branch that's also used in another workspace, you'll need to make sure the branch has been pushed to the repository. You'll then be able to clone the branch when creating your new workspace.

To create a workspace by cloning a repository:

1. Click **Workspaces**  in the VB Studio left navigator to view your workspaces in the project.

Even if you've never explicitly created a workspace before, you may see workspaces in this list. That's because a workspace is created for you automatically when you open VB Studio from a page in your Oracle Cloud Application.

2. Click **Clone from Git**.
3. In the Clone from Git dialog, enter a name for your workspace.
4. Select the Git repository to clone.
5. Select the repository branch you want to check out. This is typically `main` for the latest changes, but can be any branch.
6. (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. Select the Development Environment. You might have only one option.

Each of your workspaces will probably use the same development environment because a project typically has only one. The environment must also support the type of project you are working on, so to create a workspace for extending an Oracle Cloud Application, your project must be associated with an Oracle Cloud Application environment. If you don't have an environment, ask the project owner or an administrator to create one for you.

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



### Clone from Git ✕

Workspace name  
New Feature Workspace

Repository Name  
appextn-repo.git

**i** This repository contains an application extension for an Oracle Cloud Application.

Parent Branch  
b1

New branch from selected

New branch name  
newFeature

Development Environment  
Development

Sandbox (optional)  
No sandbox selected

9. Click **Create**.

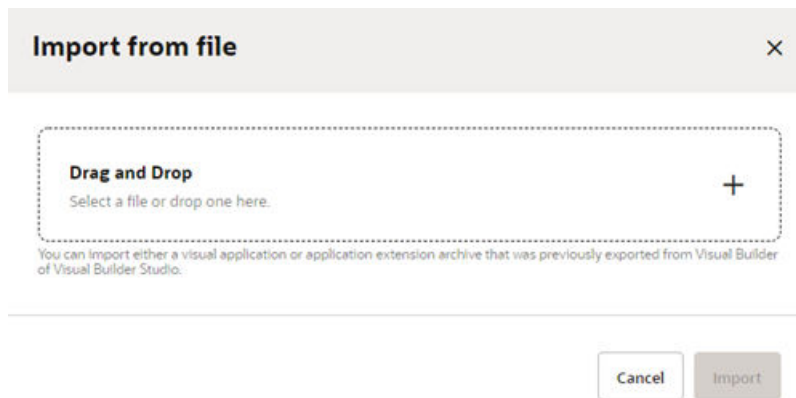
VB Studio creates a workspace and opens it up in the Designer, so you can edit the extension.

## Create a Workspace by Importing an App Extension Archive

If a team member gives you an archive of an app extension, you can import it to create a workspace containing all the files in their branch of the app 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. Click **Workspaces** in the VB Studio left navigator to open the Workspaces page.
2. Click **Import** to open the Import from file dialog box.



3. 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 app extension archive. If it's not, you will see a message that you can't create a workspace by importing it.

4. Type a name for the new workspace.
5. Select the development environment for the app extension, and a sandbox if you need one. Click **Create Workspace**.
6. Choose the Git repository you want to use with the workspace:
  - **Use scratch repository** to create a new scratch repository;
  - **Create new repository** and provide a name for the new Git repository and branch that'll be created in the workspace.

## What is a Scratch Repository?

When you create a workspace, you have the option to create a scratch repository, rather than creating a new repository or using a clone of the project's Git repository. You may want to create a scratch repository when you are 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. If you want to let your team members use your scratch repository, you'll need to push the scratch repository to a **new** remote repository.

No build pipeline is set up for you if you create a scratch repository when creating your workspace. If you want to build and publish artifacts, you might want to create a new repository and branch instead of a scratch repository. When you create a new repository while creating a workspace, a build pipeline is automatically set up for you when the workspace is created.

If you want to build and publish artifacts from a scratch repository, you'll need to first push the scratch repository to a new remote repository. After the new repository is created, you or your project administrator will need to set up build jobs for the repository.

## Push Your Scratch Repository to the Remote Repository

If you chose to use a scratch repository when creating your workspace, you'll need to push the scratch repository to the remote repository if you want other team members to see its contents. Pushing your scratch repository creates a **new** remote Git repository in the project.

1. Open your workspace.
2. In the header, click the arrow next to your Git repo and select **Push** in the menu.
3. In the Push Scratch Repository to Remote dialog, type a repository name. This name cannot be the same as an existing project repository.
4. Enter a commit message, and click **Push Repository**.

**Push Scratch Repository to Remote** ×

*i* Pushing scratch repository to remote will make it a regular repository, available for other team members and allowing more git actions (e.g. branching).

Repository Name \*  
\_git will be appended

Description

Commit Message \*

Cancel Push Repository

After you push your scratch repository to the new remote repository, you can create a pipeline with jobs for packaging the build artifacts from the repository and deploying the artifacts to your environment.

## Set Up a Build Pipeline for a Scratch Repository

When you create a scratch repo, Visual Builder Studio doesn't automatically create package and deploy jobs for your app extension or visual app as it would otherwise. If you decide you want to publish what's in your scratch repo, you'll need to create these jobs manually. After setting up the jobs, you can create pipelines to run the package jobs and deployment jobs in sequence. Usually you'll want to set up one pipeline for deploying your app extension to your development instance, and a second one to deploy to the production environment. The steps for setting up the jobs in the two pipelines are basically the same, but some of the configuration options might differ, for

example, the security options and credentials for deploying to a production environment might be different from those for the development environment.

If you've already created a workspace with a remote repository, your project should already contain packaging and deployment jobs that were automatically set up for the remote repository. While these jobs were configured for a different repository, you can copy them when creating a build job, and then reconfigure the details in your copy as needed. If your project doesn't contain any jobs, you'll need to complete all the steps for configuring the new jobs.

For more detailed steps on how to configure packaging and deployment jobs, see *Create and Configure Production Build Jobs* in *Administering Visual Builder Studio*.

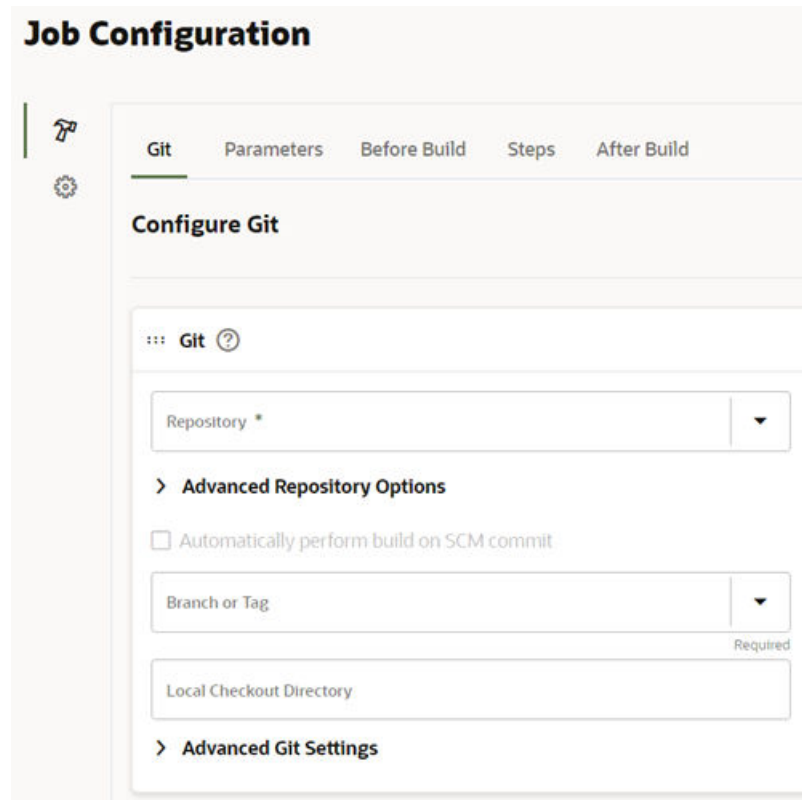
To set up a development pipeline for a scratch repository:

1. Push your scratch repository to create a new remote repository.
2. Click **Builds** in the VB Studio left navigator.
3. Create a build job to package your artifact:
  - a. In the Builds page, click **Create Job** in the **Jobs** tab to create a build job to package your artifact.
  - b. Type a name for the package job and select the **System Default OL7 for Visual Builder** template. Click **Create**.

If your project already contains a similar packaging job, you can select **Copy from Existing** to use it as the basis of your new job. You'll need to set the job's Git repository to your new remote repository.

- c. Set the job's details in the Job Configuration page. Click **Save**.

When configuring the packaging job, you select your new remote repository in the Git tab. If you want the packaging job to start each time changes are added to the remote repository, select **Automatically perform build on SCM commit**.



4. Create a job to deploy your build artifact to your development instance.
  - a. Return to the Builds page, then click **Create Job** to create a new job to deploy your build artifact to your development instance.
  - b. Type a name for the deploy job and select the **System Default OL7 for Visual Builder** template. Click **Create**.  
 If your project already contains a similar deployment job, you can select **Copy from Existing** to use it as the basis of your new job. If you're not deploying to the same instance, you'll need to set the target instance and credentials in the Steps tab.
  - c. Set the job's details in the Job Configuration page. Click **Save**.  
 When setting the target instance, select your development environment.
5. In the Builds page, open the **Pipelines** tab, then click **Create Pipeline** and type a name for your new development pipeline. Click **Create**.
6. Design your pipeline by adding the packaging and deployment jobs you created. Click **Save**.

For the steps for designing the pipeline, see Set Up Production Build Pipelines.

If you set the packaging job to run each time a change is committed to the remote repository, you can use Publish in the header to commit your changes to the repository and start the pipeline. You can also start jobs and pipelines manually in the Builds page.

For details and more configuration options for builds and pipelines, see Set Up the Project to Deploy to Production in *Administering Visual Builder Studio*.

## Use Branches to Isolate Changes

The repository for your app extension was automatically created when your project was created. When you create a workspace, you name the branch in the repository where your changes will be stored. You can have separate branches for each new feature or change to your app extension.

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 you are modifying a table component in your app 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 a 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.

If you want to rename or delete a branch, you can click the branch name in the header to open the Git menu containing the Rename Branch and Delete Local Branch commands.

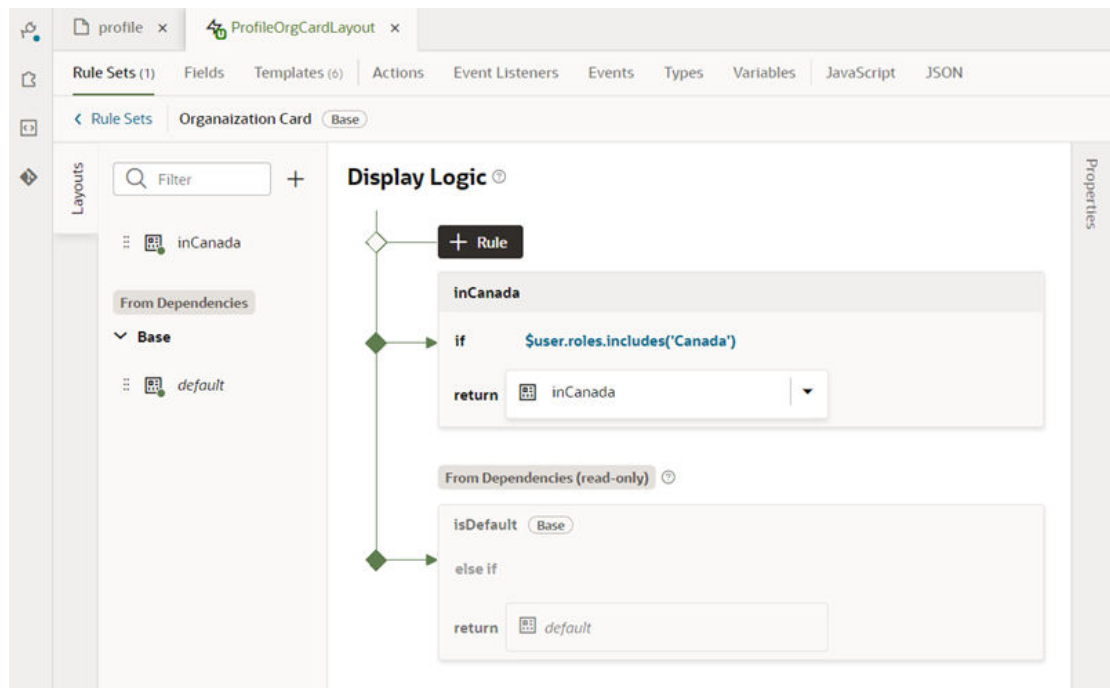
# 3

## Customize Dynamic Tables and Forms

By default, a dynamic component is rendered according to what is specified in the built-in rule and accompanying built-in layout (set by Oracle), but you can customize how the page appears by configuring a component's *rule set* with your own *layouts* and *display logic*.

To open a rule set, you can select the dynamic form or table in the Page Designer and open its rule set from the Properties pane, or, if you know the name of the layout, you can select it in the Application Extensions panel in the Navigator.

Here's what a rule set looks like for the Organization Card dynamic form, with a rule to determine if the user's role includes the string 'Canada', and a layout used when that rule is true.



## Create a Layout for a Dynamic Form or Table

A rule set's *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 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 one layout that shows certain fields in a dynamic form when the user is a manager, and another layout for non-managers. 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 user is a manager—then the layout you selected

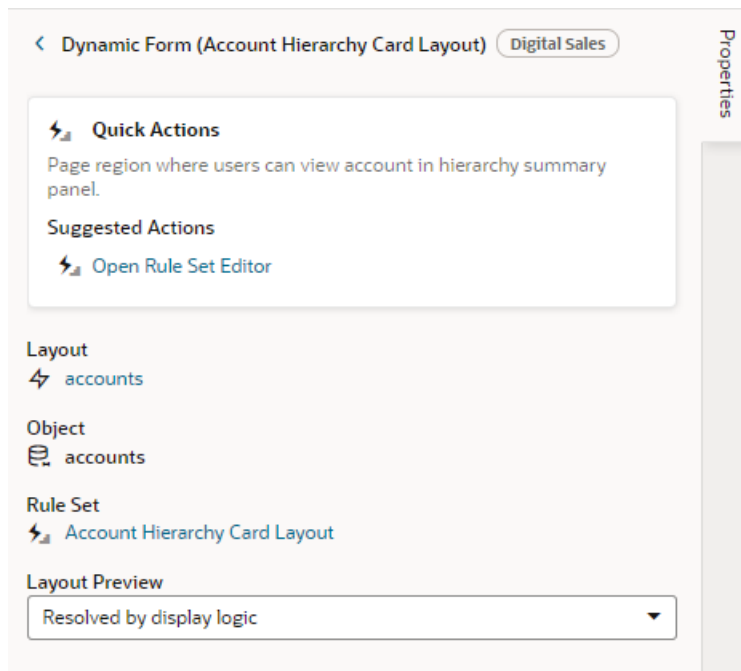
for that rule is applied to the component and the user will see the fields he needs in the form. Non-managers will see a different layout.

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 a layout artifact's data resource, and this data resource is defined by the base app. For example, the base app might define a data resource that has five fields. You can choose which of these five fields (and also other custom 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 layout artifacts, see [Create Fields For a Layout](#).

To create a new layout:

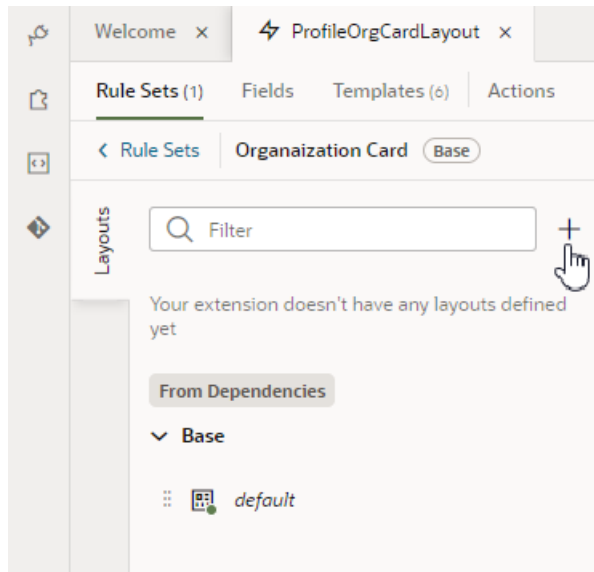
1. When your page is open in the Page Designer, click the dynamic form or table you want to work with in the canvas area, or select it in the Properties pane.
2. Click **Open Rule Set Editor** in the Properties pane for the form or table.



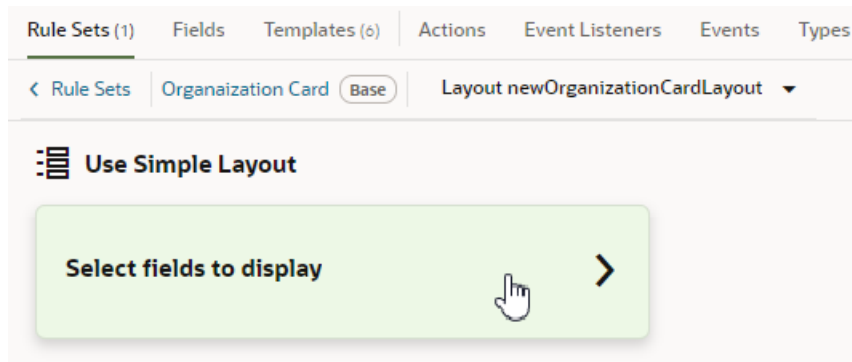
In this image of the Properties pane, you can see the name of the dynamic component's rule set (Account Hierarchy Card Layout).

3. Click **+** in the rule set's Layouts pane and type a name for the new layout, or click **📄** in an existing layout to use it as a starting point.





4. Click the new layout name, then click **Select fields to display** to open it in the layout editor.

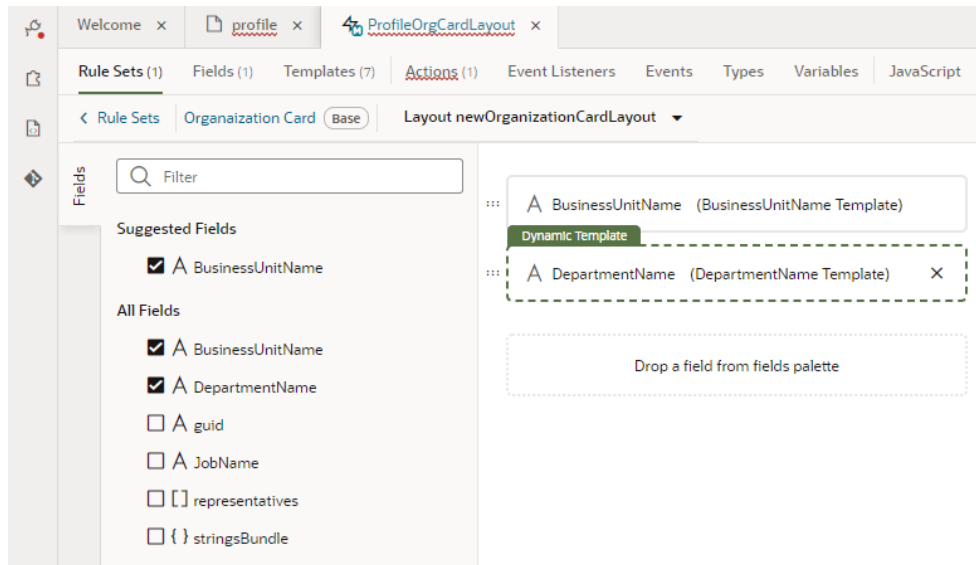


When you create a layout and haven't selected any fields for it yet, you'll see the **Select fields to display** option when you open the layout. (You won't see this option if the layout is a duplicate.) You'll also see the templates that already exist in the rule set listed as layout options. Click a template name if you want apply the template to the layout, otherwise, click **Select fields to display**.

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 layout.

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. 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 `_c` is appended to the name) and the fields required by the layout (the fields where the "Required" property is set to "true" in the layout's Fields editor).

You can also filter the list of fields by entering a string in the Filter field at the top of the Fields palette.

6. Change the order that fields are displayed in the component by dragging fields into position in the center pane.

After a layout is created, you can include it in a display logic rule, as described in [Determine What's Displayed at Runtime](#). 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:

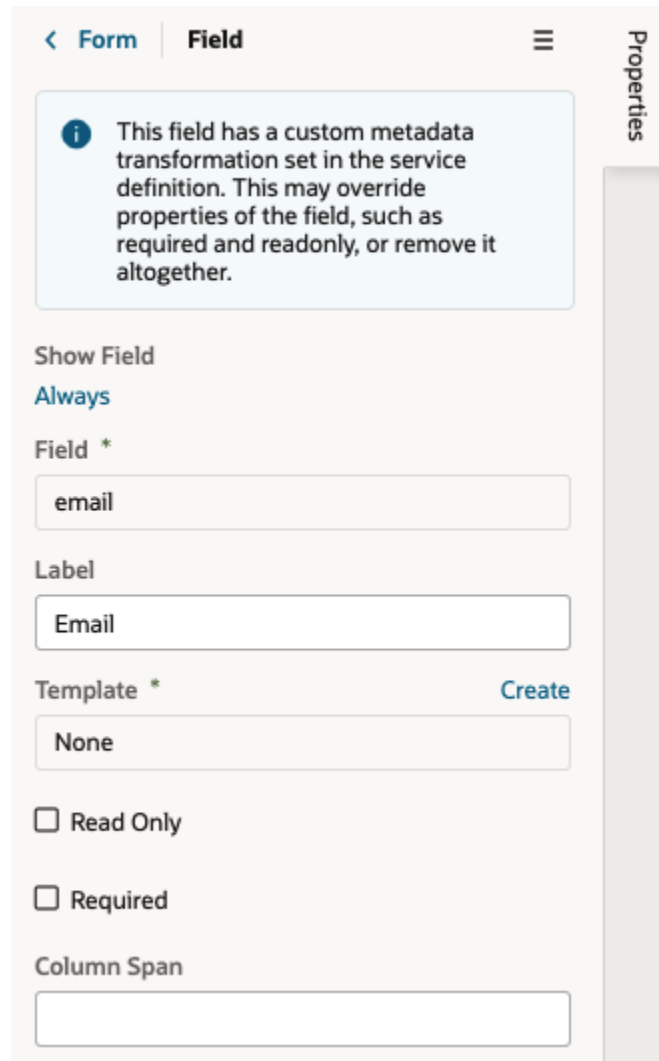
1. In the Rule Set editor, open the layout and select the field in the center pane.
2. Set the Show Field property to control when the field is displayed.

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](#).

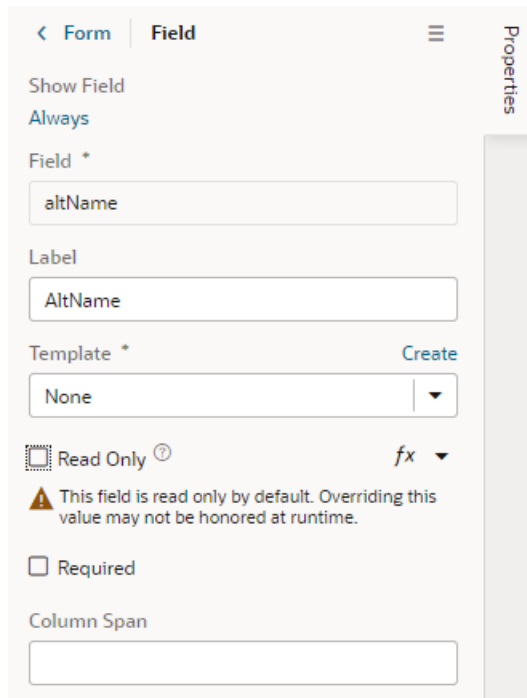
3. Edit the field's Read Only and Required properties in the Properties pane.

If you are editing the properties of a field defined by the Oracle Cloud Application service, the image below shows the message you'll see in the Properties pane if

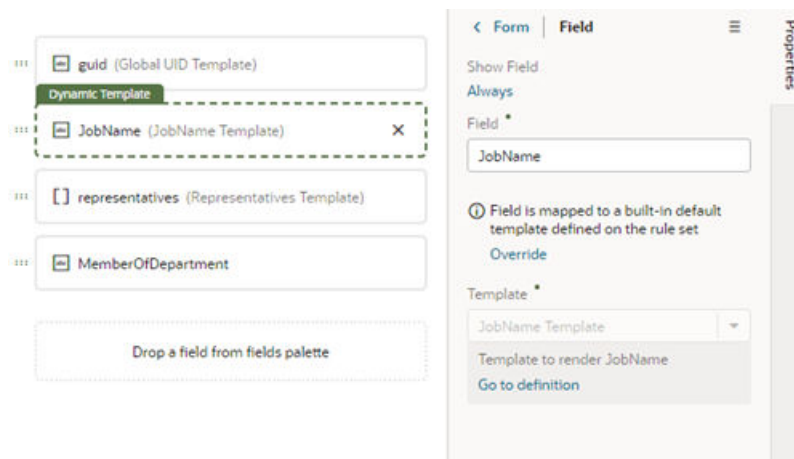
the field has undergone a custom transformation, which may override any changes you make to the field's properties.



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](#).



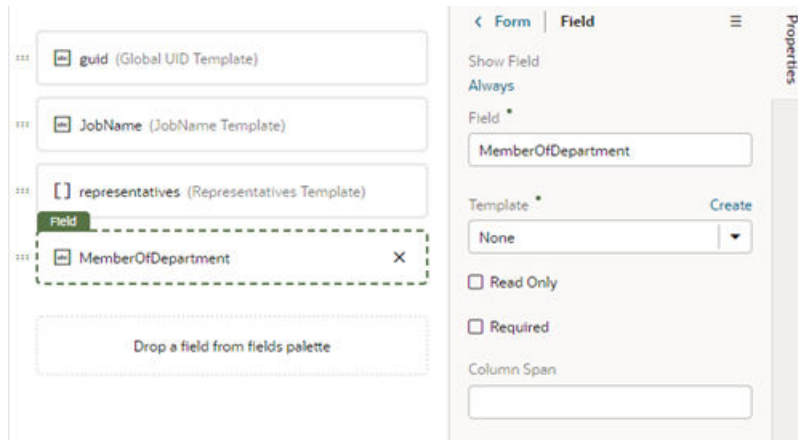
## Use Conditions to Hide and Show Fields in a Layout

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 Set editor, open the layout and select the field in the center pane.

When you select the field, you can see the field's properties in the Properties pane. By default, the Show Field property is set to `Always`, so the field is always displayed.



2. In the Properties pane, click **Always** under the Show Field property to open the Edit Show Field Condition dialog box.
3. Define the field's conditions by selecting attributes, operators and values in the condition builder in the dialog box. Click **Save**.

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.

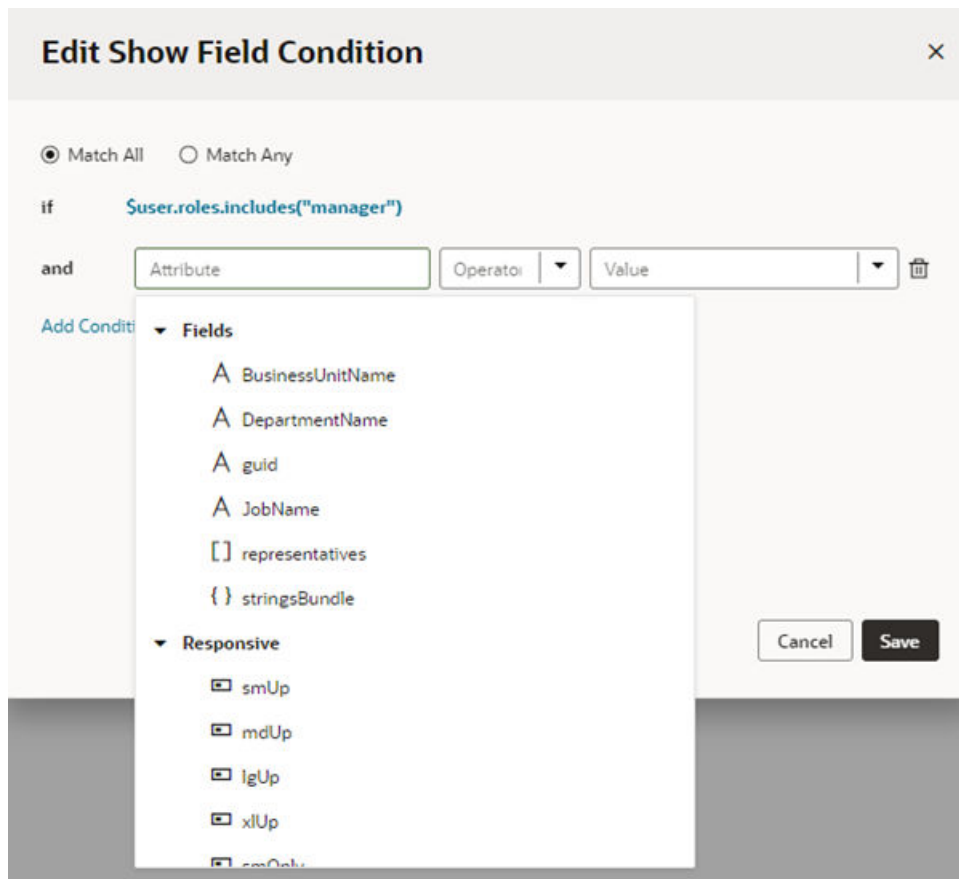
 **Note:**

For each field, regardless of type, you can choose `$numberValue` (for example, `$fields.ConflictId.numberValue()`) or `$value` (`$fields.ConflictId.value()`). You should use `$numberValue` when you know the field's value should contain a number. For example, if the `ConflictId` field's type is a string and you choose `$numberValue`, the field's value will be converted to a number, if possible. If the value can't be converted, the `$numberValue` will be `NaN` (Not a Number). The only limitation is that `$numberValue` is limited by the maximum precision allowed by the `Number` type in Javascript.

- *\$responsive* variables determined by the screen size of the device the app is currently displayed on. For example, the `responsive.mdUp` variable's value is `True` if the current user is using a device where the screen width is 768 pixels or more, such as a tablet. Look for these variables under the **Responsive** group in the condition builder.
- *\$user* variables determined by the current user. For example, the `user.isAuthenticated` variable's value is `True` if the current user is an authenticated user. You can use the `user.roles` variable to check the role of the user using the app. Look for these variables under the **User** group in the condition builder.

 **Note:**

When using `user.roles`, the Value drop-down lists the available Oracle Applications Cloud job and abstract roles. (The drop-down will not list any duty roles. If you want to specify a duty role, you can manually type the duty role name in the Value field.)



## Set How User Assistance is Rendered in a Layout

You use the User Assistance Density property to set how a field's user assistance text such as messages, help text and hints are displayed in the form.

To edit a field's User Assistance Density property in a layout:

1. In the Rule Set editor, open the layout and select the field in the center pane.
2. Select the field's User Assistance Density property from the dropdown list in the Properties pane.

The screenshot shows the 'Field' properties panel in Oracle APEX. The 'User Assistance Density' dropdown menu is highlighted with a red box. The dropdown is open, showing three options: 'Compact', 'Efficient', and 'Reflow'. The current selection is 'Efficient'. Other visible properties include 'Show Field' (Always), 'ID' (email), 'Label Hint' (email), 'Template' (None), 'Read Only' (unchecked), 'Required' (unchecked), and 'Column Span' (empty).

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 assistance 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:



The screenshot shows a form titled "main create" with a decorative header. It contains four input fields: "Name", "Job title", "Last Updated By", and "Id \* ?". The "Last Updated By" field is highlighted with a blue border and contains a vertical cursor. Below it, the text "Do not update" is visible. At the bottom of the form is a "Save" button.

In this form, the User Assistance Density property for the fields are set to different values:

- the Name field is set to `efficient`,
- the Job Title field is set to `reflow`,
- the Last Updated By field is set to `reflow`, and
- the Id field is set to `compact`.

You can see that both the Job Title and Last Updated By fields are set to `reflow`, but that the space below the Last Updated By field, where the insert cursor is) has expanded so that the user assistance text can be rendered below it.

## 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. 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 Set editor, open the dynamic table's layout
2. 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.

< Table | Field

Show Field  
Always

ID \*  
year

Label Hint  
Year

Template Create  
None

Read Only

Frozen Edge

Sortable

Width  
33%

Minimum Width  
33%

Maximum Width  
33%

Properties

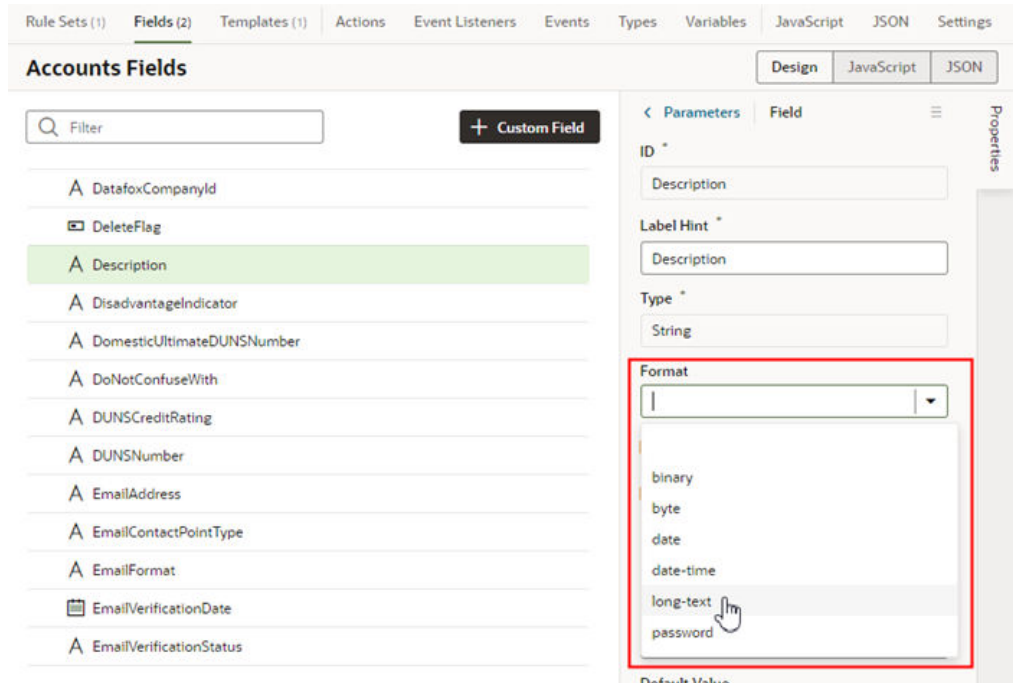
For fields in a dynamic table's layout, the Properties pane contains some display properties to control how the field is rendered in the table:

- **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:
  - In the layout's **Fields** tab, select the field you want to work with.
  - Set the **Format** property in the Properties pane to `long-text`.



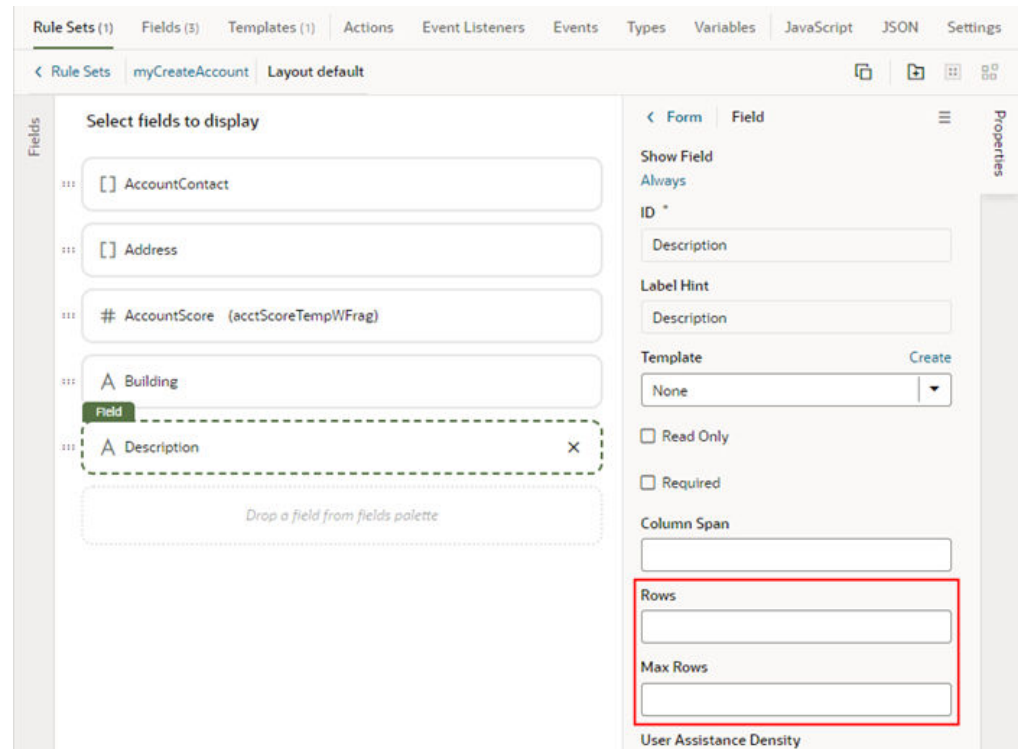
Notice that the Format property now has a vertical gray bar next to it to indicate the property has been modified.

When you switch the format to `long-text`, two additional properties are displayed in the Properties pane.

- Set the **Rows** property to the number of rows to display in the form by default.
- 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.

The screenshot shows the 'Properties' panel for a field in Oracle APEX. The panel is titled 'Parameters' and 'Field'. It contains several input fields: 'ID' with 'Description', 'Label Hint' with 'Description', 'Type' with 'String', and 'Format' with 'long-text'. There are also checkboxes for 'Read Only' and 'Required'. A red box highlights the 'Rows' and 'Max Rows' input fields. Below them is a 'Help Hint Definition' input field.

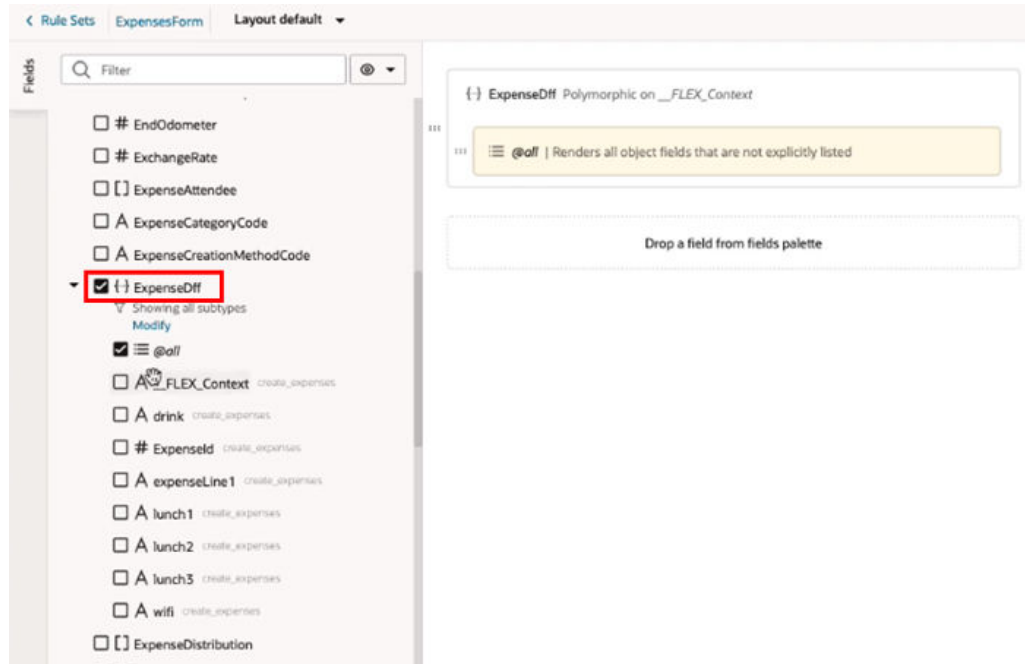
- To set a field to display as a text area in a particular form layout:
  1. In the Rule Set editor, open the layout and select the field in the center pane.
  2. Set the **Rows** property to the number of rows to display in the form by default.
  3. Set the **Max Rows** property to the maximum number 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.



## Work with Polymorphic Objects in a Layout

When you are adding fields to your layout, the Fields palette might include *polymorphic* objects containing fields you can select. Polymorphic objects are defined by the service that your app 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.

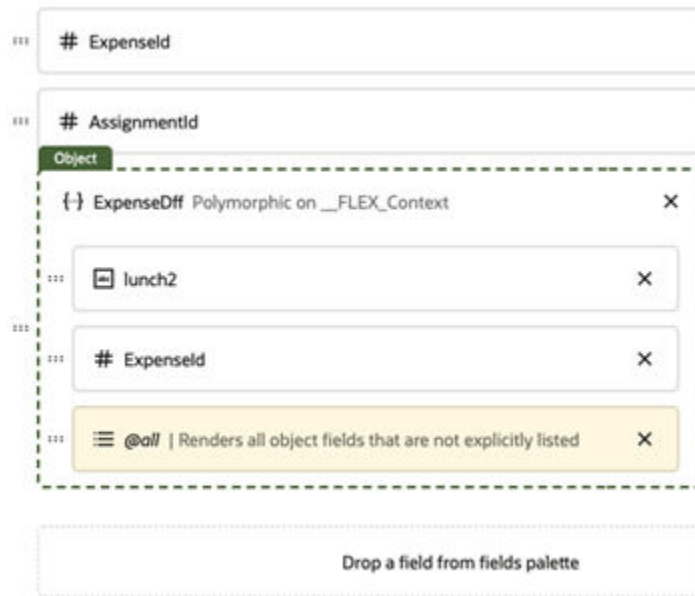
In the image below, `ExpenseDff` is a polymorphic object, and `__FLEX_Context` is the discriminator field.



An @all 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 @all 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 (`ExpenseId`, `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`, `ExpenseId`), 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:


1. **lunch2** (added and positioned explicitly)
2. **ExpenseId** (added and positioned explicitly)
3. **expenseLine1** (added using `@all`)
4. **lunch1** (added using `@all`)
5. **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` sub-type is applied to the discriminator.

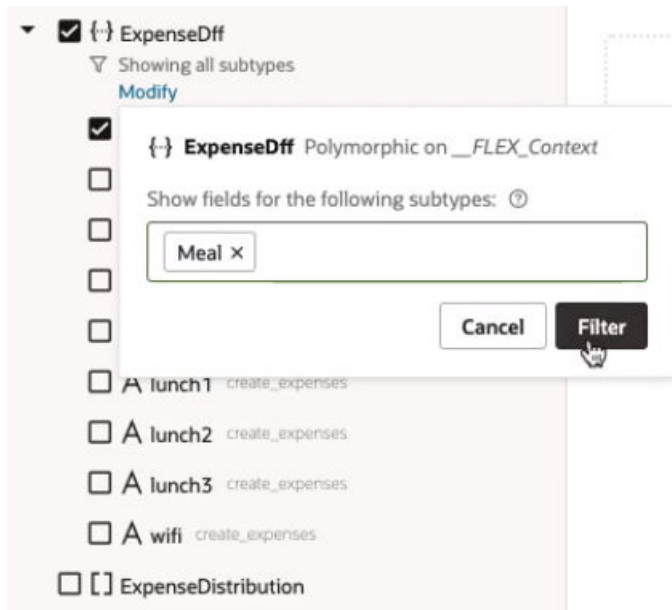
To add a polymorphic object to your layout:

1. In the Rule Set 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.

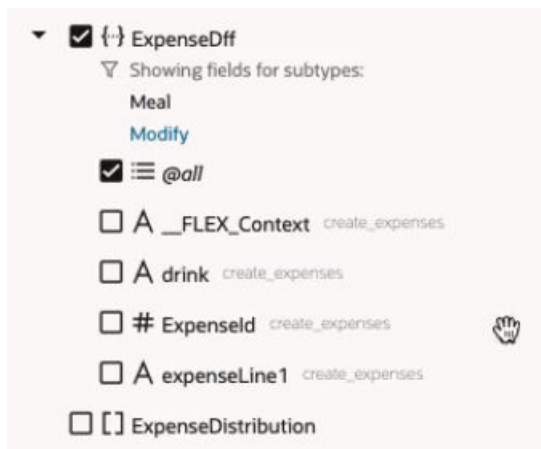
The Fields palette lists all the fields and objects you can add to your layout, including polymorphic objects. Polymorphic objects in the palette have a special object icon (  ). You can expand the object node in the palette to see the sub-fields that could potentially be displayed if you added the object 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 `ExpenseDff` are filtered by the `Meal` sub-type. Instead of listing all the object's sub-fields, only the sub-fields defined by the `Meal` sub-type (`drink`, `ExpenseId`, `expenseLine1`) are listed under the object node. These are the fields that will be displayed when the `@all` field is added to the center pane and the `Meal` 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.

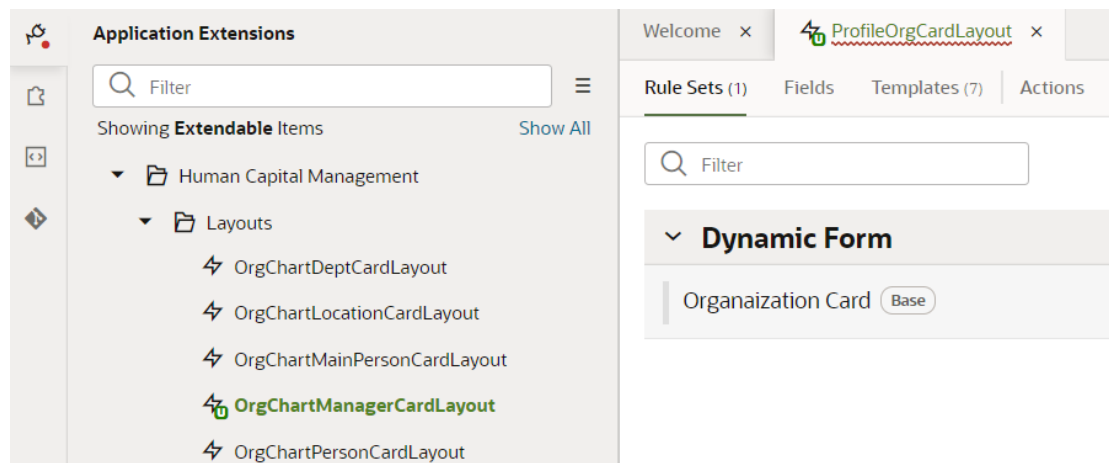


## Determine What's Displayed at Runtime

You control what's displayed at runtime on a page through the use of *display logic*, which you configure on the Rule Sets tab. Suppose you want to configure a dynamic table so that certain columns are hidden and others are added when the user viewing the page is in Canada. You'd then create a rule that checks for the user's location and, if the user is indeed in Canada, apply the layout you've created that contains the desired columns. All non-Canadians would see the page with the default layout applied.

You can have more than one rule for a given component, and the rules are listed in a display logic tree when you select a 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 in Canada—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.

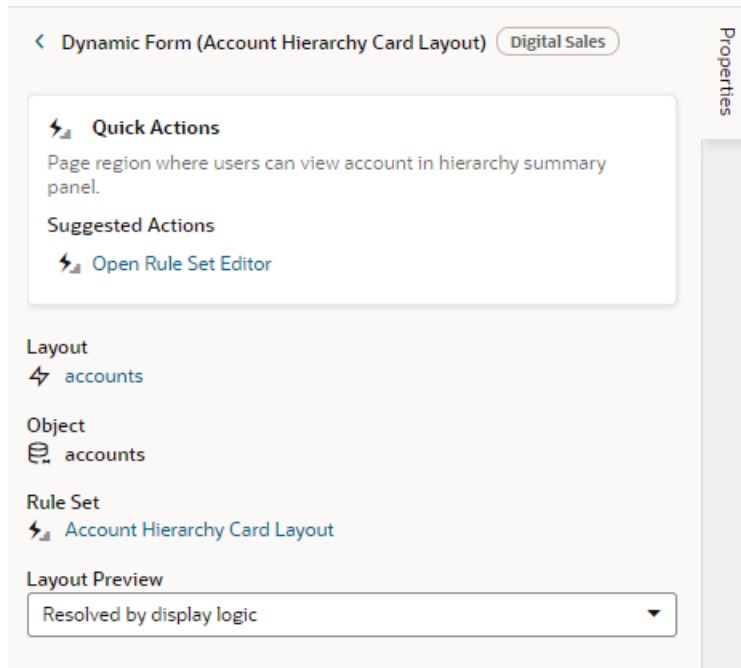
When you select a layout in the Application Extensions pane, then open the layout's Rule Sets tab, you'll see a list of the rule sets defined in that layout. In this example, you can see that the layout called `ProfileOrgCardLayout` uses one dynamic form. You can also see that the name of the rule set governing the dynamic form's display is called `Organization Card`:



You would see more names in the Rule Sets tab if the layout used other dynamic forms or table. You click a rule set's name to open it in the Rule Set editor.

To set the display logic in a rule set:

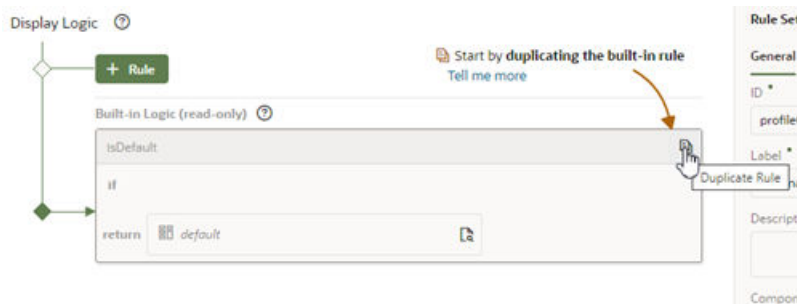
1. When your page is open in the Page Designer, click the dynamic form or table you want to work with in the canvas area, or select it in the Properties pane.
2. Click **Open Rule Set Editor** in the Properties pane for the form or table .



In this image of the Properties pane, you can see the name of the dynamic component's rule set (Account Hierarchy Card Layout).

3. In the Rule Set editor, create a rule by clicking and giving it a name, preferably something meaningful. For example, to create a layout that displays only when the user is in Canada, you might call the rule `inCanada`.

The rule set for a dynamic component always contains a default rule that is set by the base application. 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.


4. 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 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. 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.

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 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. Context variables include:

- *\$fields* variables determined by the fields displayed in the Fields editor. For example, the `$fields.firstName.value` lets you access the value of the First Name field in your data source. Look for these variables under the **Fields** group in the condition builder.

 **Note:**

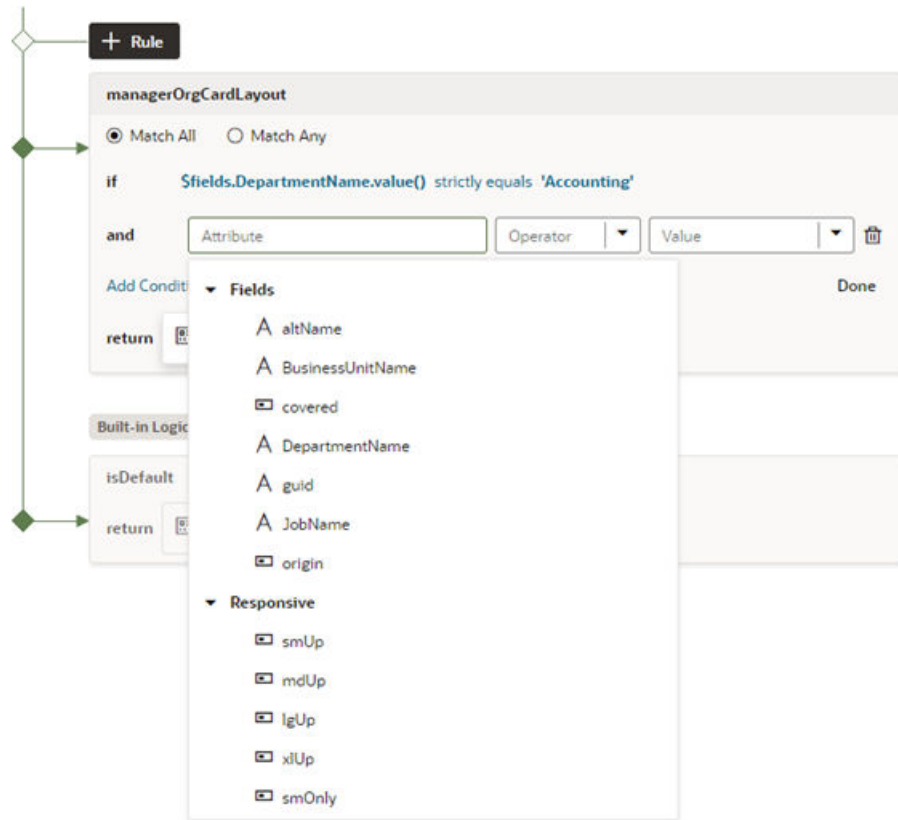
For each field, regardless of type, you can choose `$numberValue` (for example, `$fields.ConflictId.numberValue()`) or `$value` (`$fields.ConflictId.value()`). You should use `$numberValue` when you know the field's value should contain a number. For example, if the `ConflictId` field's type is a string and you choose `$numberValue`, the field's value will be converted to a number, if possible. If the value can't be converted, the `$numberValue` will be `NaN` (Not a Number).

The only limitation is that `$numberValue` is limited by the maximum precision allowed by the Number type in Javascript.

- *\$responsive* variables determined by the screen size of the device the app is currently displayed on. For example, the `responsive.mdUp` variable's value is `True` if the current user is using a device where the screen width is 768 pixels or more, such as a tablet. Look for these variables under the **Responsive** group in the condition builder.
- *\$user* variables determined by the current user. For example, the `user.isAuthenticated` variable's value is `True` if the current user is an authenticated user. You can use the `user.roles` variable to check the role of the user using the app. Look for these variables under the **User** group in the condition builder.

 **Note:**

When using `user.roles`, the Value drop-down lists the available Oracle Applications Cloud job and abstract roles. (The drop-down will not list any duty roles. If you want to specify a duty role, you can manually type the duty role name in the Value field.)

Display Logic 


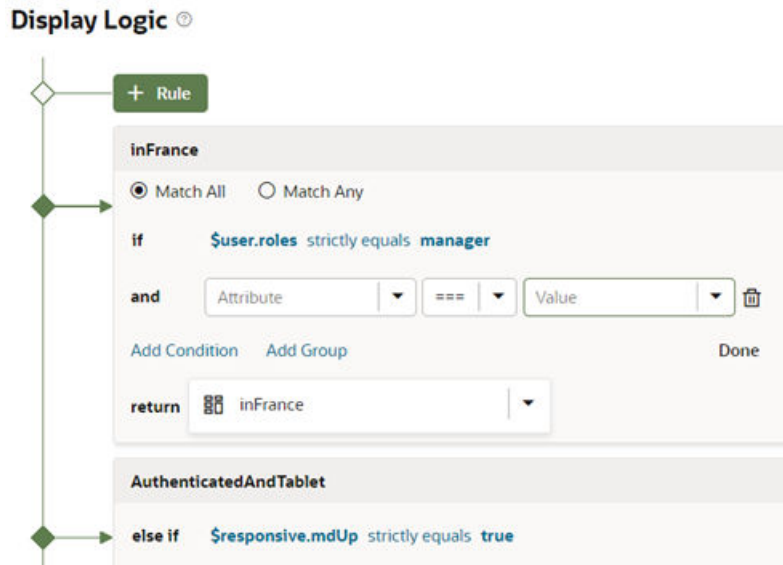
The screenshot shows the 'Display Logic' configuration for a rule named 'managerOrgCardLayout'. The rule is set to 'Match All'. The main condition is an 'if' statement: 'Sfields.DepartmentName.value() strictly equals 'Accounting''. Below this is an 'and' condition with a table for adding conditions. A 'Fields' palette is open, showing a list of fields: altName, BusinessUnitName, covered, DepartmentName, guid, JobName, origin, and a 'Responsive' section with smUp, mdUp, lgUp, xlUp, and smOnly.

 **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**.

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 is in Canada AND has the manager role. You would then create a rule with two conditions, and select **Match All** to require that both conditions are true.

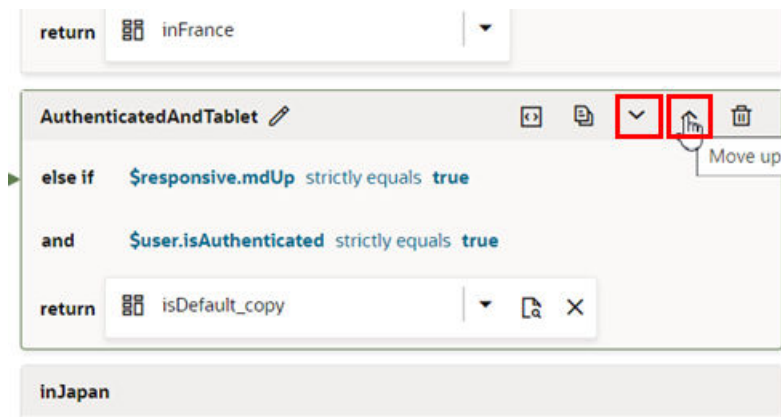


If you select **Match Any**, then the rule will be evaluated as true if any of the conditions in the rule are true.

5. 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.

6. Click **Rule** to add another rule.
7. Use the **Move Up** and **Move Down** buttons to make sure you have the rules in the order you want them evaluated.



The order and precision of your rules is important. The rules are evaluated from the top down, so the first rule evaluated as true will determine the layout that is used. When configuring the display logic, it's not a problem if there are rules that will never be used or evaluated.

## Responsive App Display Logic Example

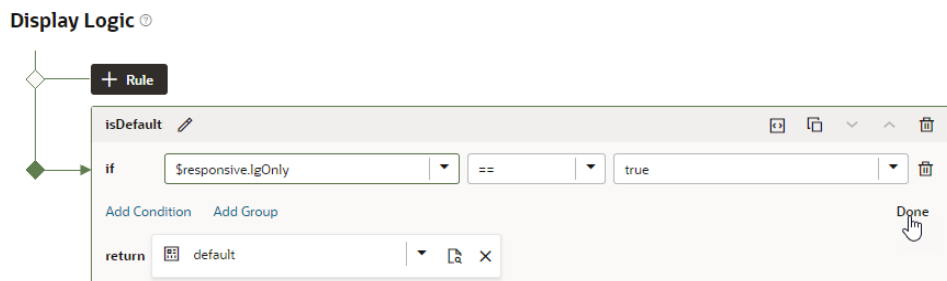
The following example shows how to configure display logic for responsive apps. Suppose you want a dynamic component that shows different fields based on the device's screen size,


say, small, medium, and large screens. You'd then create a rule that checks the current user's device screen size and applies the layout that contains the desired fields for that screen size.

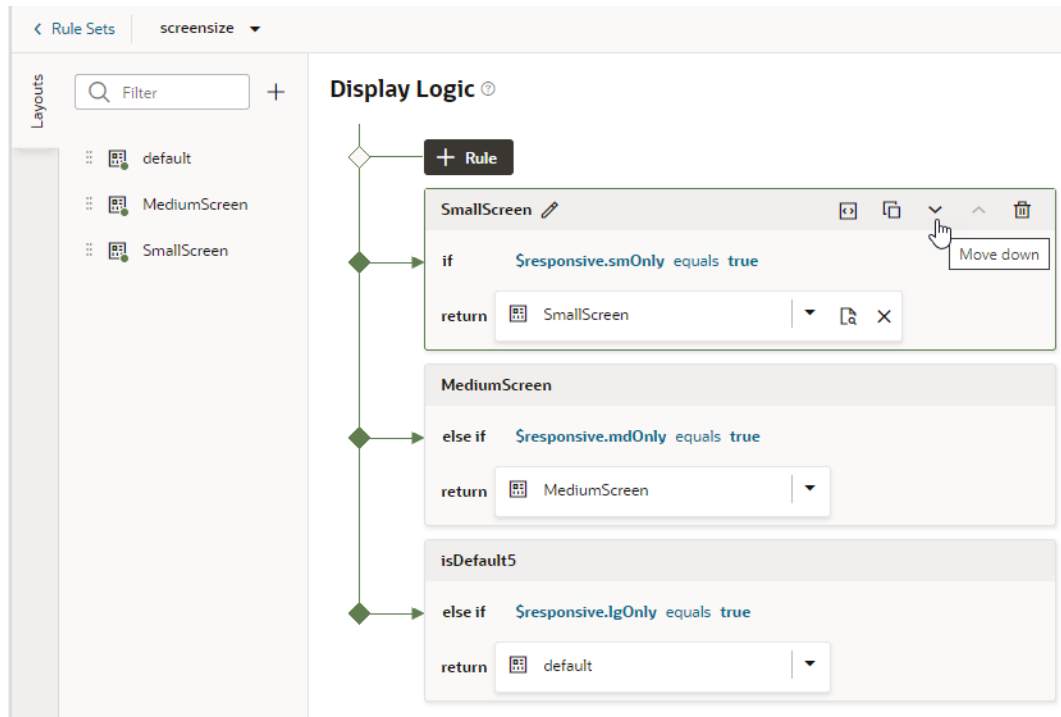
To illustrate, consider a dynamic form that displays the following employee fields in the default layout: Id, Name, Department, Email, and Hire Date. If the user's device screen is small, you might want the page to show a particular layout (say, the `SmallScreen` layout) with only the Name and Email fields. If the user's device screen is medium, you might want the page to show another layout (for example, the `MediumScreen` layout) with the Name, Department, and Email fields. If the user's device screen is large, you might show the `default` layout.

To configure a rule set for responsive logic:

1. Update the default rule to show the default layout when the device's screen size is large:
  - a. In the Rule Set's Display Logic section, click **Click to add condition**.
  - b. Choose `lgOnly` under **Responsive** in the Attributes drop-down list, select `===` from the Operators list, then remove one of the equal signs, and select `true` as the Value.
  - c. Click **Done**.



2. Duplicate the existing rule as required and use it as the basis to create more rules, in our case, the `MediumScreen` and `SmallScreen` rules. During this step, you have the option of creating copies of a particular layout which you can then update to show the fields you want when the device screen is small and when it is medium.
  - a. Click the Duplicate icon () , then enter a name for the new rule in the Duplicate Rule dialog box.  
To also create a copy of the layout to use as a starting point, make sure that check box is selected. Click **Duplicate**.
  - b. Edit the new rule and define its conditions. To continue our example, you might use the `mdOnly` attribute to show the `MediumScreen` layout when the current user's screen size is medium and the `smOnly` attribute to show the `SmallScreen` layout when the current user's screen size is small.
  - c. As part of configuring the new rules, click the newly created layouts in the **Layouts** tab (`MediumScreen` and `SmallScreen`), then select the fields you want to show when the device screen is small (Name and Email) and when it is medium (Name, Department, and Email).



3. Use the **Move Up** and **Move Down** buttons to make sure you have the rules in the order you want them evaluated.
4. Test your application using different screen sizes in the Page Designer toolbar. For example, use iPhone to test the display logic on a small screen, iPad to test a medium-sized screen's display, and desktop to test a large screen's display.

## Use Application and Page Parameters in a Layout

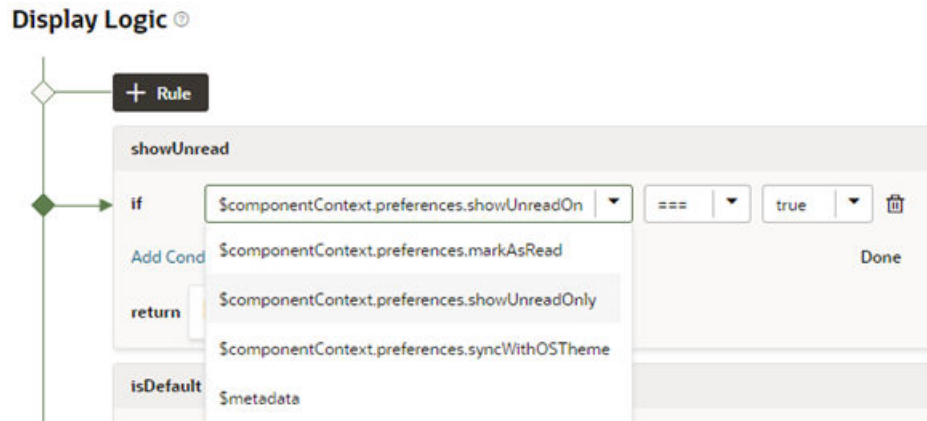
When using a layout on your page, you can pass the layout a context, including:

- values produced by your application that come from outside your layouts, such as page variables,
- details from other parts of the application,
- other values that might be useful when extending the application.

The context you pass to the layout must adhere to a contract defined by the base layout developer at Oracle. The contract specifies the valid context parameters that can be passed to the layout.

If any of these parameters are exposed in your app extension, you can access them in the `$componentContext` in a dynamic component's Expression Editor and in the rule set condition builder. The parameters in a component's `$componentContext` are available in all the rule sets in the component.

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.



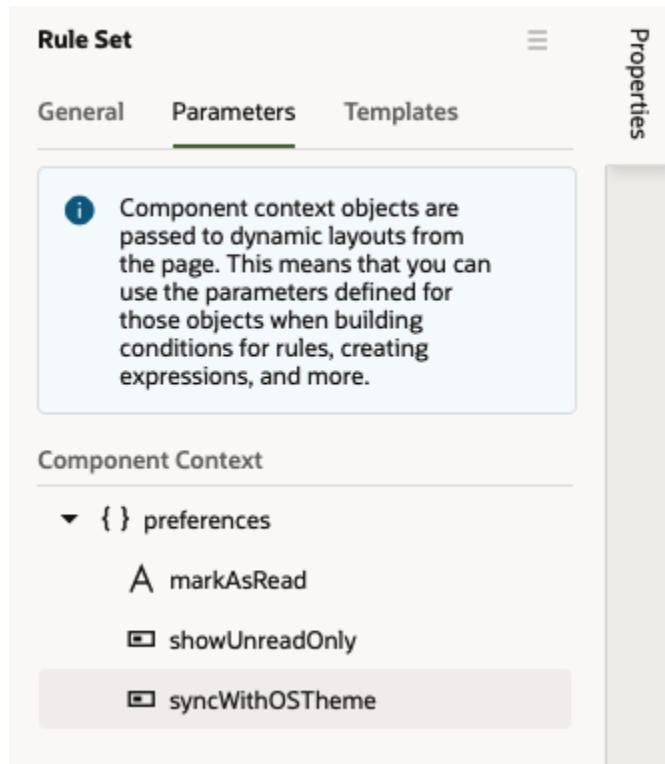
You can see the list of valid `$componentContext` parameters in the Parameters tab in the Properties panes of rule sets and in a layout's Fields editor. The Parameters tab shows the name and type of each parameter. To see a parameter's description and the valid values, move your cursor over the tooltip icon which is displayed when you hover over the parameter name.

 **Note:**

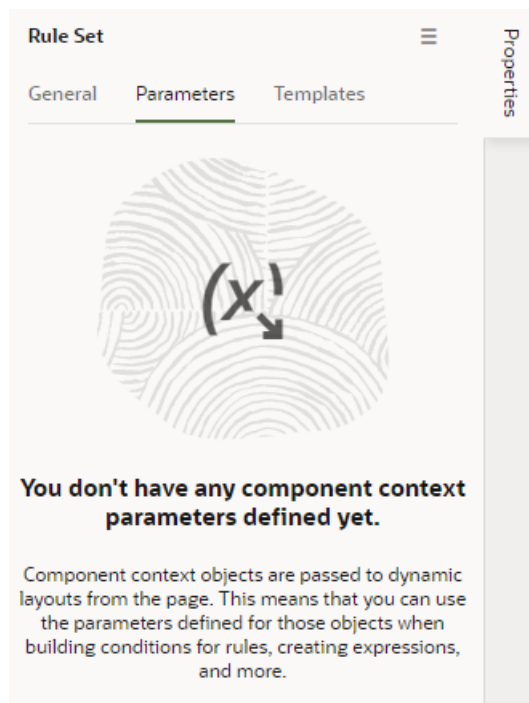
If you enter an invalid parameter value in the Expression editor or condition builder, a warning will be displayed in the layout's JSON file.

In a rule set's Properties pane, the Parameters tab lists the parameters in the component context that you can use in the rule set.





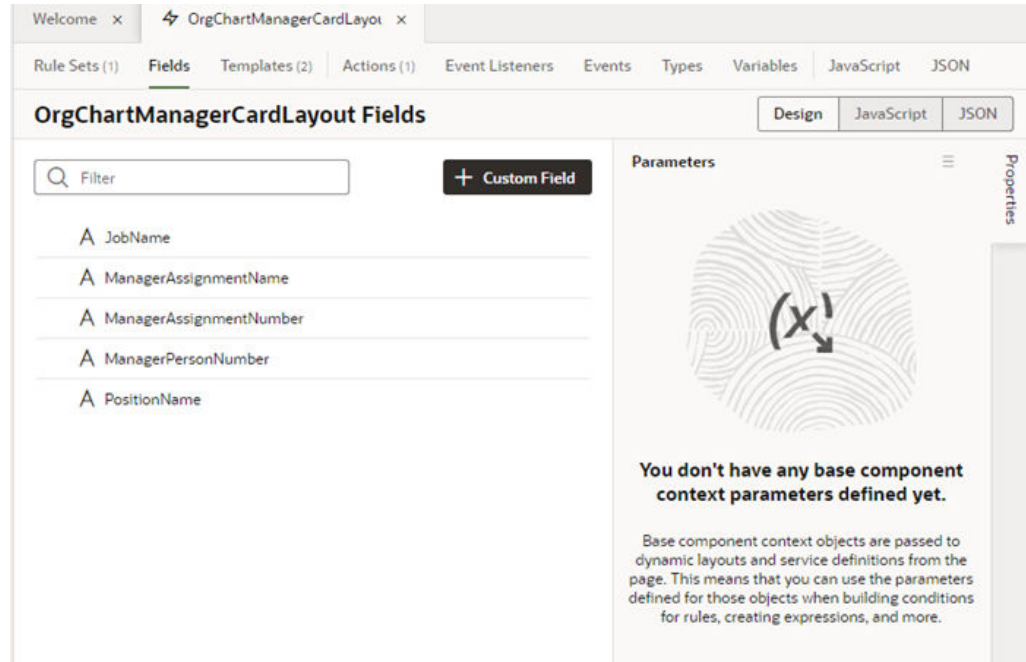
If you don't see any parameters in the Parameters tab, this means that the base app has not defined any `$componentContext` parameters for the component.



In a layout's Fields editor, the Parameters tab lists the parameters in the base component context that you can use in the layout. You can use parameters in the base component

context to pass information to your layouts as well as to the service definition, for example, to add parameters to the query used to call your Cloud Application service.

The Parameters tab is displayed in the Fields editor's Properties pane when no field is selected, and will list the available base component context parameters.



## Preview Different Dynamic Layouts

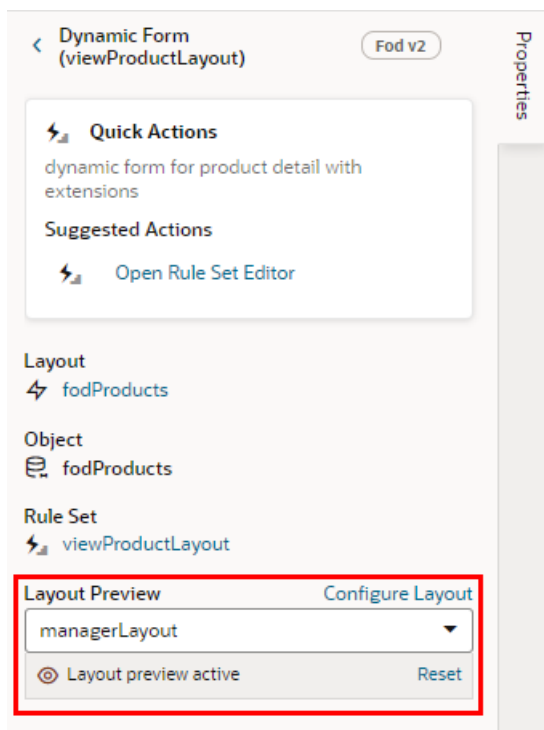
To preview how different layouts will look when applied to your page, use the Layout Preview in the Properties pane. Layout Preview forces the Page Designer to use the layout you select and ignore the rules in the rule set. For example, if you created a layout that only managers can see, you won't see it in the Page Designer if you're not logged in as a manager. You use Layout Preview in the Properties pane to override the display logic and render the page using the layout you select.

To preview a dynamic component layout:

1. Open the page in the Page Designer and select the dynamic table or form you want to work with.

The Layout Preview dropdown list displays all the layouts that have been defined for this component's rule set.

2. Select a layout in the Layout Preview dropdown list:



In this image, the inCanada layout is selected in Layout Preview and is being displayed in the dynamic form. To open the layout in the editor, click **Configure Layout**.

3. Click **Reset** to return to the default "Resolved by display logic" option.

## Using Field and Form Templates

You can customize how a dynamic form is rendered on the page by editing form layouts to group fields together and to apply templates to the layout and fields.

### Control How a Field is Rendered with Field Templates

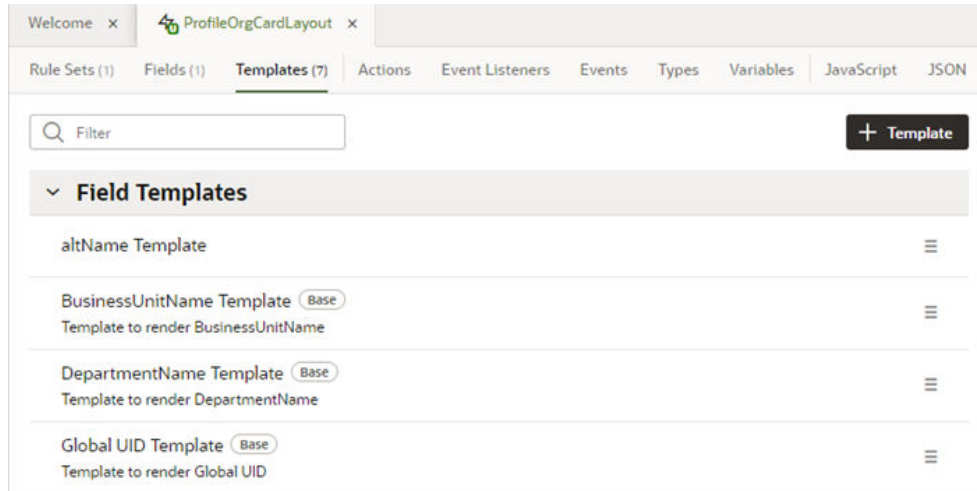
You can customize how a field is rendered in a layout for a dynamic form or table by applying a *field template*. A field template contains UI components, for example, text fields or images, and defines their properties, such as styling details. Components in a template can access the variables, constants, action chains and event listeners defined in the layout.

The base app 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 base app has applied a template called BoldType to the Update field. If you do nothing, 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.

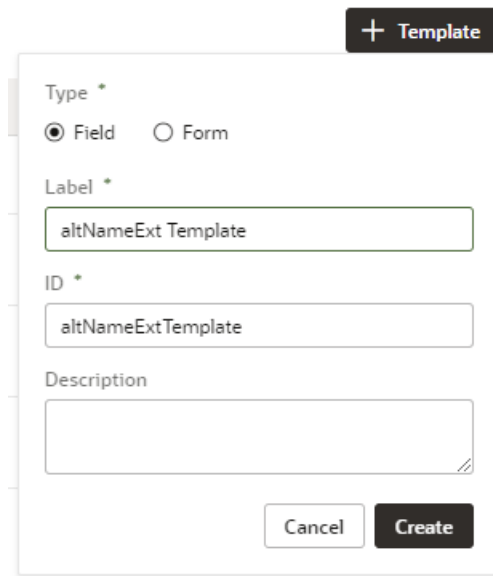
To create a field template for a field in a dynamic form:

1. Open the layout's Templates tab.

The Templates tab displays a list of field and form templates that are already defined for the artifact. Templates defined in the base app are marked with `base`.



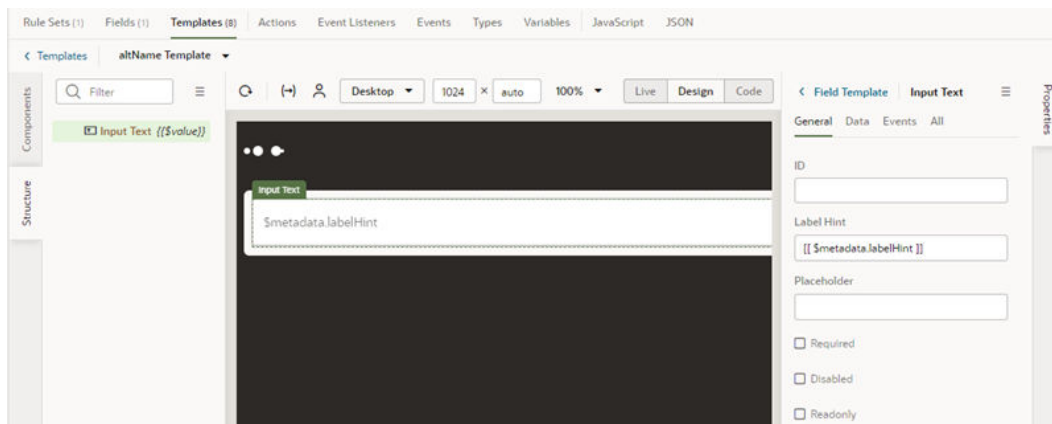
2. Click **+ Template** and select **Field** in the dialog.



3. Specify the Label and ID. Click **Create**.

The new field template opens in the template editor. The template editor contains a Components palette, Structure view, canvas and Properties pane.

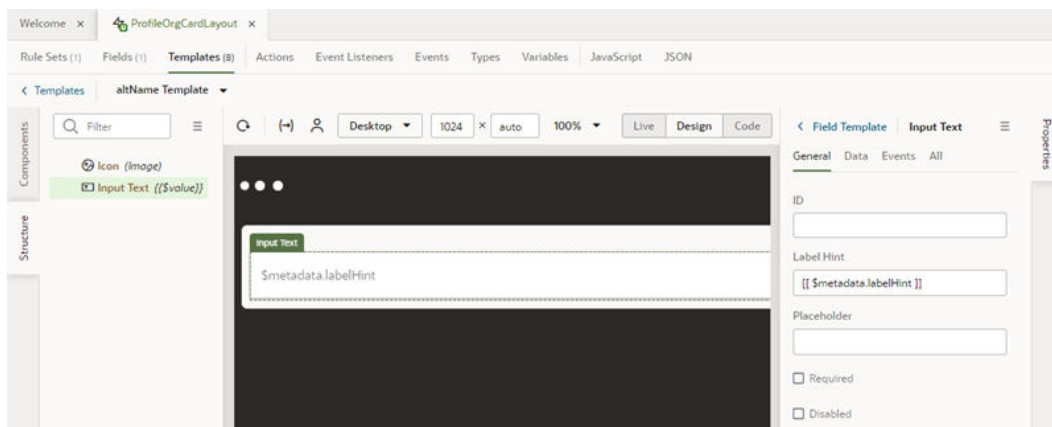
Your new field template has an Input Text component that is generated automatically. This is used to display the data and display name when you apply the template to a field in the layout.



4. 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.

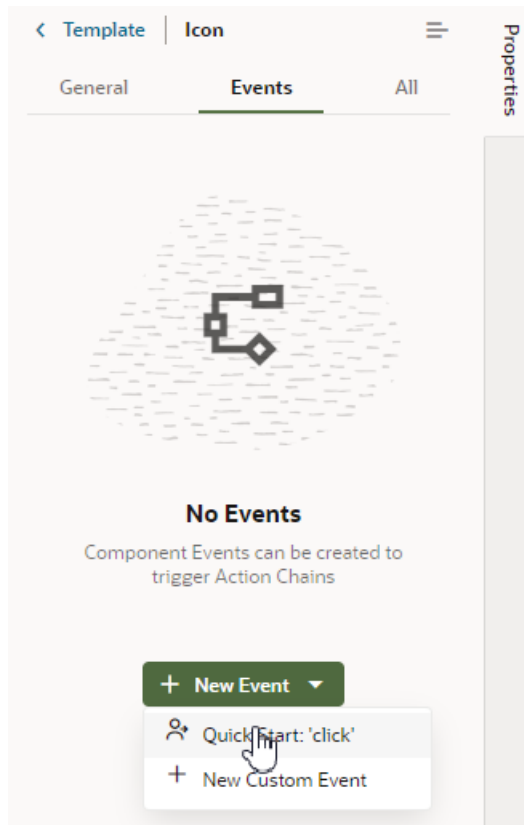
You can add more UI components above or below the Input Text component, or replace the Input Text component with a different one, for example, to render a field using a Rating Gauge component instead of an Input Text component.

In this image, you can see in the Structure view that the template contains an Icon component and an Input Text component:



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 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.

```
<template id="altNameTemplate">
  <span class="vb-icon vb-icon-image"></span>
  <oj-input-text value="{{ $value }}" label-
  hint="[[ $metadata.labelHint ]]"></oj-input-text>
</template>
```

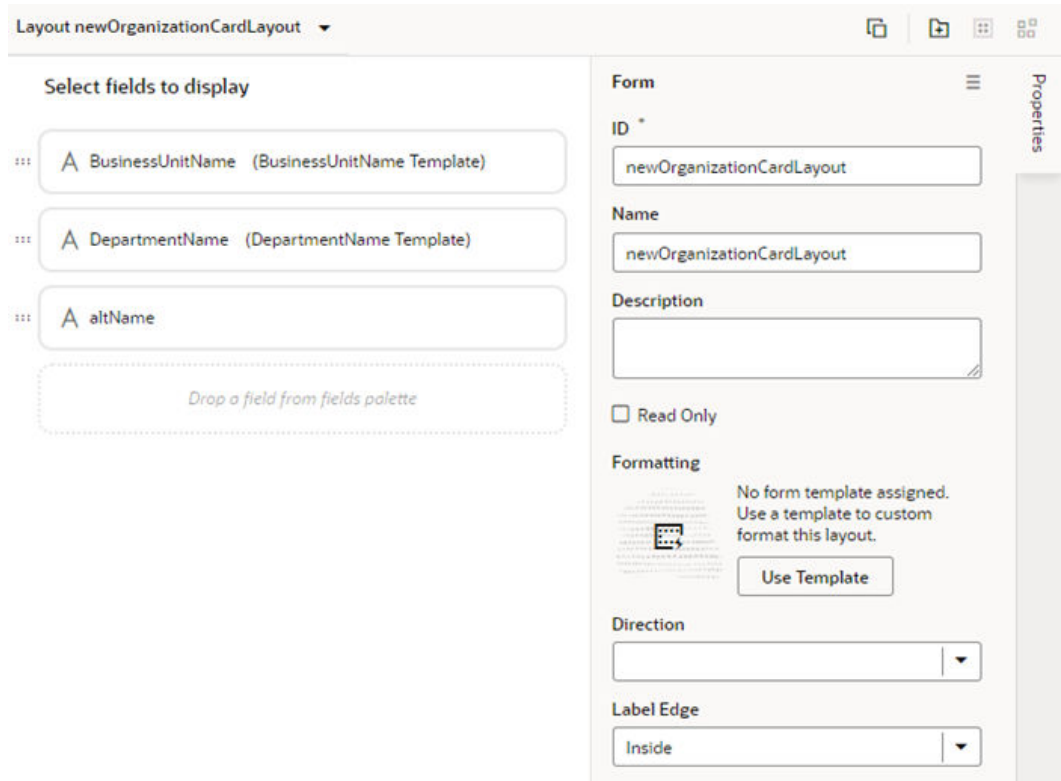
After creating the template, it's added to the list of field templates in the Templates tab. In the Templates tab, you can open and duplicate templates you've created.

## Apply a Template to a Field

To apply a template to a field in a layout:

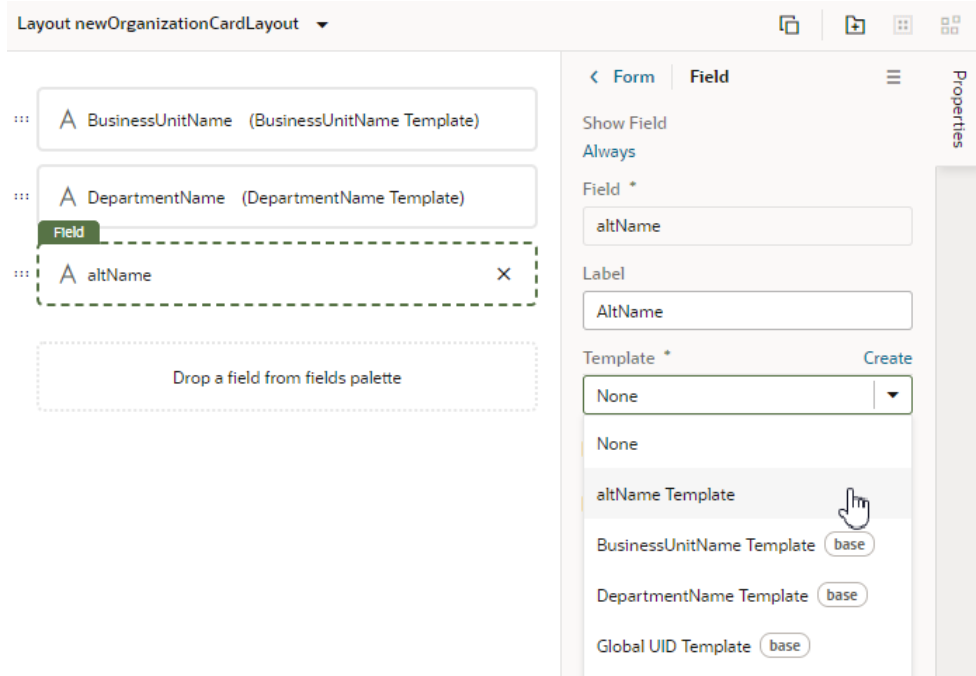
1. In the rule set editor, open the layout you want to work on.

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.

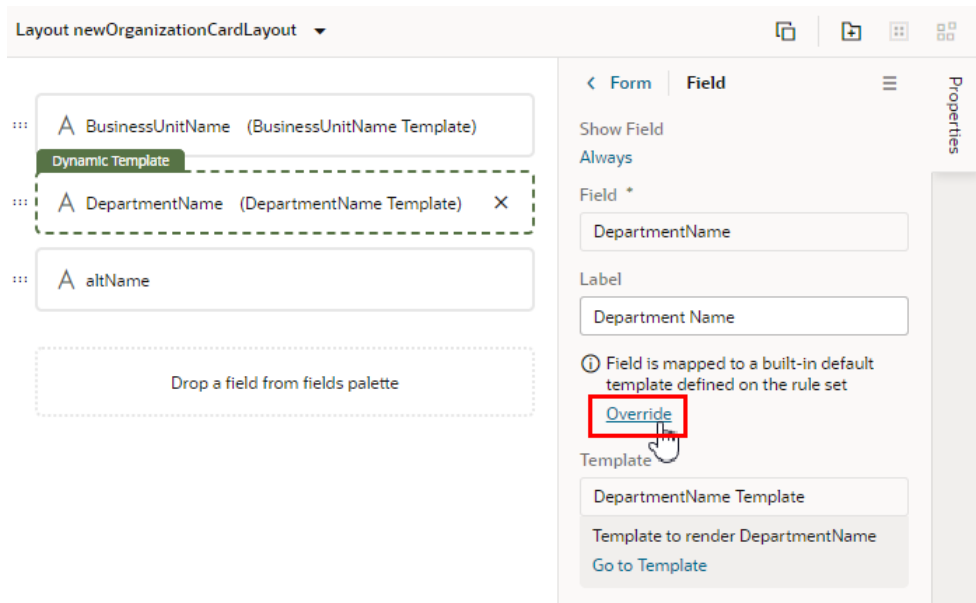


2. Select the field in the center pane.
3. Select a template from the Template dropdown list in the Properties pane.

If no template has been applied to the field (as in this image), you can select a template in the list. You can also click **Create** to create a template for the field in the template editor.

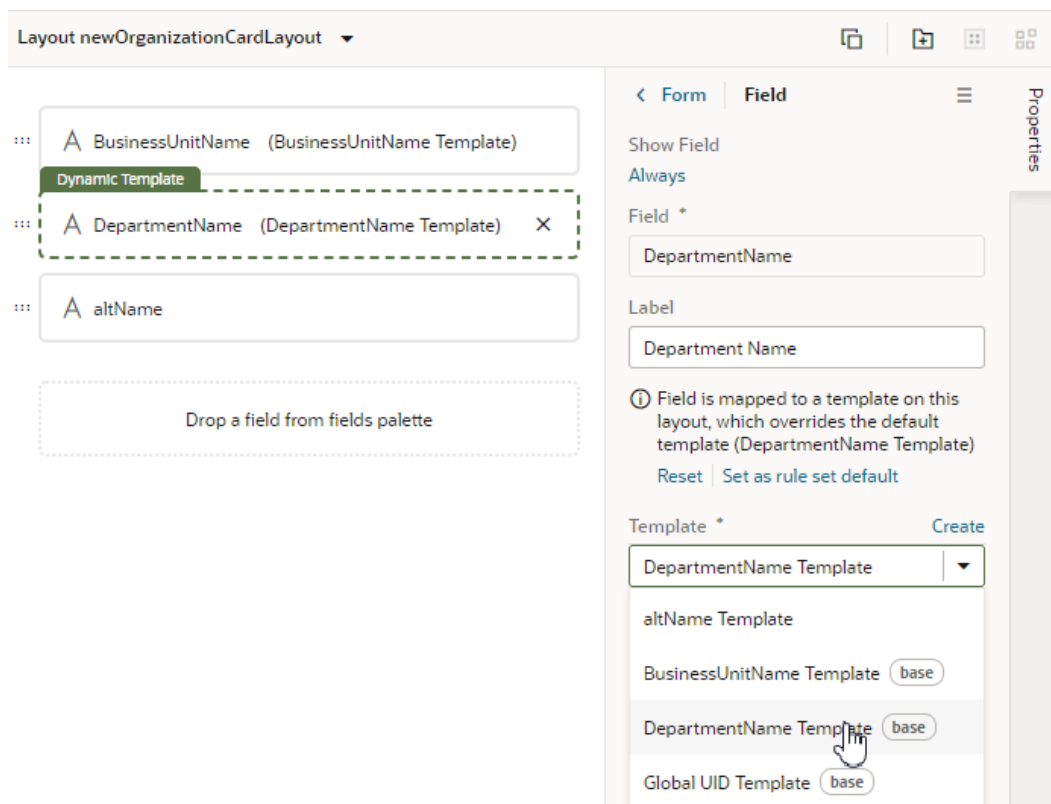


If the field already has a field template applied to it (you'll see the template name next to the field name), you can still select a template in the list unless a default template has been defined for the field. You'll see a notification in the Properties pane and an **Override** button if a default 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 set it in the rule set's Templates tab in the Properties pane.

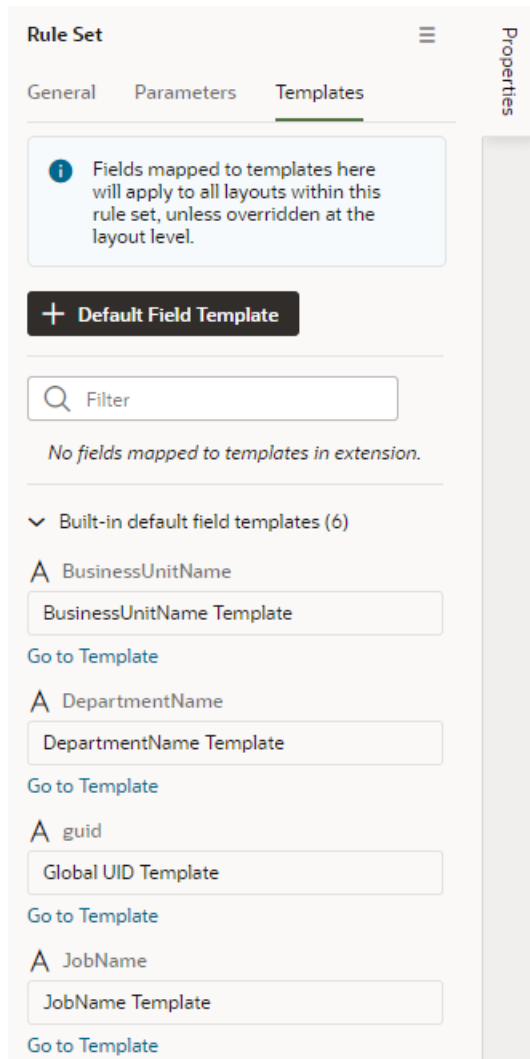
## Set a Default Field Template for a Rule Set

In addition to applying templates to fields in individual layouts, you can set the default field template for a rule set in the rule set's Properties pane. The default field template will be applied to the field in every layout in the rule set. After a default field template is applied to a field, you'll need to override it if you want to change the field's template in an individual layout.

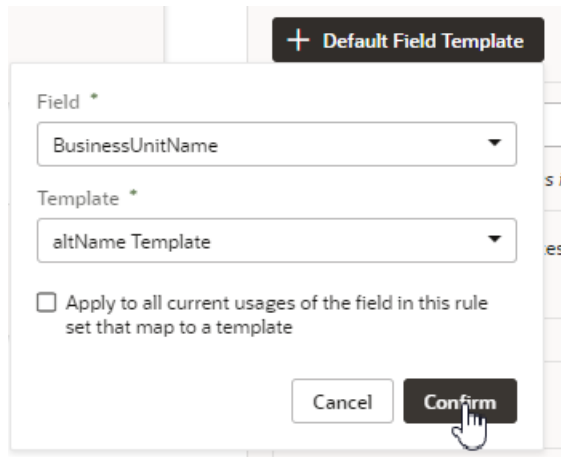
To define a default field template for a field in a rule set:


1. Open a rule set in the editor, then open the Templates tab in its Properties pane.

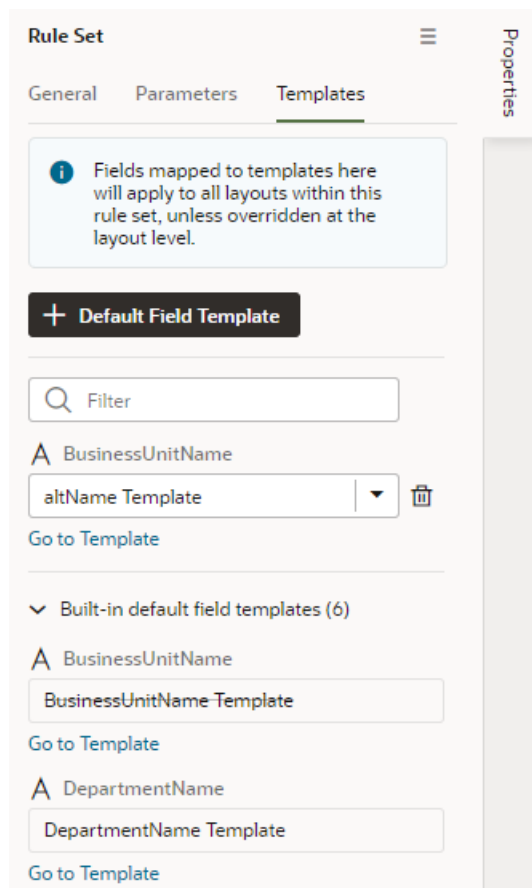
The Templates tab lists all the fields in the layout that have a default template applied to them. A field's default template might be defined in the base app (Built-in default field templates) or in the extension.



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 in the base app 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  next to the mapping in the list.



## Start an Action Chain from a Field

You can start an action chain when an event occurs in a field by adding a component event to the field in a field template. For example, you might want to display some additional details or options when someone changes the value in one of your form fields. You can add an event that's triggered when the value changes, and start an action chain that retrieves the data and displays it in your page. The Quick Start option in the component's Properties pane can help you quickly create the event, event listener and action chain. You can also use the event to start action chains that are already defined in the layout.

When creating an action chain, you can use variables and constants defined in your layout, and create new ones if you need them.

To start an action chain from a field:

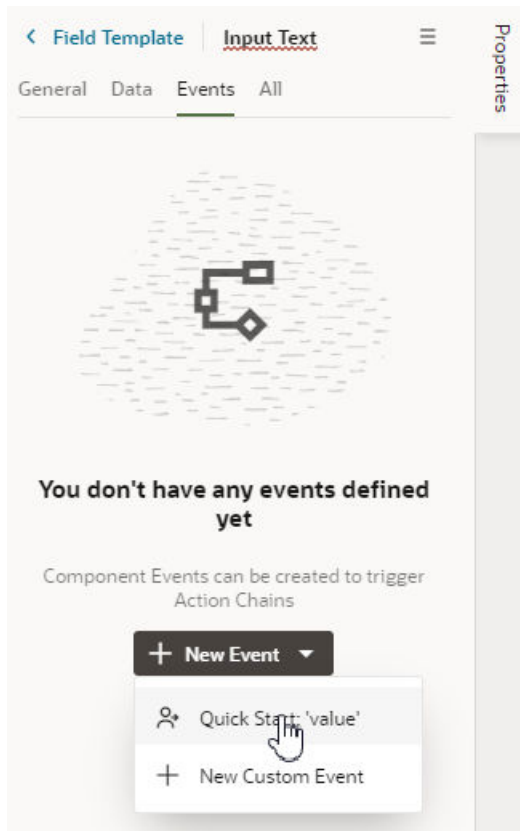
1. Open the **Templates** editor of your layout and click the field template you want to edit.

The field template opens in the Template editor.

2. Select the text field, then open the **Events** tab in the Properties pane.
3. Click **Add New Event** in the Events tab and select the Quick Start option in the drop-down list.

When you select the Quick Start option:

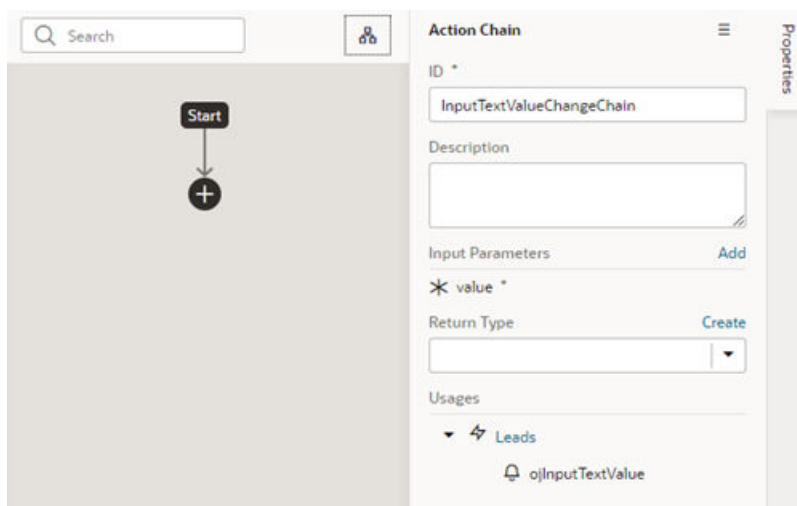
- an event is defined for the text field
- a new action chain is created
- an event listener is created that will trigger the new action chain when the event occurs.
- you are navigated to the new action chain in the Action Chain editor



The Quick Start defines the event suggested 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. You can also add more events to the text field.

4. In the Action Chain editor, define the action chain properties

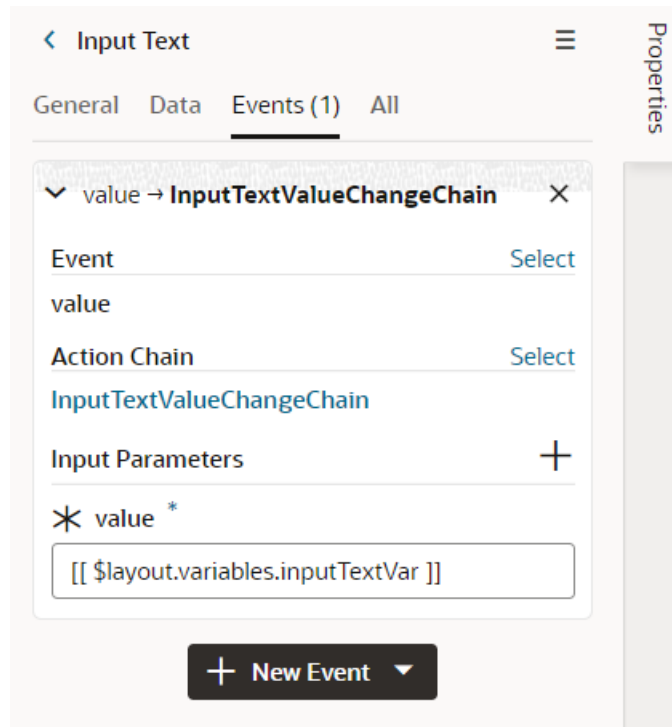
In the action chain's Properties pane, you can edit the default ID, add a description, and configure the action chain's input parameters and return type.



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 can 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.



## Video: Customize the Fields in Your Dynamic Layout

This video shows how to create field templates and how to create your own custom fields.



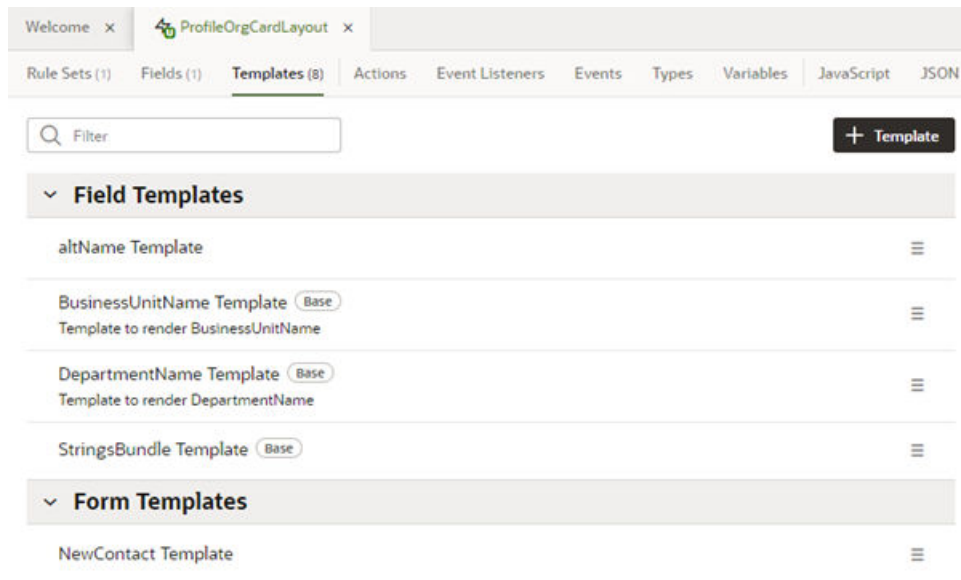
## Control How a Form Layout is Rendered

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. The template can be one of the built-in templates defined in the base app, 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 the same layout, but templates can't be shared between layouts.

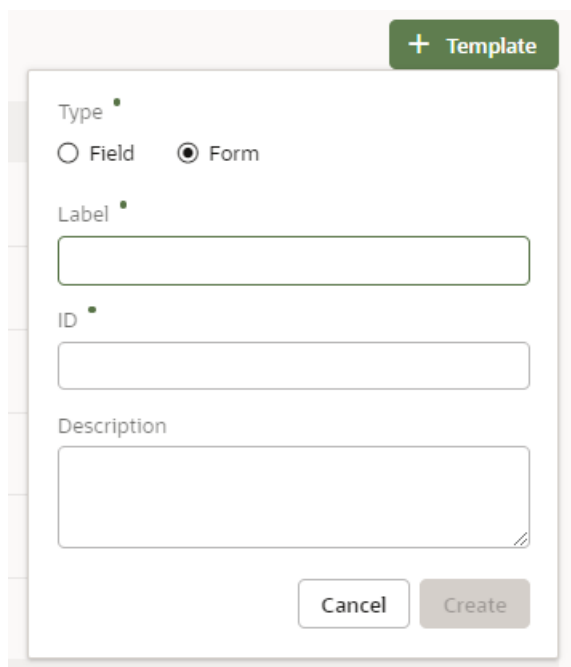
To create a form template for a dynamic form:

1. Open the layout's Templates tab.

The Templates tab displays a list of field and form templates that are already defined for the artifact. Templates defined in the base app are marked with `base`.



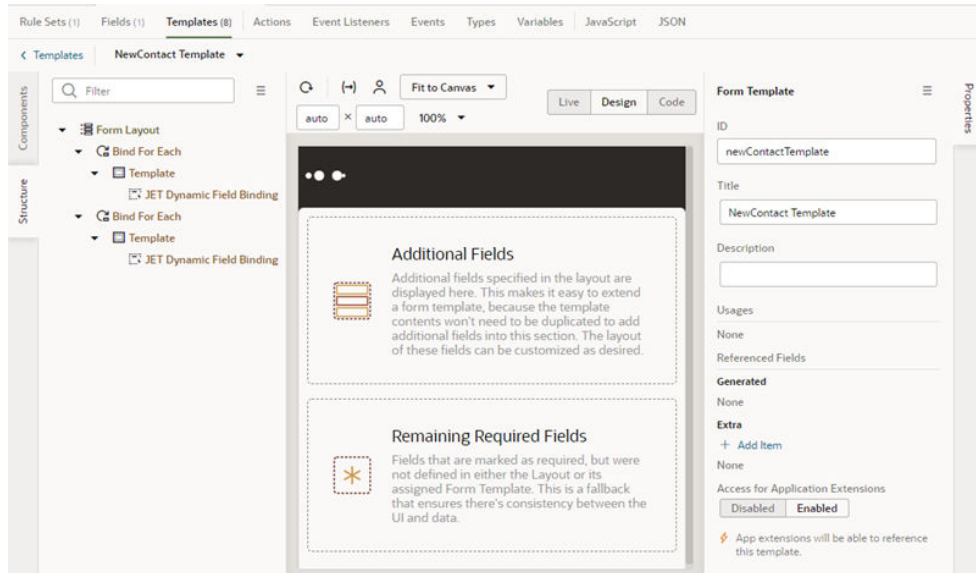
2. Click `+ Template` and select **Form** in the dialog.



3. Specify the Label and ID. Click **Create**.

The new template opens in the template designer. The template designer 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 to if you want it to appear in the form. This image shows the Properties pane after adding the `altName` field to the template as an Extra.



**Form Template** ☰ Properties

ID

Title

Description

Usages

None

Referenced Fields

Generated

None

Extra

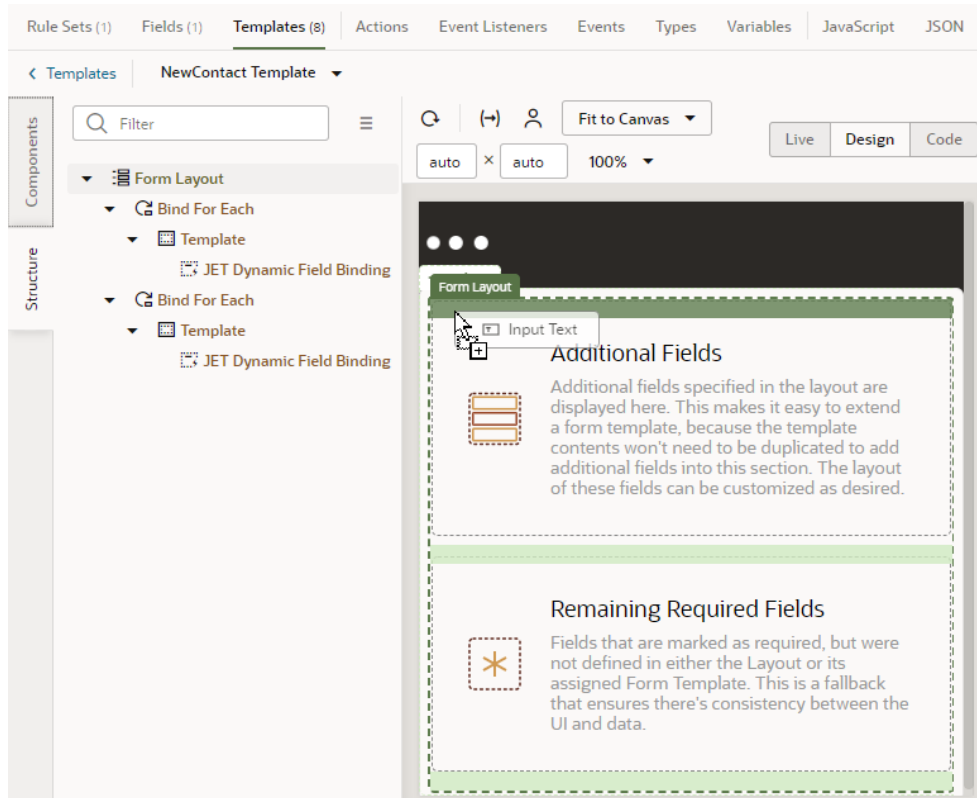
+ Add Item

A altName

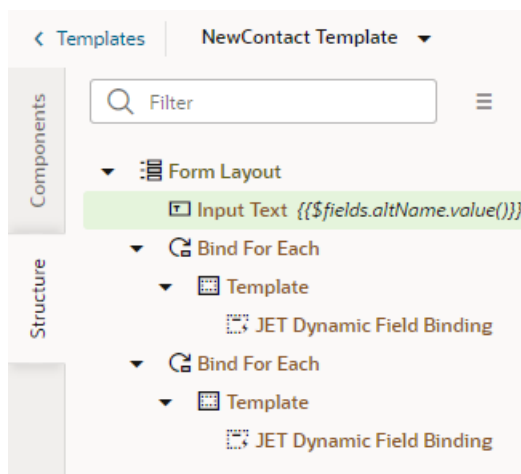
Access for Application Extensions

⚡ App extensions will be able to reference this template.

5. Drag the component you want to add from the Components palette and position it in the Structure view or on the canvas.

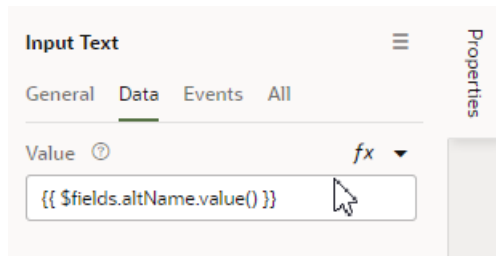


You can add components above and below the read-only template fields, but not within them. In the Structure view of this template, you can see a Input Text component that was positioned above the Additional Fields template in the Form Layout.



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 *fx* to open the Expression Editor, or ▼ to open the Variables picker.

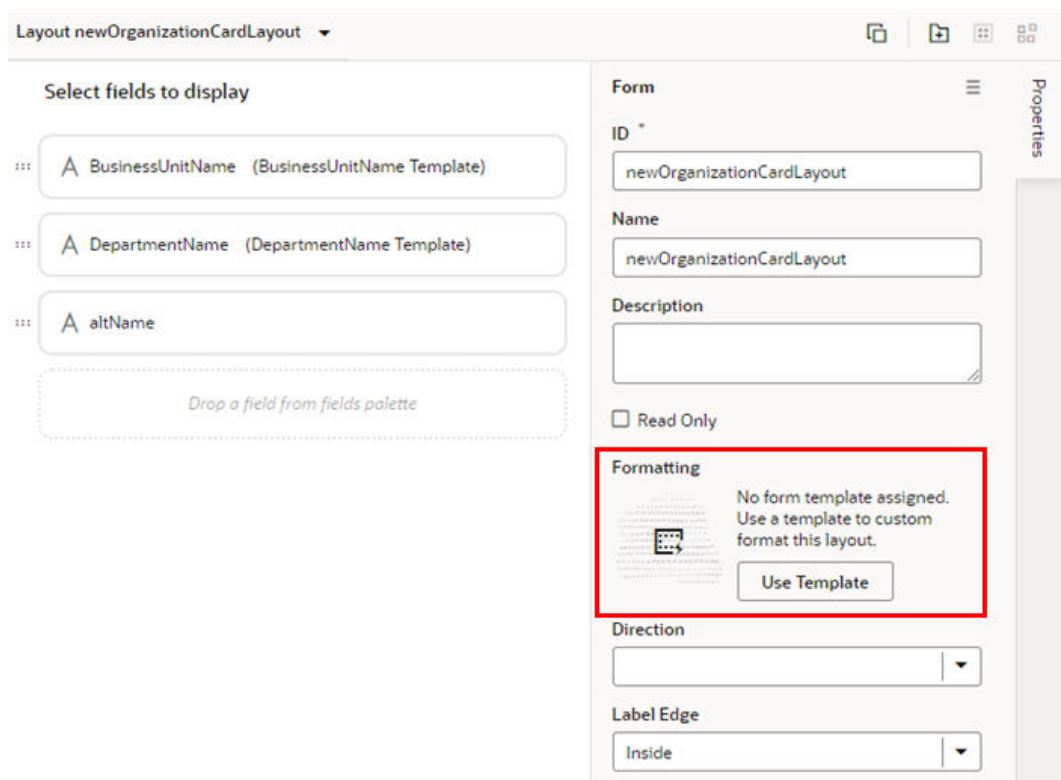


After you've added the components and fields to your form template, you can apply the template when you edit a layout in the Rule Sets editor.

## Apply a Template to a Form

To apply a form template to a form:

1. In the Rule Sets 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 the templates that are applied to them.
2. While the form is selected, click Use Template in the Properties pane.

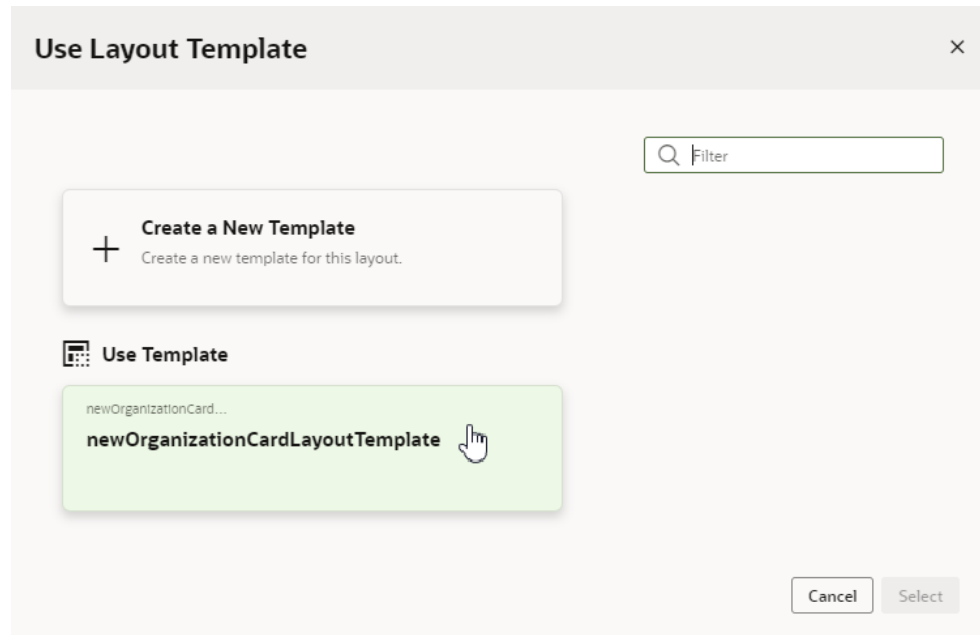


If a template has already been applied to the form and you want to switch to a different one (or remove it), click **Select** in the Properties pane. The list of templates will include form templates defined in the base app as well as the templates you've created.

You can click **Create** in the Properties pane if you want to create a new form template. For more on using templates with a form layout, see [Control How a Form Layout is Rendered](#).

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 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 applied to the form layout (`NewOrganizationCardLayoutTemplate`) and the fields defined by the template (`altName, origin`).

You can't edit templates defined in the base app, but you can edit the fields defined in templates you've created. You can click **Go to Template** in the Properties pane to open the template in the editor.

If the template 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**.

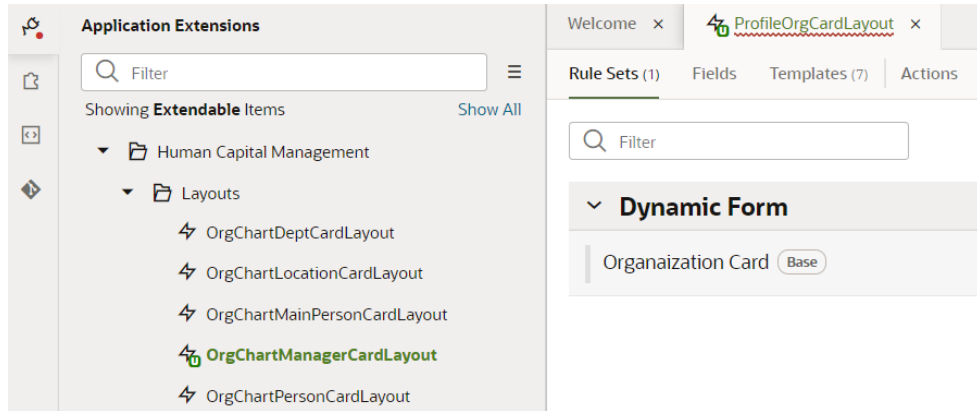
## Add and Group Fields in Dynamic Form Layouts

When creating a layout for a dynamic form on the Rule Sets tab, you can group fields so that they are displayed together in a layout, and 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:

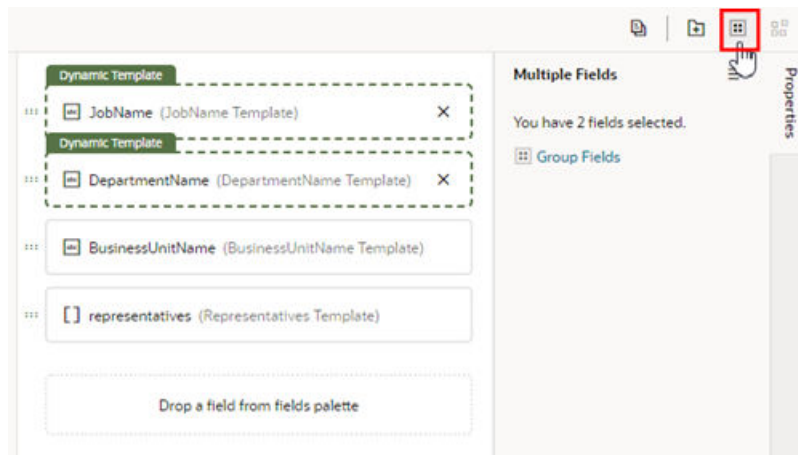
1. Open the Rule Sets tab for the dynamic form you want to work with.  
Select a layout in the Application Extensions pane, then choose the dynamic form; click the form on a rendered page in the canvas area; or select it from the Properties pane.



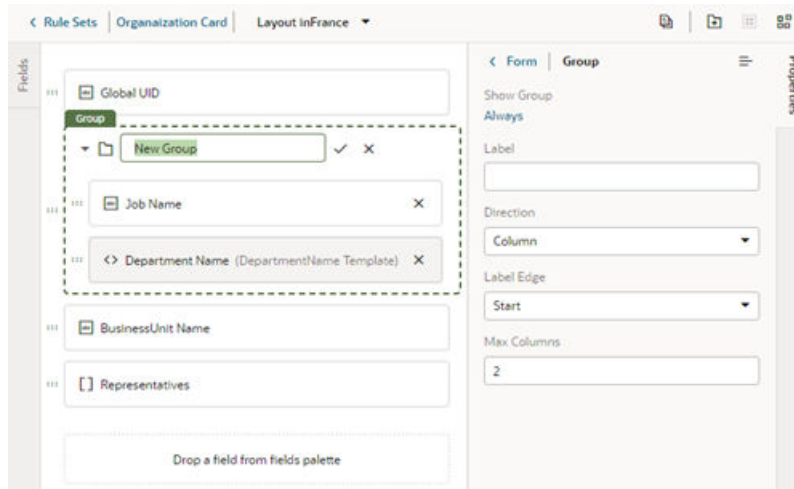
2. Open the form you want to edit.

You can use buttons in the toolbar to duplicate the current layout (📄), to copy selected fields in your current layout to another layout (📄➔), and to copy fields from another layout into the current one (➔📄).

3. In the layout diagram, select the fields that you want to group together, either by holding down the CMD key (on macOS) or the Ctrl key (on Windows).
4. Click **Group Fields** in the Properties pane, or 🗑️ in the toolbar.



The selected fields are grouped under a new folder in the layout diagram:



5. Type a name for the new group by naming the folder in the layout diagram. Click ✓ to save the group name.
6. Optionally, use the Properties pane to set properties for the group. You might even click the Always link to set conditions that determine when the group is displayed in a layout. The default setting is to always display the group.

After a group is created, you can still use the handles for fields to drag them into and out of a group.

## Create Fields For a Layout

You can create fields in your 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.

You can see the list of fields you can use in your layouts in the Fields editor. This list will contain any fields you create, but most of the fields will be defined by your Oracle Cloud Application. The service provides a service definition, such as an OpenAPI3 definition, that describes the fields and properties. Service definitions determine the Oracle Cloud Application fields and properties that you can use in your layouts, and the base app defines the service definitions that your app extension uses.

### 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](#).

Your app extension can't create or modify the fields in your Oracle Cloud Application, or the service definition used by your app extension, but it 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, in your app extension 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

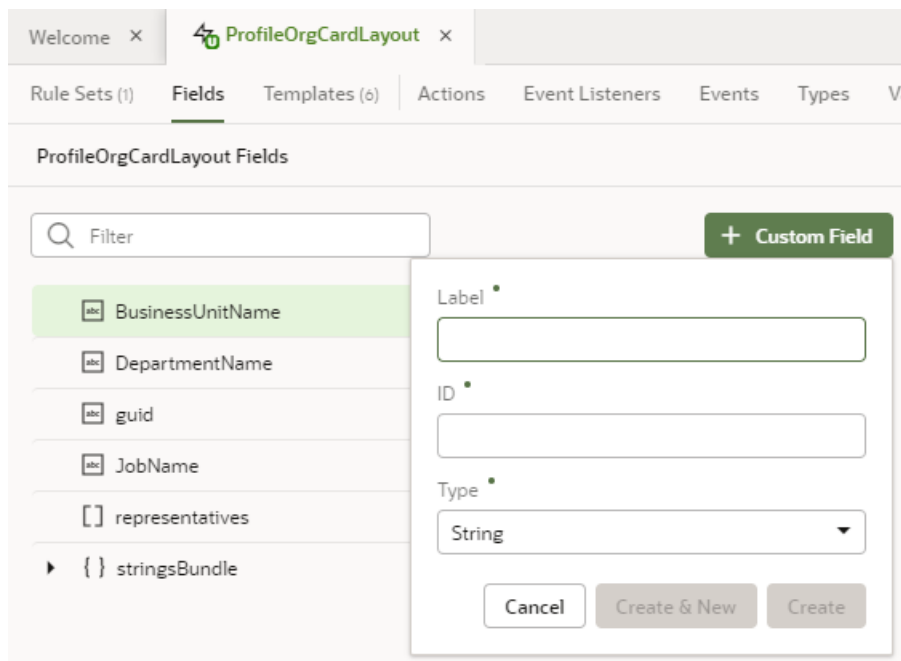
be set to False. However, you can't override a property to make it less strict, meaning you can't set a Required property to False if it is already set to True in the service definition.

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 `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, then click the **Fields** tab.



2. 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.



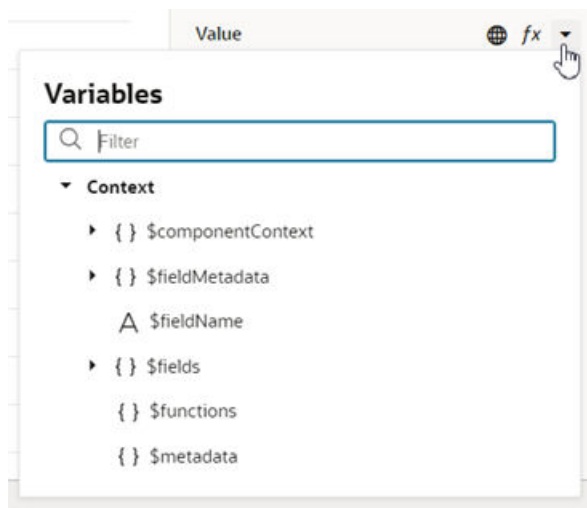
5. If you want to create an expression and use an existing field, click **Referenced Field**, then select a field in the list. Click **+** to add it.

You can add any field available in your 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 because the Value property is now used to specify the expression.

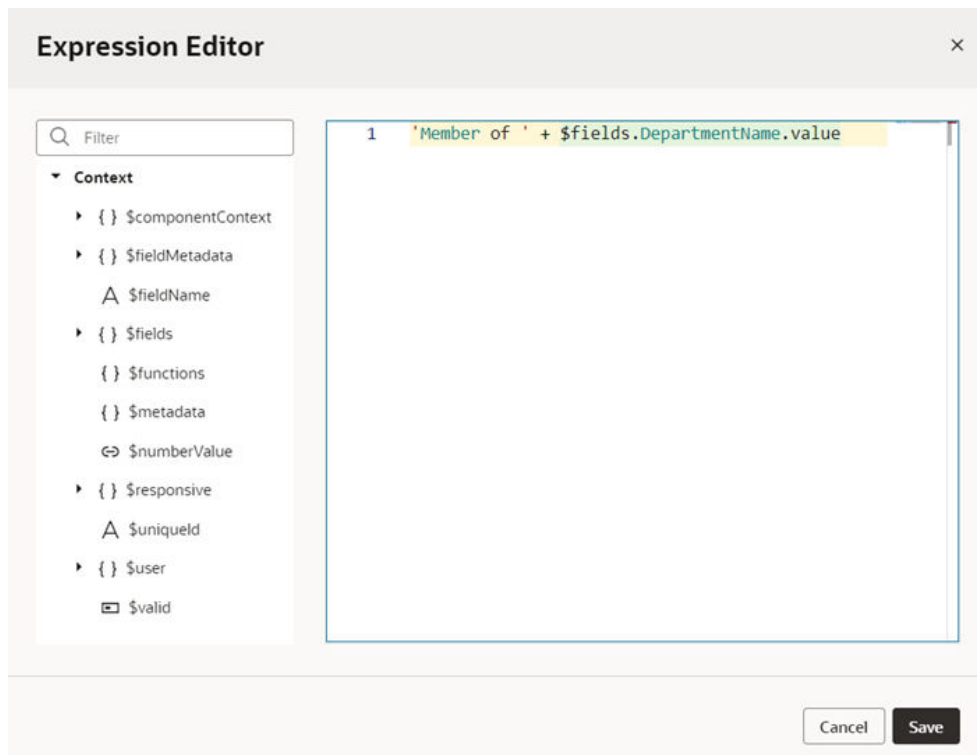
If you set a calculated field to Read Only, you set the field's value or expression in the Value field (there is no Default Value for read-only fields)

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 the arrow to open the Variables picker.



If you want to use an expression, you can click *fx* to open the Expression Editor. In the Expression Editor, you can select field variables in the Variables pane to add them to your expression. You can also add text strings to your expression by typing in the editor. Click **Save**.



The expression you create in the editor is added to the Value field, for example, `[[ 'Member of ' + $fields.departmentName.value() + $fields.JobName.value() ]]`.

**Field** ☰ **Properties**

ID •  
memberOfDepartment

Label Hint •  
Member Of Department

Type •  
String

Read Only

Required

Help Hint Definition

Help Hint URL

Value fx ▼  
[[ 'Member of ' + \$fields.departmentName.val

Referenced Fields Add

departmentName

Converter Add

*Field does not have a converter*

Validators Add

*Field does not have a validator*

7. Optionally, you can click **Add** to add converters and validators to the field.

You can add suitable built-in converters 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, as shown below.

Your custom fields (and any fields that you have modified, for example, in the Properties pane) are indicated by a gray bar to the left of the field name. In this screenshot, you can see the gray bar next to `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.

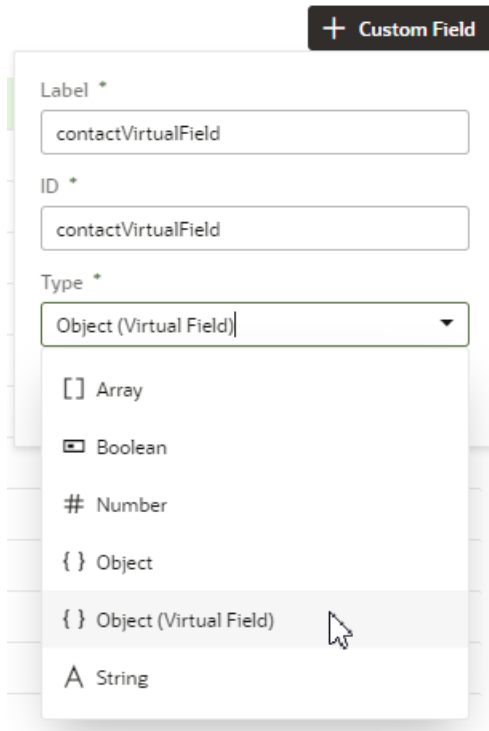
When you add a virtual field to a layout, you'll need to define a field template to display it. You'll need to create the field template if it doesn't exist. The template will contain components for each of the referenced fields that you want to display in the layout.

To create a virtual field:

1. Open the Fields tab of the dynamic table or form you want to work with.
2. Click **Custom Field** and type a label for the field (the field's display name).

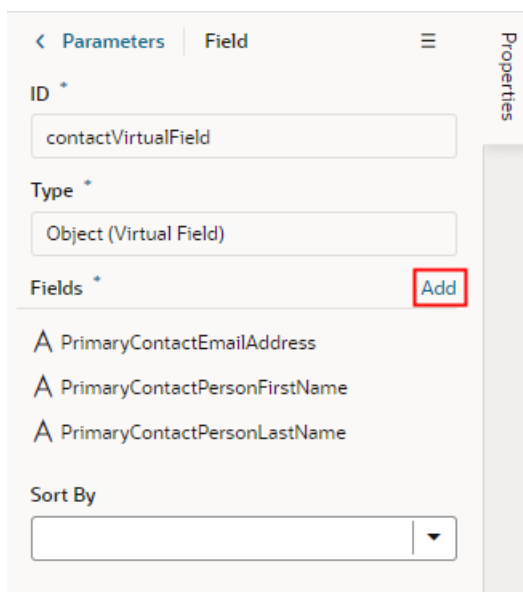
When you type in the label field, a suggested ID is generated for you. The ID can't be changed later.

3. Select the `Object (Virtual Field)` type. Click **Create**.



4. 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.



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

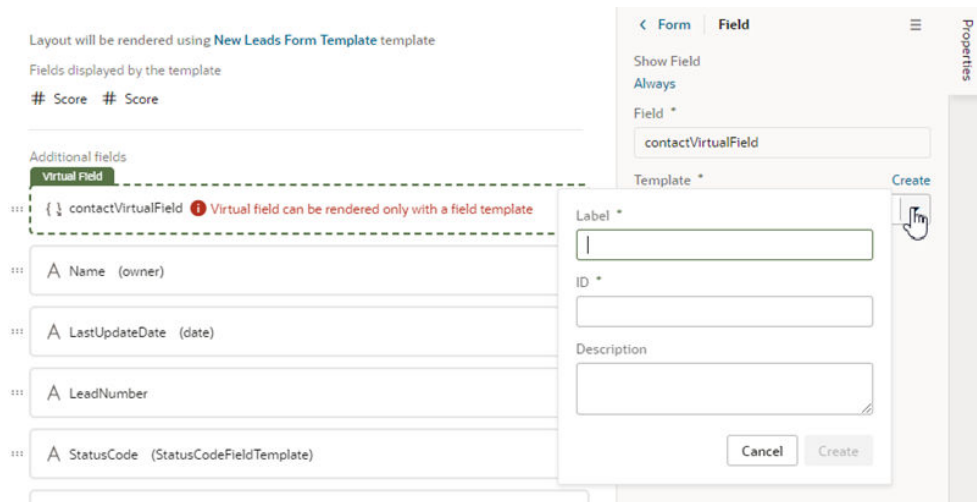
be used when the table is sorted by FullName. The Sort By field will be used for sorting regardless of how the virtual field is rendered in the table by the template. (Remember, you need to use a field template to display a virtual field).

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 rule set layout 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.

9. 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. 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 might want to add a validator to a field to check if a value entered in a field is valid, for example, to check if a date is not earlier than the current date.

You can find details and examples in the *Oracle JET Developer Cookbook*:

- [Built-in Oracle JET converters](#)
- [Built-in Oracle JET validators](#)

To add a converter or validator to a field:

1. In the **Fields** tab, select the field you want to work with.
2. In the Properties pane, click **Add** next to Converter or Validators, then select one from the list.

The screenshot shows the 'Properties' pane for a field named 'ID'. The field's value is 'hireDate', its label hint is 'Hire Date', and its type is 'date'. The 'Read Only' checkbox is checked, and the 'Required' checkbox is unchecked. The 'Help Hint Definition', 'Help Hint URL', and 'Default Value' fields are empty. The 'Converter' section shows 'No converters defined' and an 'Add' button circled in red. The 'Validators' section shows 'No validators defined' and an 'Add' button circled in red.

A default option is selected based on the field's type. For example, the default validator for an employee's Email field that uses the Email format is the Expression Validator:

3. Change the type if needed, enter additional details, then click **Add Validator** or **Add Converter**.

The details you'll need to enter will depend on the validator or converter you use, so you might need to consult the samples and documentation for the specific options. Use the JSON editor if you want to add options other than those shown on the UI. For the Length Validator shown here, the options specify how to count the characters and the minimum and maximum string lengths allowed:

You can also create your own validator or converter by selecting the **From Code** option. With this type, the **path** field specifies the location of a JavaScript file that implements the custom validator or converter; the **name** field specifies the name of the constructor; and the **Option** field specifies the options specific to the custom validator or converter, for example:



Converter Type \*

From Code ▾

path \*

resources/js/RelativeDateTimeConverter


name \*

RelativeDateTimeConverter



Options

```
{
  "relativeField": "day"
}
```

In this example, the `RelativeDateTimeConverter` JS file implements a converter with a constructor named `RelativeDateTimeConverter` and a `relativeField` option whose value can be, for example, `day`, `week`, `month`, and `year`. The implementation would convert a date value like `2014-01-02T20:00:00` to a relative date value, like `Today`, `Tomorrow`, `This Week`, `Next Week`, and so on, based on the value of the `relativeField`.

It's possible to update your validator and converter options any time after they've been added. Hover near the validator or converter name, click , and make your updates. You can add as many validators as you want, but a converter can only be replaced because a field can have only one converter.

Converter • Replace

RelativeDateTimeConverter   day

relativeField


Validators Add

No validators defined

Converter • Replace

Expression Converter

expr `[[ $functions.petNameConverter() ]]`

Validators Add 

Length Validator •

countBy codeUnit

min 1

max 25

Regular Expression Validator •

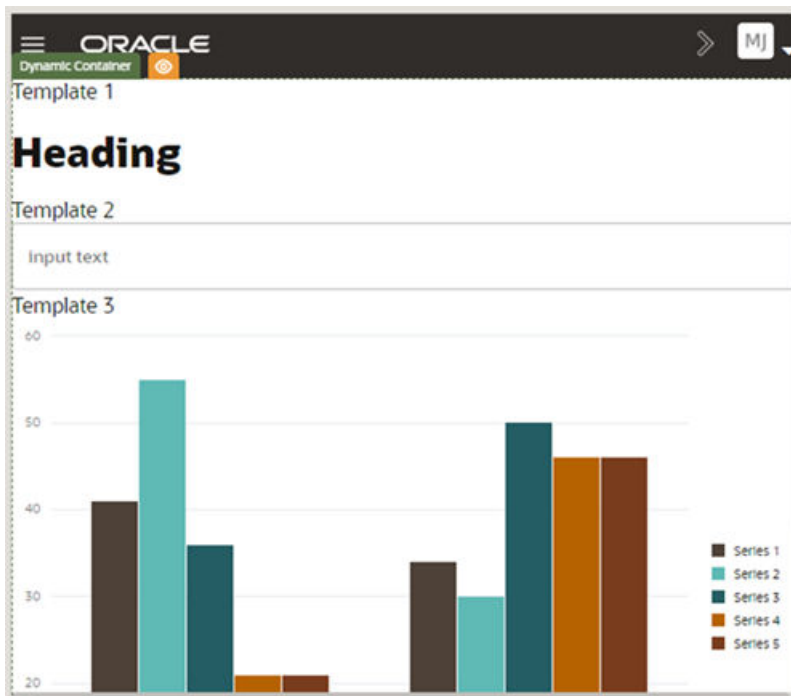
pattern `^[a-zA-Z][a-zA-Z0-9]*$`

# 4

## Display Your Own Content

If the base app developer inserted a dynamic *container* on the page and made it extensible, 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 base app 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.

### Note:

Unlike dynamic tables and forms, a dynamic container is scoped to the page on which it appears, which means it cannot be shared across other pages or app extensions.

## 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 layouts used in dynamic forms and tables, but instead of selecting which fields to display, you select which UI elements or 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. A container can only display one case at a time, so if a case defines three sections, when that case is displayed you'd only see those three sections displayed in the container.

## Configure a Dynamic Container

The easiest way to learn how to configure a dynamic container is to examine a real use case. Suppose the base app developer established a default case that doesn't have any sections. At runtime the container will be empty. Now let's say that you want to display a header and form in the container, but *only* when the current user is a manager.



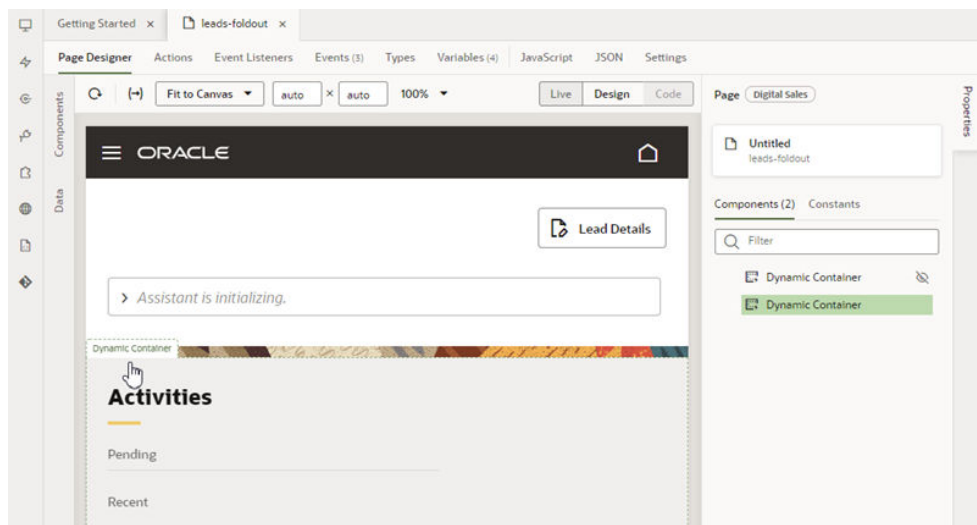
### Note:


Dynamic forms and dynamic tables cannot be added to dynamic containers at this time.

To achieve this, you would:

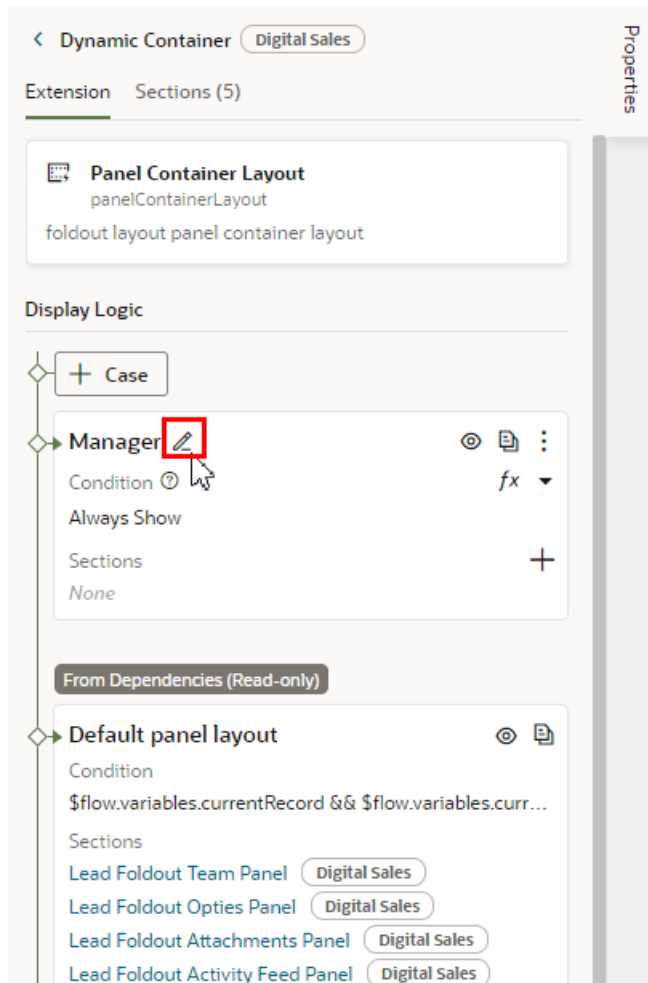
1. Open the page containing the dynamic container in the Page Designer and click anywhere within the container's green outline.


You can also open a container by clicking it in the list of dynamic components in the Components tab in the page's Properties pane.



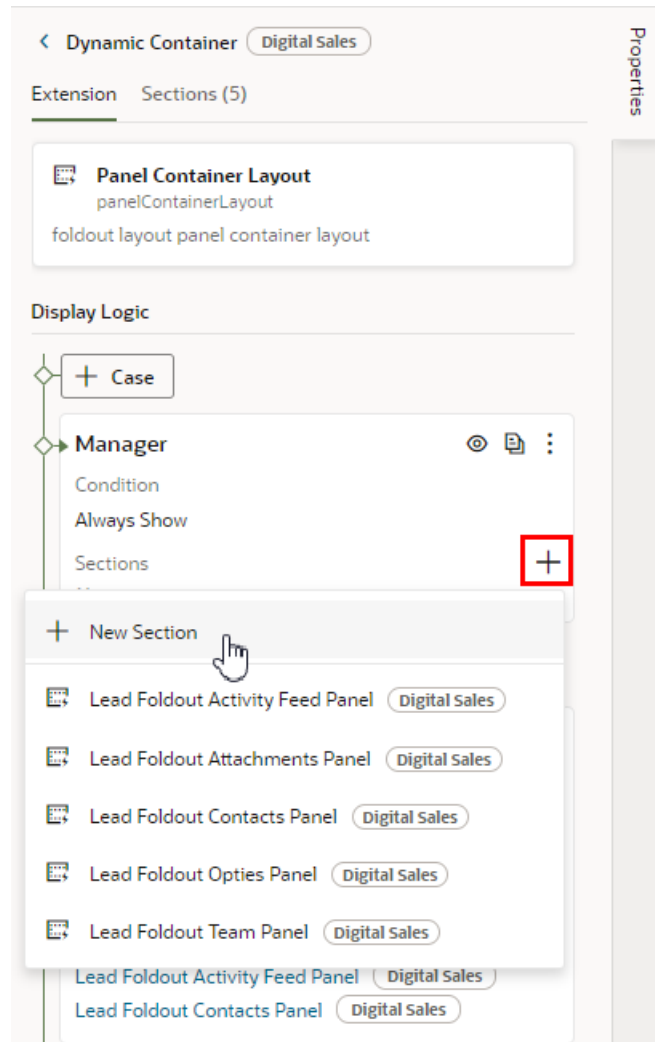
2. In the Properties pane, click  **Case** to create a brand new empty case, and give it a unique name—perhaps **Manager**.

To edit the case name, click  next to the case name:

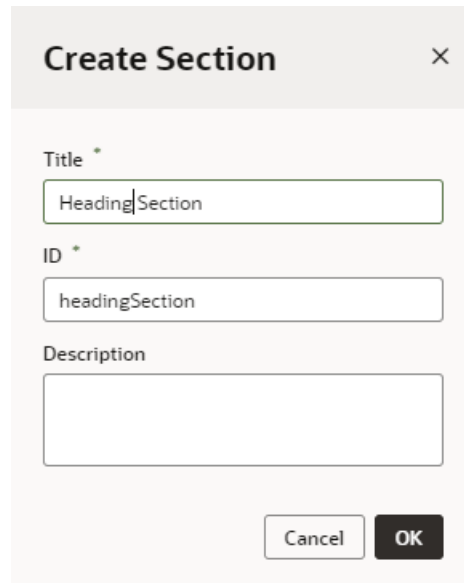


If there is a case that contains some sections you know you'll want to re-use, you could click  to duplicate it, and then remove and add sections to the new case as needed.

3. Add sections to your new case.
  - a. Click  next to **Sections**, then select **New Section** in the menu to create a section:




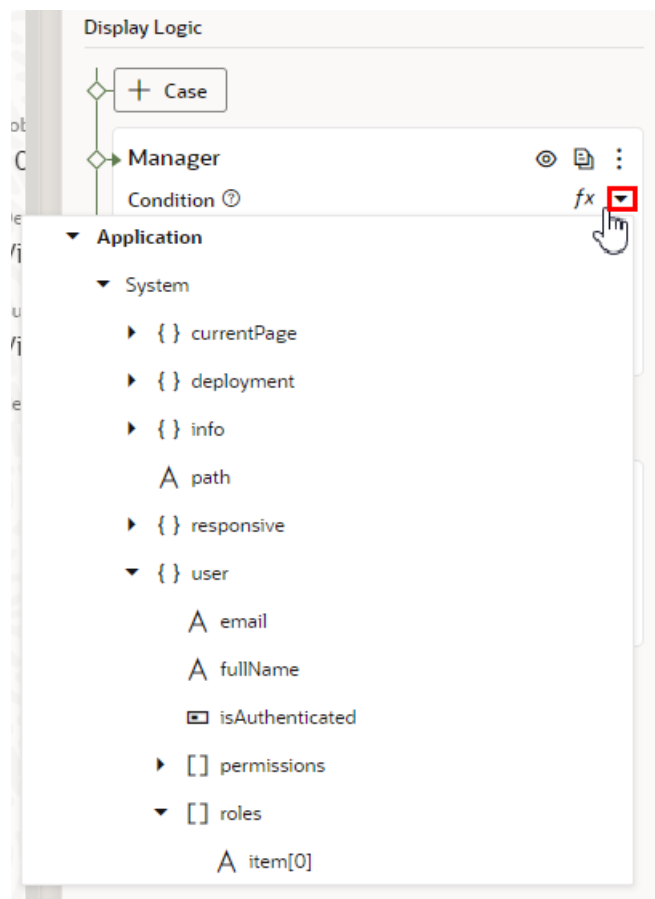
- b. In the Create Section dialog box, type a title for the section. Click **OK**.  
The ID is added for you based on the title you provided.



The image shows a 'Create Section' dialog box with a close button (X) in the top right corner. It contains three input fields: 'Title \*' with the text 'HeadingSection', 'ID \*' with the text 'headingSection', and 'Description' which is empty. At the bottom, there are 'Cancel' and 'OK' buttons.

In this example, create two sections in the case: Manager Heading Section and Form Section. These sections can be re-used in other cases.


4. In the case's Condition field, click  to display the variables you can use in your conditions:

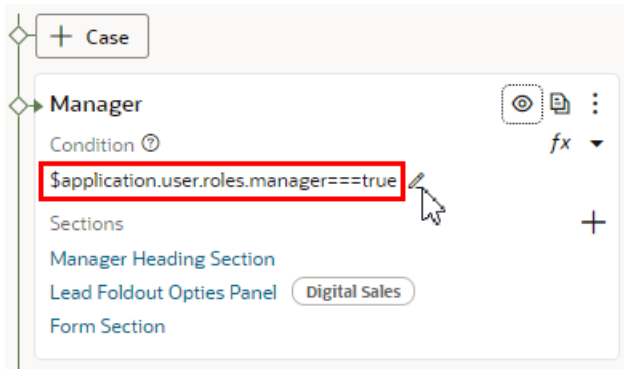


The image shows the 'Display Logic' configuration window. It features a tree view with a 'Case' node and a 'Manager' node. The 'Manager' node has a 'Condition' field. A red box highlights the 'fx' icon next to the 'Condition' field, which is being clicked by a mouse cursor. A dropdown menu is open, showing a list of variables under the 'Application' category. The variables include: 'System' (with sub-items: 'currentPage', 'deployment', 'info', 'path', 'responsive'), 'user' (with sub-items: 'email', 'fullName', 'isAuthenticated', 'permissions', 'roles'), and 'item[0]'. The 'isAuthenticated' variable has a checked checkbox next to it.

- Expand the variables tree and click **roles** (located under Application->System->user).

When defining an expression, you can use any of the variables and constants available to the app extension. Some of these might have been defined in the base app. (For descriptions of the Built-in Variables, see Built-in Variables in *Building Responsive Applications with Visual Builder Studio*.)

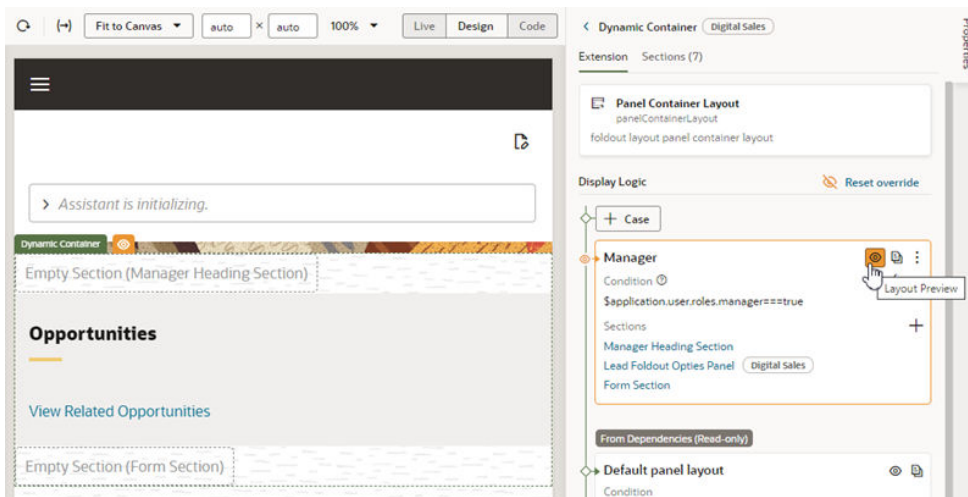
- In the Condition field, click , then type in the rest of the expression as shown:



You set the case's conditions by typing an expression that can be evaluated as true. This expression evaluates as true if "manager" is a role assigned to the user viewing the page.

- Click **Layout Preview** in the case.

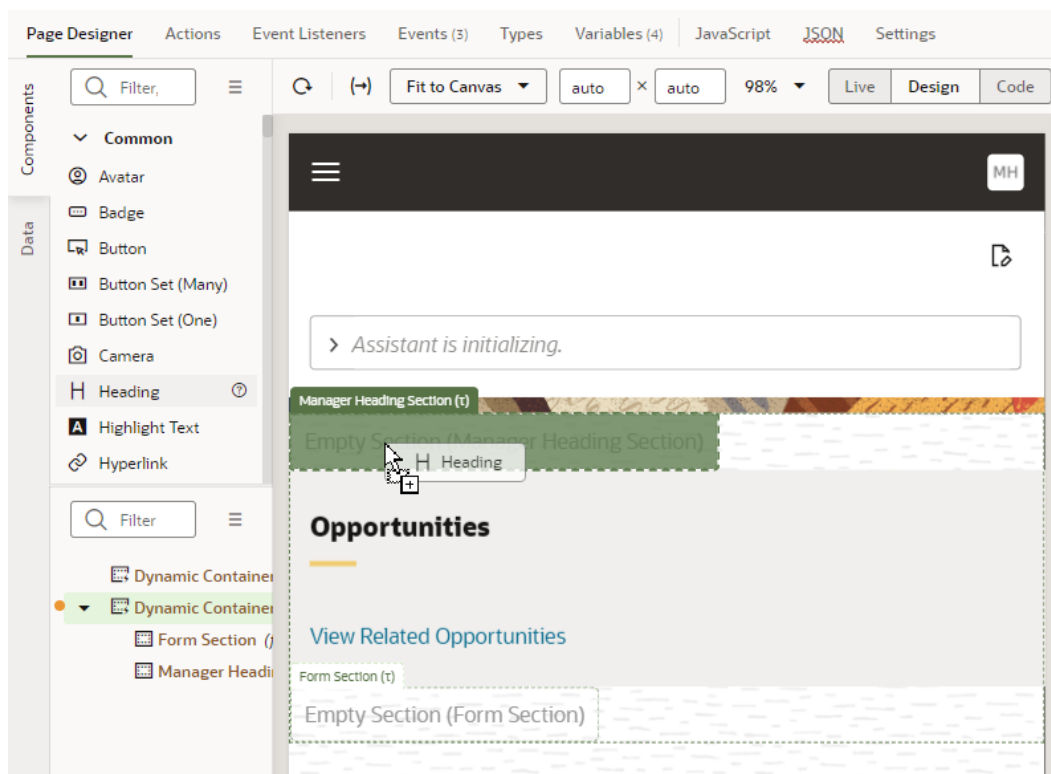
Activating a case's Layout Preview forces the Page Designer to display that case in the container, so you can see the components in the case's sections.



When a section is visible in the Page Designer, you can drag components directly into it from the Components palette and see how they will be rendered in the page.

- In the Page Designer, drag components from the Components palette into the sections.

You can drag components directly into the section on the canvas or in the Structure view.



### Tip:

If data sources are available in your extension, you could also drag an endpoint from the Data palette into the section to add forms or tables.

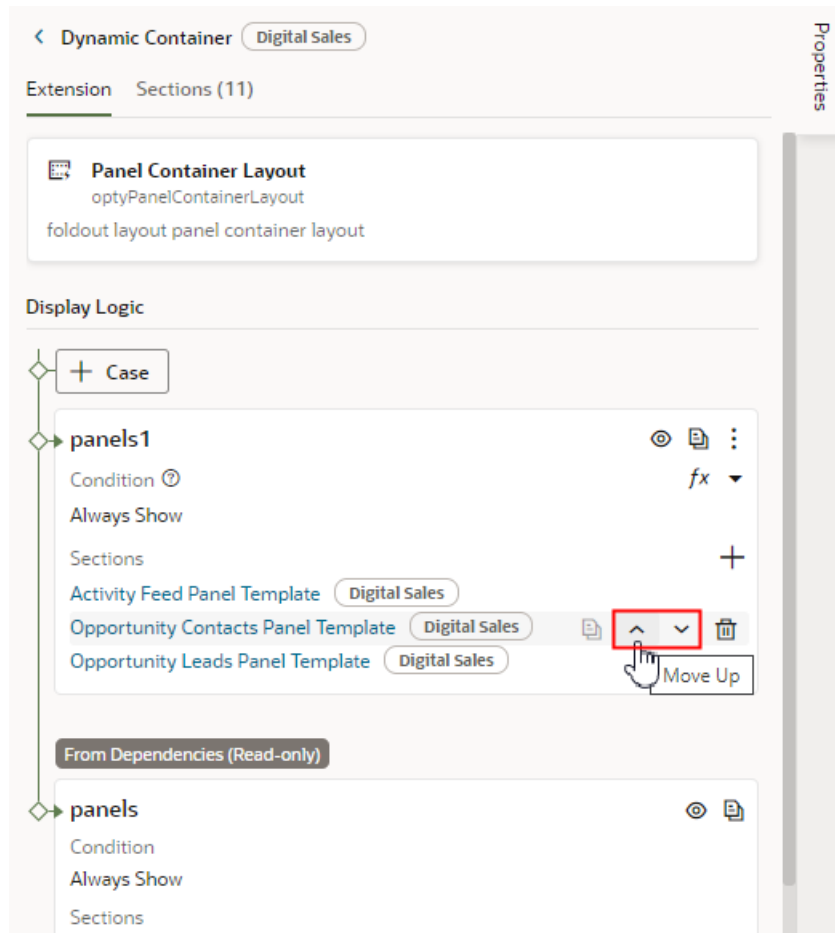
9. Select the component in the section, and then configure it in the Properties pane.
10. Click **Reset override** in the Properties pane to go back to using the container's display logic.


At runtime, if the first case, Manager, is true—that is, if the current user is indeed a manager—then the two sections will be displayed in the container in the order they are listed in the case. If the user is not a manager, then the default case is used, which in this example is a container with no sections.

## Re-Ordering a Container's Content

Besides defining new cases and templates for a container, you can also change the order the templates are displayed in the container. Just use the Move Up and Move Down arrows for the template under Sections:





You can also use the  icon to remove a section from a case. Once removed from a particular case, the section is still available to use in other cases for dynamic containers in the page.

 **Note:**

You can't edit the built-in case set by the dependency's developer, but you could duplicate the case and then use the arrows to 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:

- A section you create in an extension is the full width of the container, although the base app 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, or a button that starts an action chain in your app 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 the Oracle Cloud Application resources available in the page.

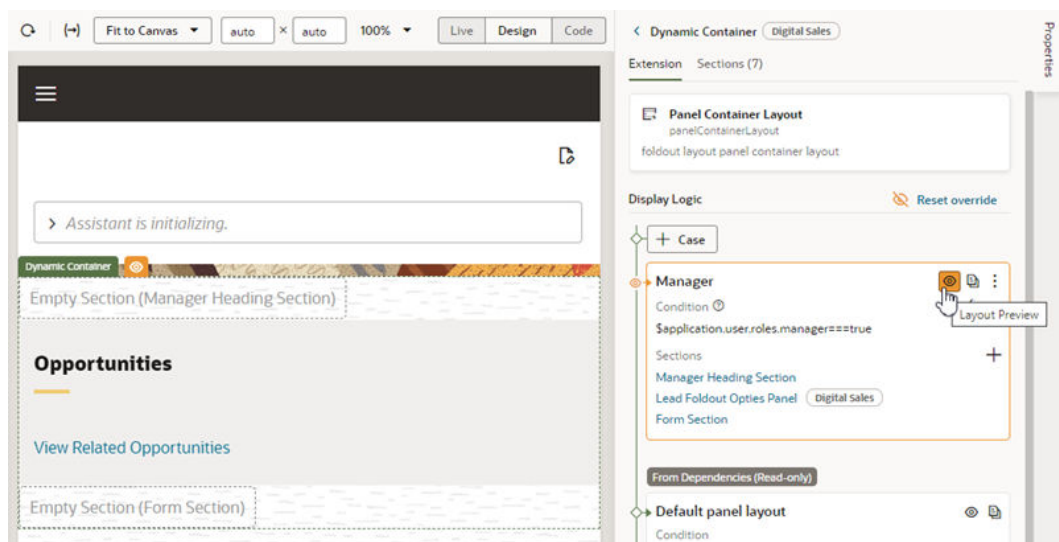
## Add Content From a Custom Object to Your Page

You can add components to a container section to render data from a data resource, for example, if you have created custom child objects in your Oracle Cloud Application. The component might be one provided in the base app, or a standard component available in the Components palette. For example, you can add a List View component to a section in a dynamic container, and then use the component's Quick Start to bind the component to the data source and select the fields displayed by the component. For more on using Quick Starts to bind data, see *Add Data to a Table or List in Building Responsive Applications with Visual Builder Studio*.

To display data from your Oracle Cloud Application in a list component:

1. In the Page Designer, select the dynamic container where you want to display the data.
2. 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:



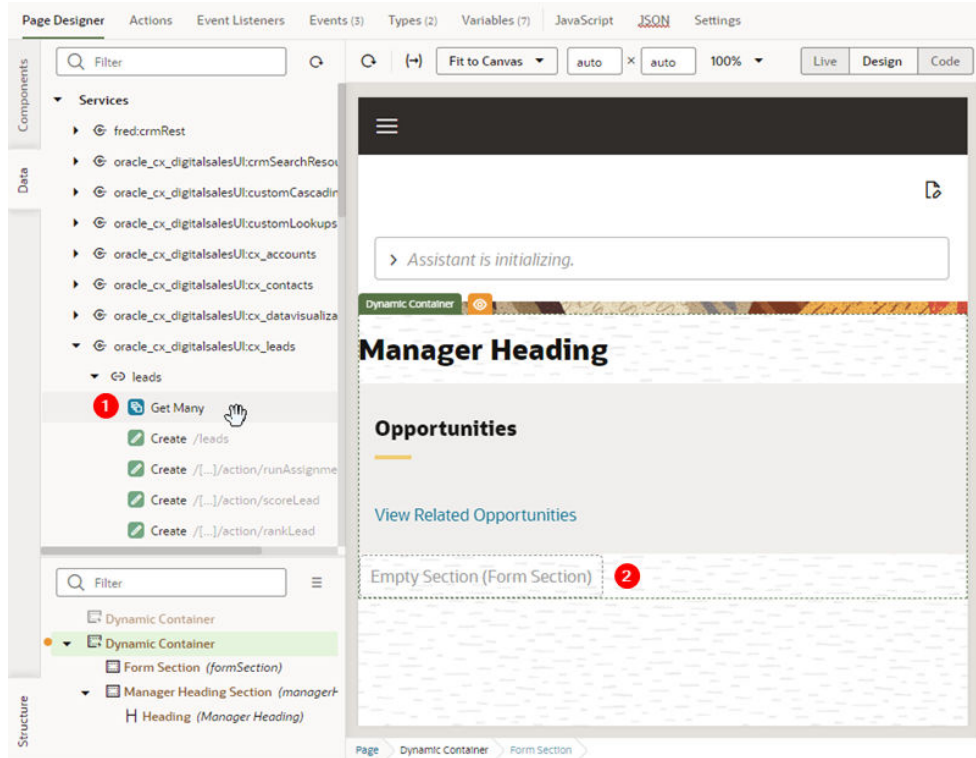
In the image above, the case already has a section named Form Section that we can use for the new component.

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 your extension's accessible endpoints.

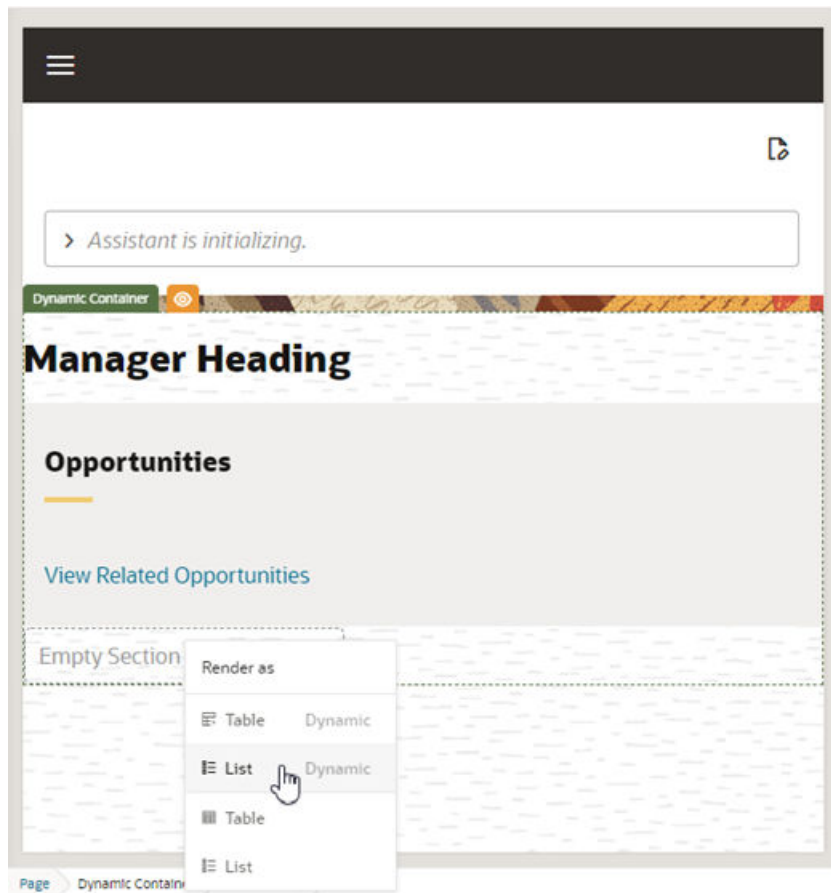
4. Drag the endpoint from the Data palette and drop it onto the section on the canvas.

You can also drag components and endpoints into the Structure view.



5. 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:

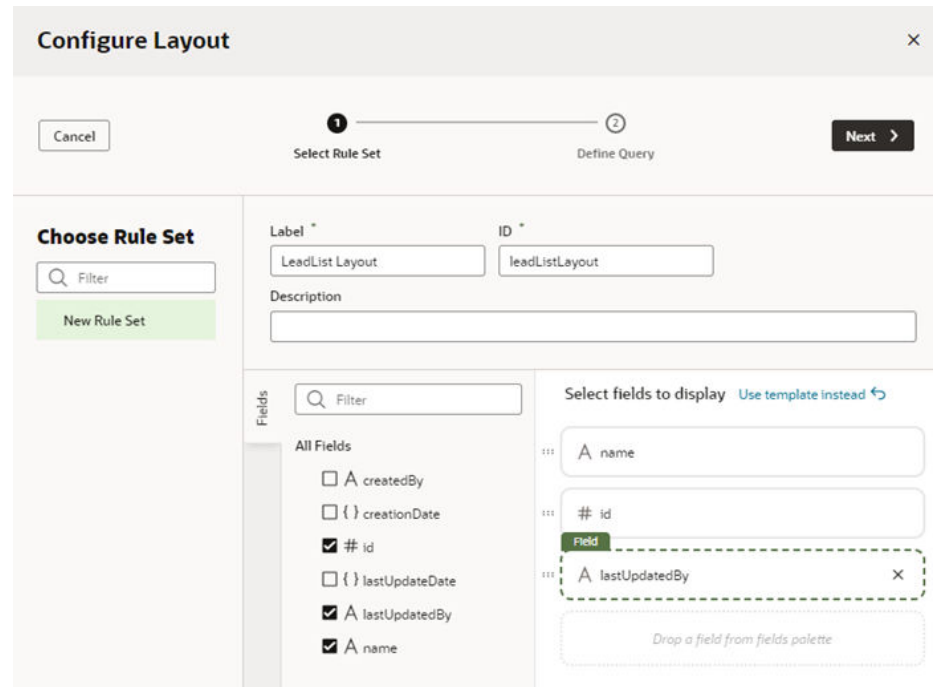


In this example, we'll add a dynamic list component. The Configure Layout wizard opens automatically when you select a dynamic component.

6. In the Configure Layout wizard:

- a. 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.



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 expression endpoint. Click **Finish**.

After you finish the wizard, you'll see the component rendered in the container in the Page Designer.

- 7. Click **Reset override** in the Properties pane to go back to using the container's display logic.

# 5

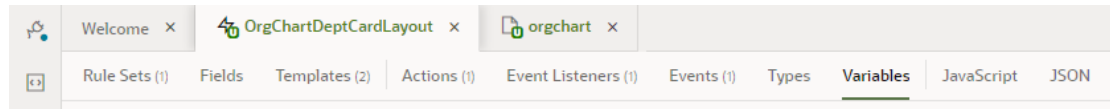
## Advanced Techniques for Customizing Your App Extension

The dynamic layout templates (field templates, display templates) expose components to their programming model (*layout model*), so you can bind components in templates to events, action chains, variables and functions defined within its model. This enables you to add more advanced custom behavior to your app extension.

For example, if you add a new button component to a layout template, you might need to edit variables, events and actions to define what the button does.

### Advanced Layout and Dynamic Component Editors

While working with your app extension's field or display templates, you can use the advanced editors (Actions, Event Listeners, Events, and Variables) to add more advanced features to the extension:



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.

Editors have access to elements in its own layout scope only. If you're editing a page, for example, the editors can access only elements defined on that page. Elements defined in a dynamic layout or the containing flow would not be available for editing.

### Customize Your App Extension with Variables and Constants

Variables and constants store static values and expressions that are used to define how pages and components in your app extension behave. For example, if the base app stores the color of a heading in a variable, you can change the value of the variable (and thus the

color) in your app extension. The base application developer or a team member might have already specified some variables and constants that you can extend, but you might also want to add others. When a page is open in the Page Designer, editable constants used in the page are listed in the Properties pane.

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 the base app, and their values don't change after they are set by the app extension. For example, the base app might define a heading that uses the constant `itemDetailHeading` for setting the heading text. Your app extension can write a value to the constant, and that value will be used to set the heading text every time `itemDetailHeading` is used in the app extension. The default value of constants might refer to expressions that contain variables.

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 artifact, it would have a page scope. In your app extension, 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 display templates, but are not accessible from within dynamic forms and tables. These are typically defined in the base app and are listed in a page's Variables editor if they are editable. For example, the base app might define an editable constant that you can use to toggle the display of a dynamic component.
Layout (dynamic forms and tables)	Variables and constants with a layout scope can be used by components and actions within dynamic forms and tables, for example, within field templates. These are typically defined by you, and you create and edit them in the layout's Variables editor. 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. Variables with this scope are listed in the Variables editor of the action and are not listed in the Variables editor for layouts or pages.

You can view, create and edit variables and constants in the Variables editor of a page or layout. If you want to use any special characters (period, colon, dash and space) when creating field names for variables, 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 editor of your page or layout.

**Note:**

If the base app defines types that can be used in app extensions, they will also be displayed in the Types editor, and you can select them in the Type dropdown list when you create a variable.

For more details about using variables, constants and types, see *Work with Variables and Types* in *Building Responsive Applications with Visual Builder Studio*.

## Change the Default Value of an Extendable Constant

Constants may be used in the base app to determine app behavior such as which cards are displayed or hidden in the app. If the base app developer makes these constants available to your app extension, you can overwrite the default values to change the behavior as needed.

If a constant can be modified in the app extension, you can set its value to a static value, another constant, a variable, or an expression. A simple example of how a constant might be used in an extension is to store a text string, say for a Heading component. If the heading's value is set to an extendable constant, you can use your extension to set a value for the constant, which will be used as the text rendered by the component.

Another way a constant might be used is to configure the behavior of components. For example, a base app developer may define an element in the page, and allow you to hide or display the element using an extendable constant. In the example below, the `showEloquaProfiler` is a Boolean constant and takes either "true" or "false" as a value. If you don't want to display the Eloqua profiler element in your app, you can hide it by setting the `showEloquaProfiler` constant to "false".

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 element on mobile devices only. In that case, you can use the `$responsive.smOnly` system variable to set the value of your constant. On mobile devices where the `$responsive.smOnly` value is true, the value for `showEloquaProfiler` is set to "true" and the Eloqua profiler is displayed. On desktops where the variable is false, the element 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 same 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)`).

You can create more complex expressions using one or more constants and variables and include text. For example, you may want to modify a link on a page to point to different web pages depending on the user's country. In this case, you may need to build an expression that resolves to a particular national web site depending on the value of a variable for the user's country.



In this scenario, you might build your expression using a constant for the company name, a new string variable set to a regional top-level domain based on the user's country (".com" for the US or ".co.uk" for the UK), and text to fill out the URL:

```
[[ 'http://www.' + $companyName.value() + $regionalDomain.value() + '/support/docs/index.html' ]]
```

For a user in the US, this expression resolves to "http://www.example.com/support/docs/index.html".

 **Note:**

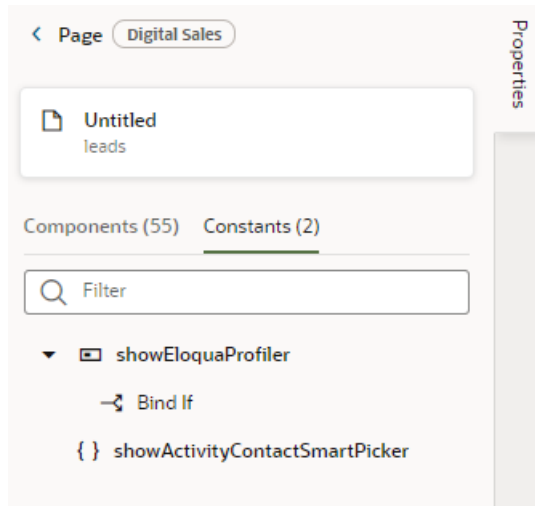
Make sure the value you enter matches the type of the constant: text for a string constant, "True" or "False" for a Boolean constant, and so on. The Page Designer does not validate the value or expression you enter, and you may experience problems during runtime if the type doesn't match.

You can change the value of a constant using either the Page Designer or in the Variables editor tab. The Properties pane in the Page Designer lists all constants used by the selected page; the Variables editor lists all constants defined in the page or layout, including those that are not currently in use. From the Variables editor, you can open each constant and set a new value. In this topic, we will show you how to change the default value from the Page Designer, but the procedure is the same when you use the Variables editor.

To overwrite the default value of a constant in a page:

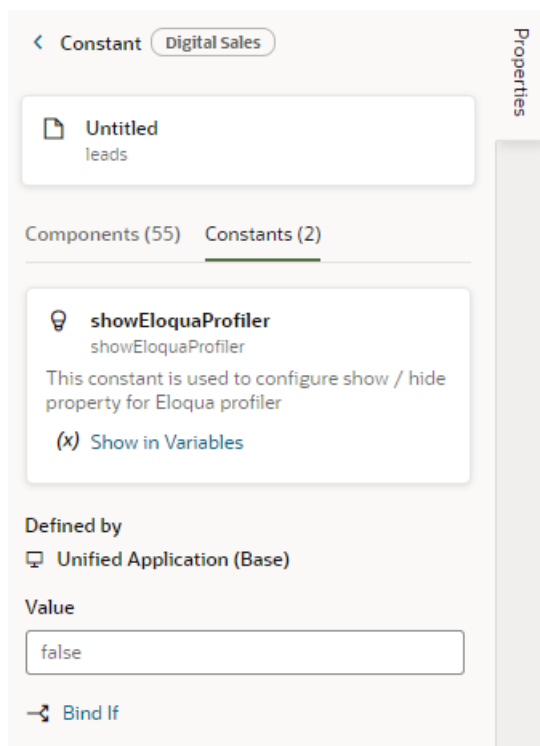
1. Open the page in the Page Designer.

Editable constants used in the page are listed in the Constants tab in the Properties pane:



In this example, you can see that the page has two editable constants.

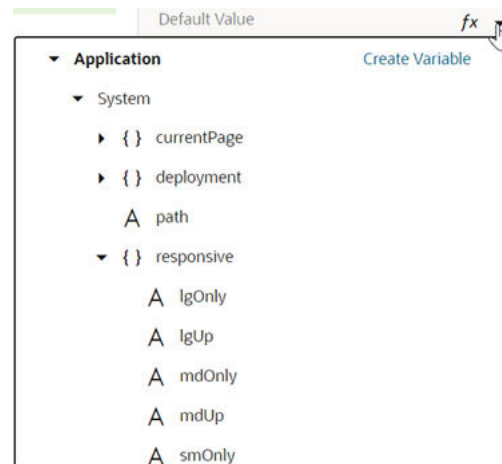
2. In the Constants tab, select the constant you want to edit to see more details about it.



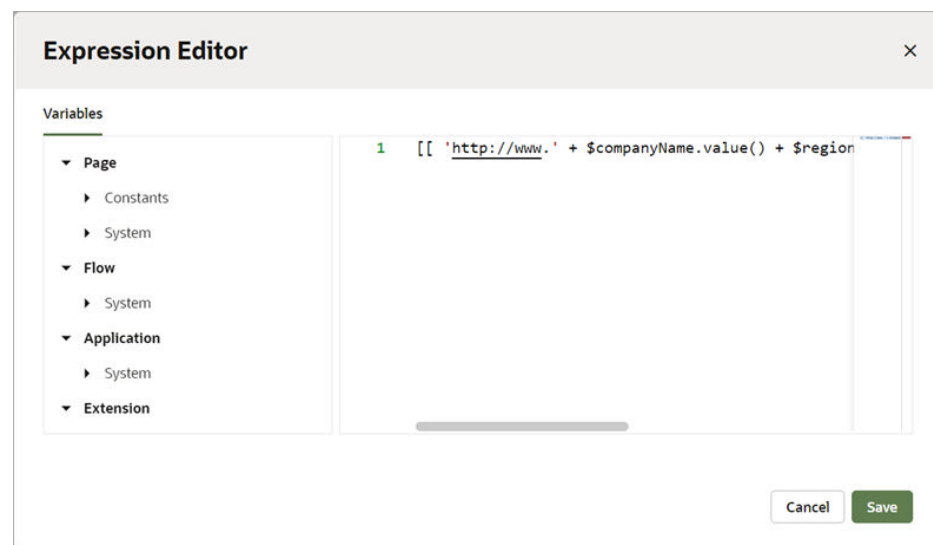
In this example, in the Constants tab you can see details about the `showEloquaProfiler` constant. The developer has provided a description telling you that you can use the constant to show or hide the Eloqua profiler element in the page. The constant is used by a `Bind If` component (which uses a Boolean value), and the value of the constant is set to `false`.

There is also a link to open the constant in the Variables editor tab.

3. Do one of the following:
  - Set a static value for the constant by typing a new value in the **Value** field. In this example, if you type "true" in the Value field, the Eloqua profiler element will be displayed on the page during runtime. If you open the Variables editor, you'll see that the default value for the constant is now "true".
  - Use a constant or variable to set the value of the constant by clicking the arrow above the **Default Value** field to open the Variables picker and select a variable.



- Create an expression by clicking *fx* to open the Expression Editor. 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**.



The expression you create in the editor is added to the **Value** field, for example, `[[ 'http://www.' + $companyName.value() + $regionalDomain.value() + '/support/docs/index.html' ]]`.

## Create or Edit Variables and Constants

You can create or edit variables and constants in a page or layout using its Variables editor. Each page and layout has a Variables editor. The editor lists the defined variables and constants, and contains an Add Variable button for creating new variables and constants.

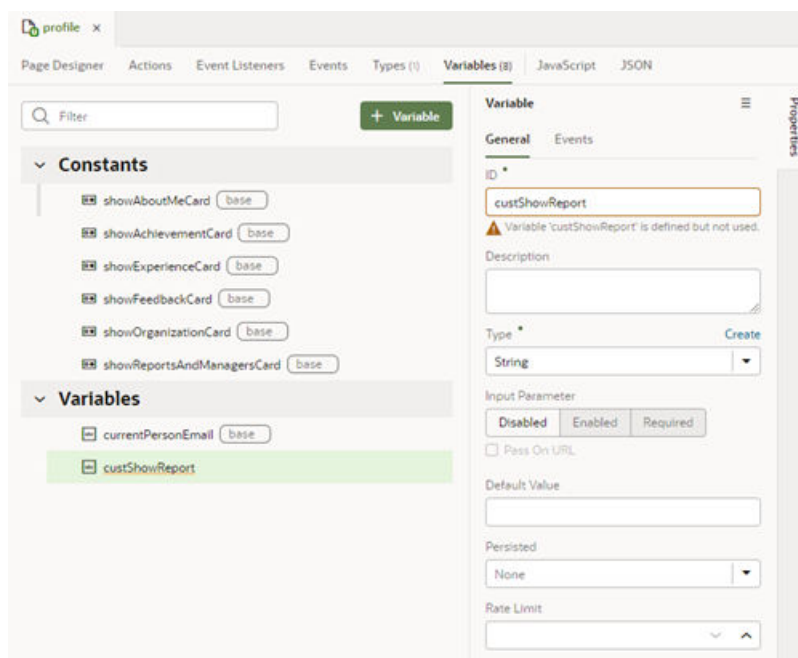
To edit a variable or constant in the editor:

1. Open the app extension page or layout in the Designer.

Editable variables and constants used in a page are listed in the Properties pane when you open the page in the Page Designer.

2. Open the Variables editor and select the constant or variable.

The editor lists all the editable variables and constants, even the ones that are not currently used.

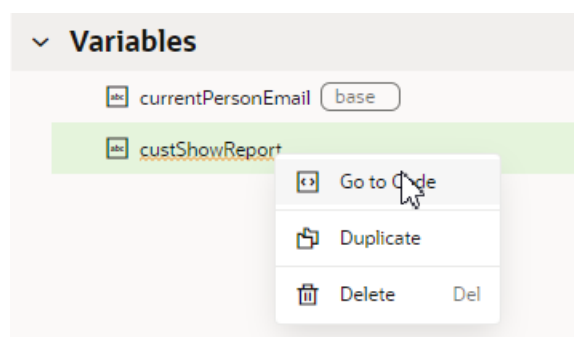


3. Edit the attributes in the Properties pane.

Variables and constants that are defined in the base app are badged with `base`, and have fewer editable attributes than those you create in your app extension.

For variables you create in your app extension, you can edit the attributes in the General tab in the Properties pane, and add event listeners in the Events tab if you want to trigger an action chain.

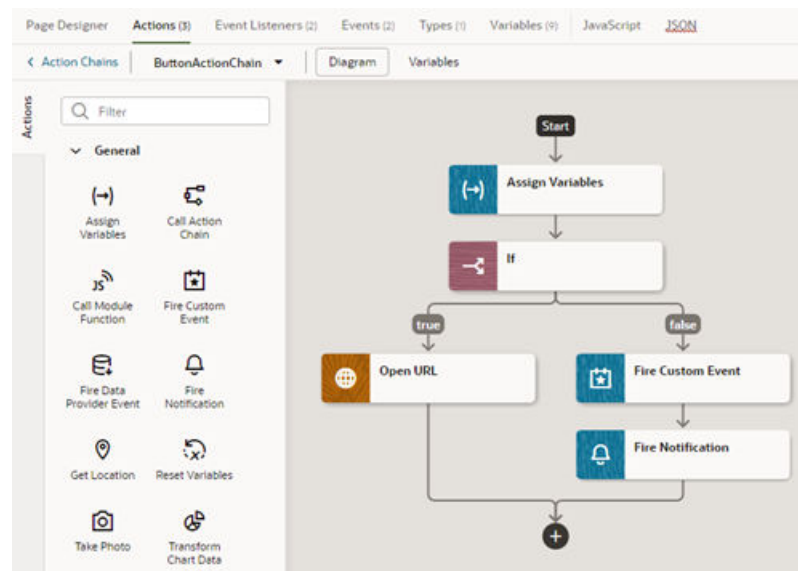
To edit the JSON file where a variable is defined, you can right-click the variable name in the Variables editor and choose Go to Code in the menu:



## Define the Behavior of Your App Extension with Action Chains

The behavior of each component in your app extension is determined by an *action chain*, a sequence of actions started by an *event listener* when an *event* occurs in a page. For example, when you click a button on a page, what happens next is determined by an action chain. To configure the button's behavior, you could define an `ojAction` event for the button, create an event listener that listens for that `ojAction` event to occur, and select the action chain that the event listener will start. When the button's `ojAction` event occurs, the event listener starts the action chain.

An action chain might be a short sequence of a few actions, but it could contain many actions as well as logic for determining what happens in the sequence. An action chain might contain actions such as assigning data to a variable, sending data to a database, navigating to another page, and even starting other action chains. This image shows what an action chain that opens a URL might look like.



Action chains can be defined in your app extension and in the base app. You can't edit action chains defined in the base app, but you can create and edit action chains in your app extension using the Action Chain editor. You can create action chains used in pages and in a layout, and, like variables and constants, the scope of the action chain depends on where it's created. An action chain created in a page's Action Chain editor can only be used in that page, and can only access variables defined in that page. For details on how to create and use action chains, see [Work with Actions and Action Chains](#) in *Building Responsive Applications with Visual Builder Studio*.

Though you can't edit the action chains, events and event listeners defined in the base app, the base app might allow you to use them in your app extension.

## Define Events in Your App Extension

An event occurs when something happens in your app extension. Some examples of common events include when a page loads (a lifecycle event), a button is clicked (a component event), and when the value stored in a variable changes (a variable event). There are different event types, and the type depends on how it is triggered. For example, changes in a button component would trigger a component event.

For some event types, for example, component events, there are many pre-defined events you can choose from, and the events that are applicable will depend on the component. An event like `ojAction` that is applicable to a button might not be applicable to a text field. For other event types, for example, variable events, you can only define one event (`onValueChanged`) for the variable.

The base app might also define events that you can use in your app extension. If an event is designated as "listenable" in the base app, you can create an event listener for it in your app extension.

### Types of Events

How you define events and event listeners in the Designer will depend on the type and scope of the event. This table describes how to define the different types of events.

Type of Event	Description	How to Define
Variable event	A variable event occurs when the value of a variable changes. The only applicable event for a variable is <code>onValueChanged</code> .	You define a variable's event and create its event listener in the Variables editor. See <a href="#">Start an Action Chain When a Variable Changes</a> .
Component event	A component event occurs when a component in a page (including components in dynamic components) triggers an event, for example, an <code>ojAction</code> event for a button. You can choose which events a component triggers.	You define a component's event and event listener in the component's Properties pane in the Page Designer or Templates editor. See <a href="#">Start an Action Chain From a Component</a> .
Custom event	A custom event is triggered by the Fire Event action ( <code>fireCustomEventAction</code> ) in an action chain.	You can create a custom event in the Action Chain editor or in the Events editor of a page or layout. You create event listeners for custom events in the Event Listeners editor. See <a href="#">Use Events Defined in the Base App</a> .
Lifecycle event	Lifecycle events are pre-defined events that occur during a page's lifecycle. You can create event listeners for these pre-defined events: <code>vbEnter</code> , <code>vbBeforeEnter</code> , <code>vbExit</code> , <code>vbBeforeExit</code> , <code>vbAfterNavigate</code> .	Lifecycle events are pre-defined, but you can create event listeners for them in the Event Listeners editor. See <a href="#">Start an Action Chain From a Lifecycle Event</a> .

## 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. When you create an event listener, you select the event it should listen for and the action chains you want it to trigger. You can create event listeners

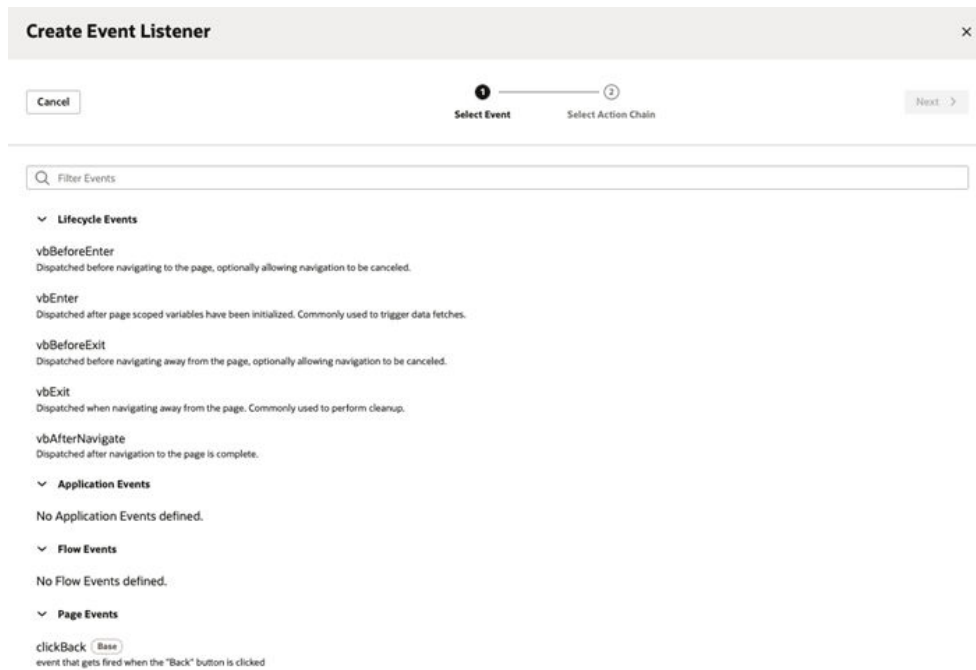
for events that you define, and for events defined in the base app which are designated as "listenable". You can also create listeners for the pre-defined lifecycle events.

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. For more, see Start an Action Chain with an Event in *Building Responsive Applications with Visual Builder Studio*.

To create an event listener:

1. Open the Event Listeners editor in the layout or page, then click **Add Event Listener**.
2. In the Create Event Listener wizard, select the event you want to trigger an action chain. Click **Next**.

The Select Event step in the Create Event Listener wizard displays a list of listenable events. The events in the list will depend upon where you are creating the listener, and might include page events, custom events and component events, in addition to lifecycle events.



3. Select the action chain you want to trigger. Click **Finish**.

The Select Action Chain step in the wizard displays the action chains that can be triggered by the listener. If you want to create a new action chain now, you can click **+** and enter an ID for the new action chain, which you can edit later in the editor.

### Create Event Listener ✕

< Back 1 Select Event 2 Select Action Chain Finish

Filter

Action Chains ⊕

New Action Chain

ID

AddToContactsActionChain

cart

getCustomerPhoto

getMissingPhoto

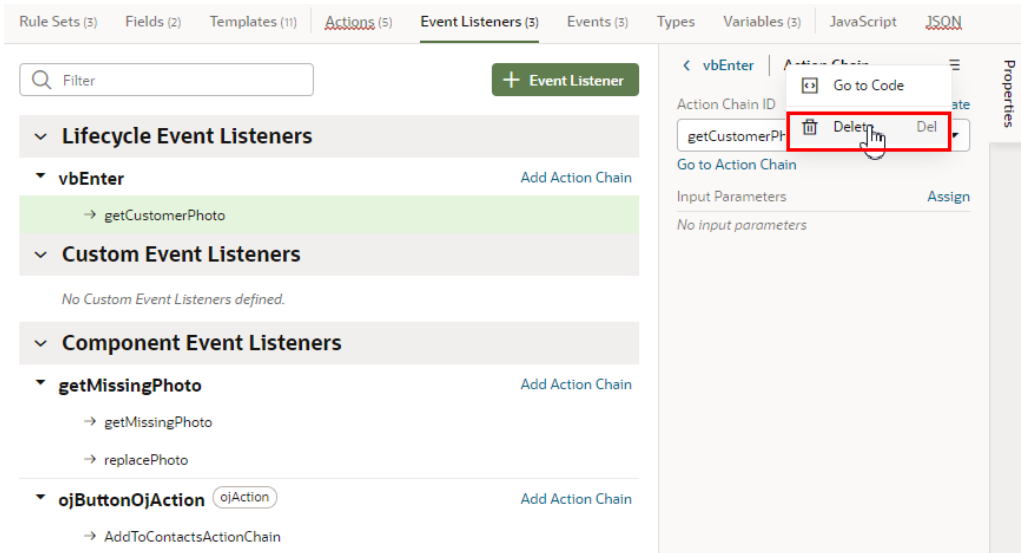
 **Note:**

An event listener can trigger multiple action chains, however, you can only add one action chain at a time in the wizard.

You can add additional action chains to an event listener at any time by clicking **Add Action Chain** for the event listener in the Event Listeners tab. Moving your cursor over the name of an action chain in the tab displays a link you can use to open the Action Chain editor.

If you want to delete an event listener, or remove an action chain triggered by an event listener, you can select it in the list in the Event Listeners tab and then click **Delete** in the options 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.





## Start an Action Chain From a Component

When you add a component to a field template, layout template or container, 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 Quick Start option in the component's Properties pane can help you create them.

There are various pre-defined events that you can apply to a component, and the events available are usually determined by the component. For example, the `ojAction` event is triggered when a button is clicked, so you would typically apply it to a button component (and you couldn't apply it to a text field component). Each button will have a unique event and an event listener listening for the button's `ojAction` event, and the listener would start an action chain (or multiple action chains) when the event occurs. Each component event will usually have a corresponding component event listener.

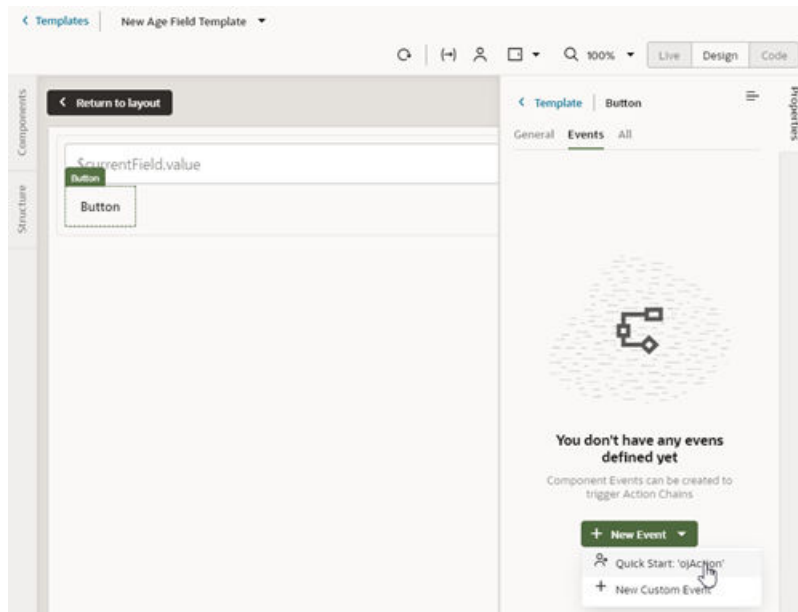
### Note:

You can only create a **component event** from a component's Properties pane. You can't create one in the Events tab of a page or layout.

To start an action with a component.

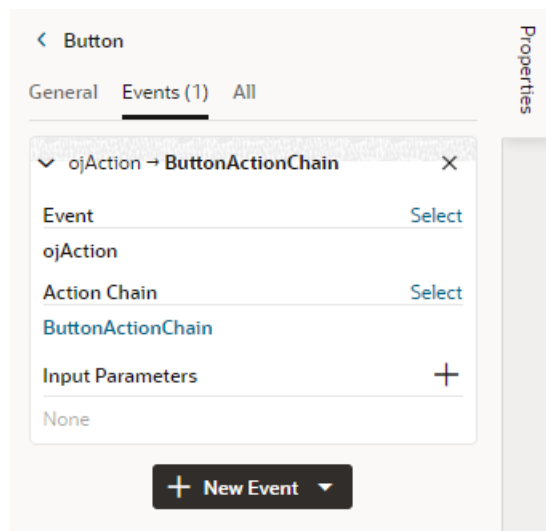
1. Select the component in the template or container section.  
You can select the component on the canvas or in the Structure view.
2. In the Events tab in the Properties pane, click **New Event** and select the suggested Quick Start option in the list.

In the New Event dropdown list, you can choose the Quick Start option or you can create a custom event. The Quick Start uses the suggested type of component event for the component, but you could choose the custom event option to use a different event.

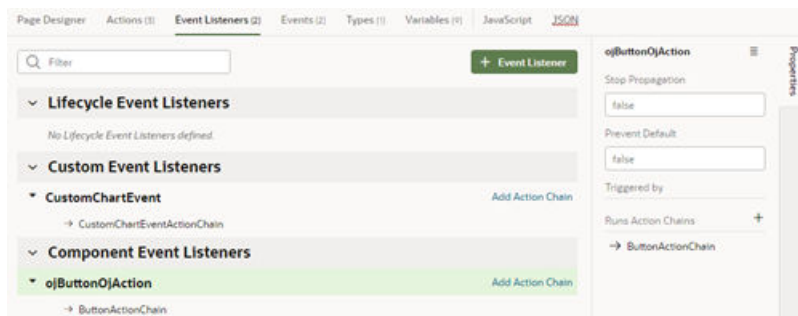


When you add the new event using the Quick Start, an action chain is created for you and the Action Chain editor opens automatically. For details on how to use the Action Chain editor to create action chains, see *Work with Actions and Action Chains* in *Building Responsive Applications with Visual Builder Studio*.

The Events tab in the Properties pane shows the properties of the component event, and you can edit the properties in the tab, for example, to add input parameters that you might want to use in the action chain.



The Quick Start creates a component event listener for the new event, and maps the listener to the action chain it created for you. If you open the Event Listeners editor, you'll see it listed under Component Event Listeners, and the action chain that it will trigger. This image shows the Event Listeners editor with the list of all the event listeners defined in a page.



## Start an Action Chain When a Variable Changes

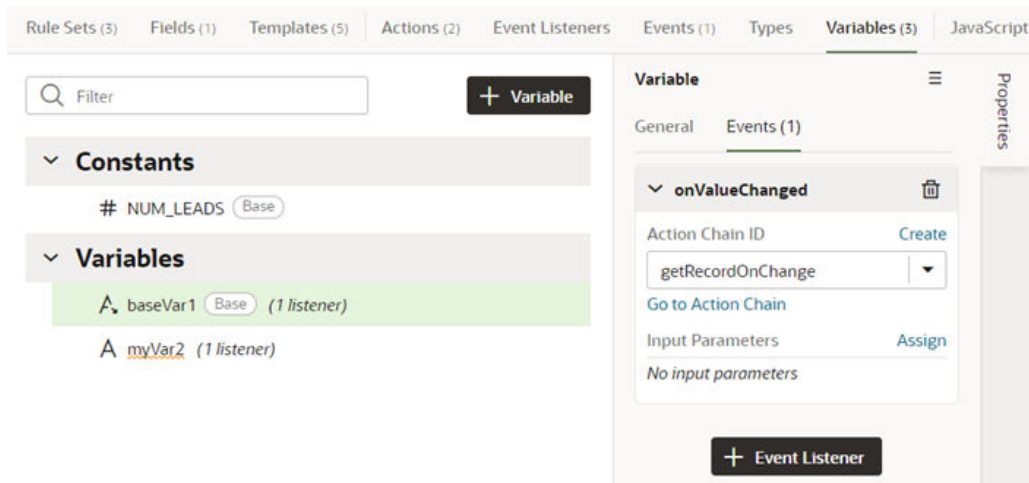
You can start an action chain when a variable changes by adding an event to the variable. For example, you might use a variable in a template to store an amount, and an action chain is triggered when a user changes the amount. You can add the event to variables defined in your app extension, and also to extendable variables defined in the base app.

To start an action chain when the value of a variable changes:

1. Open the **Variables** editor of the page or layout.
2. Select the variable in the list, then open the **Events** tab in the Properties pane.
3. Click **Add Event Listener** in the Events tab.
4. Select an action chain from the list. Click **Select**.

If you want to create a new action chain, click **+** and type the name of the action chain.

When you add the event to the variable, a variable event listener is automatically created that listens for the `onValueChanged` event on the variable. The variable's Events tab displays the action chain the event listener will trigger, and you can change or remove the action chain, assign input parameters, and add more action chains.



**Note:**

Events and events listeners for variables are not listed in the Events or Event Listeners editors.

## Start an Action Chain From a Lifecycle Event

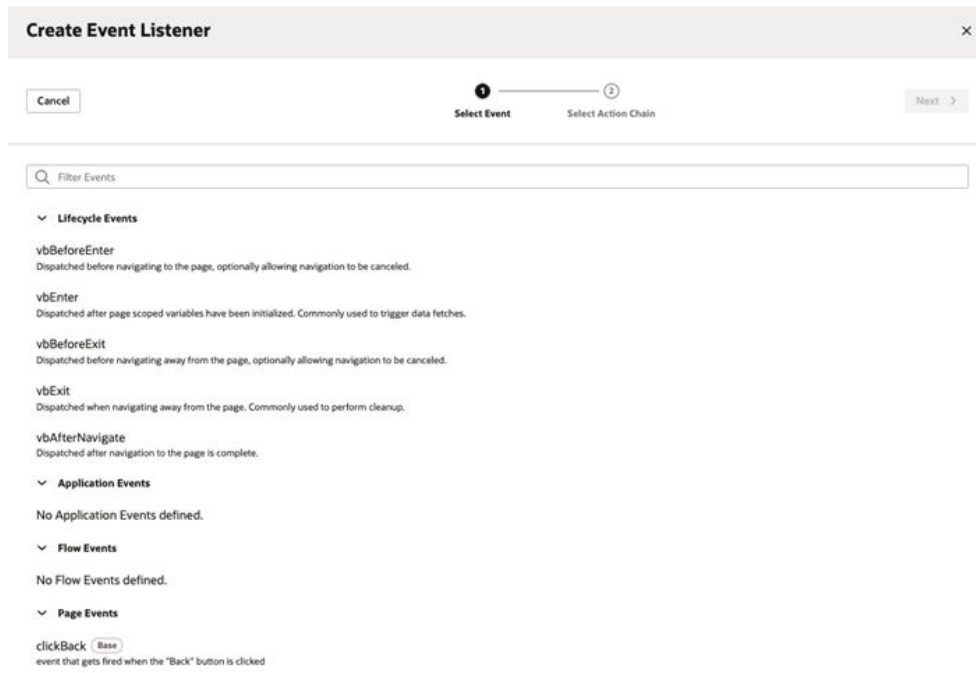
Lifecycle events are pre-defined events that occur during a page's lifecycle. You can start action chains when these events occur by creating event listeners for them. For example, if you want to initialize some dynamic component variables when the page opens, you can create an event listener in your artifact that listens for the `vbEnter` event. You could then set the event listener to trigger an action chain that assigns values to the component's variables.

This table describes the lifecycle events you can use.

Lifecycle Event	Description
<code>vbEnter</code>	Triggered after page-scoped variables have been initialized. Commonly used to trigger data fetches.
<code>vbExit</code>	Triggered when navigating away from the page. Commonly used to perform cleanup.
<code>vbBeforeEnter</code>	Triggered before navigating to the page, optionally allowing navigation to be canceled.
<code>vbBeforeExit</code>	Triggered before navigating away from the page, optionally allowing navigation to be canceled.
<code>vbAfterNavigate</code>	Triggered after navigation to the page is complete.

To start an action from a lifecycle event:

1. Open the Event Listeners editor in the layout or page, then click **Add Event Listener**.
2. In the Create Event Listener wizard, expand the Lifecycle Events category and select the event you want to trigger an action chain. Click **Next**.



3. Select the action chain you want to trigger. Click **Finish**.

## Start an Action Chain From an Action Chain

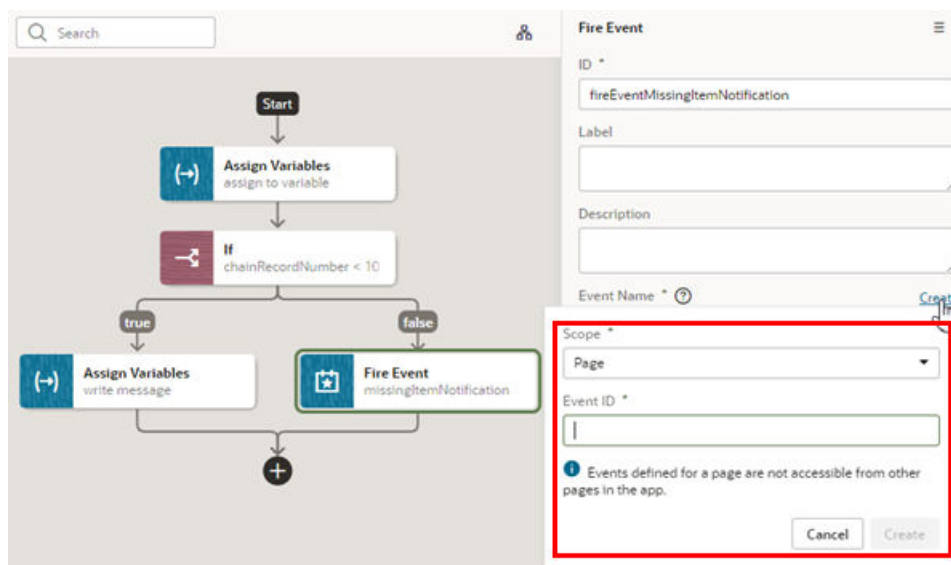
You can start an action from within another action chain using a custom event, which is triggered by the Fire Event action in an action chain. Typically you would use a custom event when you want to trigger a notification, like displaying a popup window with a message, or perhaps when you want to transform 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 Your Custom Events Call Event Listeners](#).

To start an action chain with a custom event:

1. Open the **Actions** editor in the layout or page.  
The editor shows a list of action chains that are available in the current scope.
2. Select the action chain in the list you want to edit. The action chain opens in the action chain's Diagram editor.  
If you want to create a new action chain, click **Add Action Chain**.
3. 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.
4. Select the Fire Event action in the action chain and specify the Event Name in the Properties pane.

If you are using the Fire Event action to trigger a new event, click **Create** and type 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 dropdown 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. Open the Event Listeners editor and click **Add Event Listener** to open the Create Event Listener wizard.
6. Select the custom event you added to your action chain. Click **Next**.
7. Select the action chain you want the event to trigger. Click **Finish**.

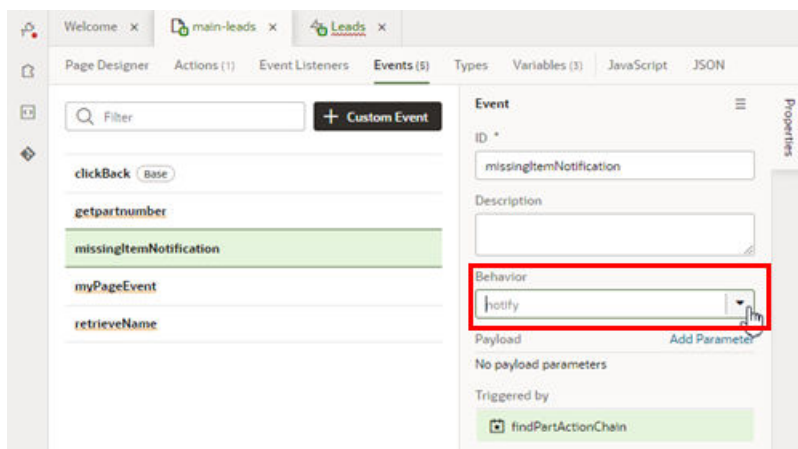
## Choose How Your Custom Events Call Event Listeners

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 is used for listeners defined in base apps as well as extensions, and is not specific to events defined in the "interface".

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	<b>Parallel</b> - The event is triggered but the application does not wait for the extension to process it. Chain results are <b>not</b> available to the Action (or helper) that fired the event (because the listeners are called without waiting). This is the <b>default</b> behavior.
NotifyAndWait	<b>Serial</b> - Each action chain listener must complete (and resolve any returned Promise, if any), before another event listener action chain is called. Chain results are <b>not</b> available to the Action (or helper) that fired the event.
CheckForCancel	<b>Serial</b> - 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 listeners Chains returns a "success" with a payload of { "stopPropagation": true }, the application will stop calling event listeners. Chain results are <b>not</b> available to the Action (or helper) that fired the event.

Behavior Type	Description
Transform	<p><b>Serial</b> - Each action chain listener must complete (and resolve any returned Promise, if any), before another event listener action chain is called.</p> <p>The "eventListener" will have access to a new context variable, <code>\$previous</code>, which is a peer of "<code>\$event</code>". This will be the result of the previous listener invocation's chain result, or undefined for the first invocation. The "eventListener" for a "transform" event can also have a "returnType" declaration, analogous to the "payloadType", but corresponding to the <code>\$previous</code> value. If the event declaration has a "returnType", <code>\$previous</code> should match the type, otherwise, it will be coerced to the type.</p> <p>When calling listeners defined in both extensions and the base application, the listeners in the "base" are called first. In other words, the base fires an event with a value, and the extensions may optionally modify that value. (This convention is the opposite order of the "cancel" behavior). The final result, "returnType", when all the listeners have been called, will be returned as the result of the <code>fireCustomEventAction</code> that initially raised the event.</p> <p>If the application wishes to use the "transform" mode, the convention it should follow is:</p> <ul style="list-style-type: none"> <li>• For any listeners for an event with a "transform" behavior that reference more than one Chain, the Chains are called in array order.</li> <li>• All listeners for an event with a "transform" behavior should pass the <code>\$previous</code> as an argument to their Chain (and this will typically be wired by the design-time).</li> <li>• All listeners for an event with a "transform" behavior should define a "returnType", which matches the "returnType" for the event declaration.</li> <li>• The Chain variable argument that corresponds to <code>\$previous</code> should have an argument that matches the "returnType" of the event, and the Chain should also have a "returnType" that matches the listeners "returnType" (note that "payloadType" and "returnType" cannot currently reference defined Types, because of restrictions on types used in the "interface" section).</li> <li>• The Chain should have a "returnType" defined, that matches the "returnType" of the event (this is currently not used by runtime.).</li> </ul> <p>The parameters for the listener, and the inputs for the Chain, to provide <code>\$previous</code> can be added by the design-time, just as it provides the <code>\$event</code>.</p> <p>Here is an example:</p> <pre> "events": {   "myTransformEvent": {     "payloadType": {       "foo": "string"     },     "returnType": {       "bar": "number",     },     "behavior": "transform"   } } "eventListeners": {   "myTransformEvent": {     "chains": [{       "chainId": "myTransformChain", </pre>

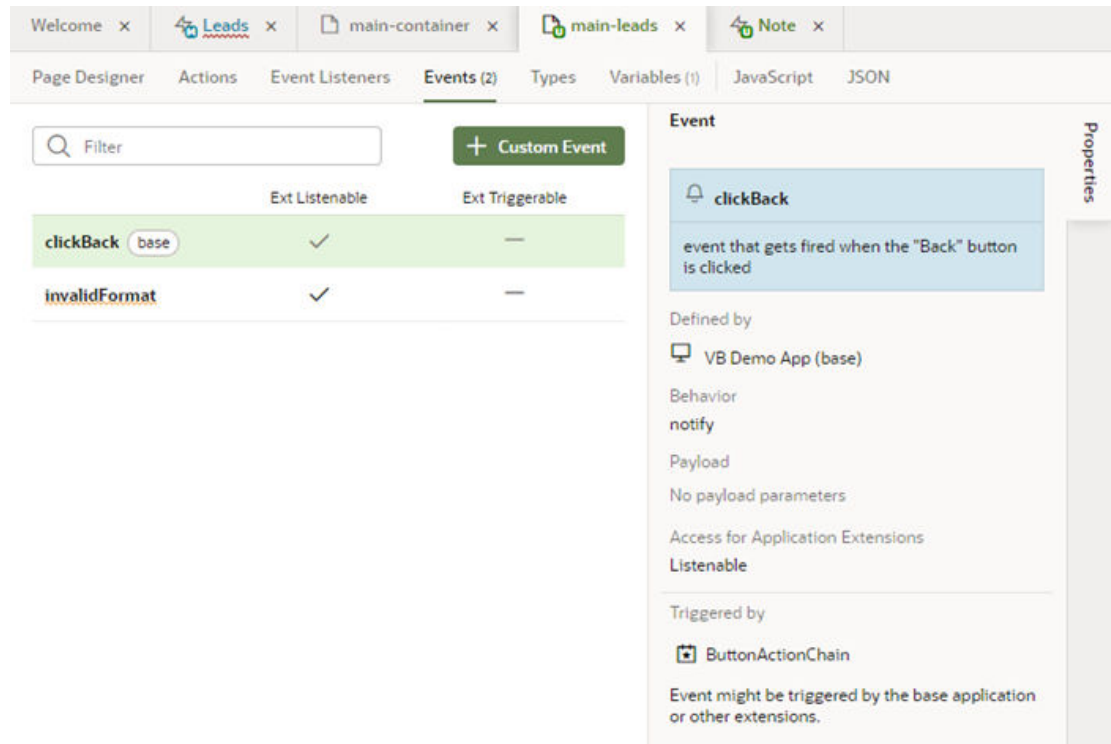


Behavior Type	Description
	<pre>         "parameters": {             "myPrevious": "{{ \$previous }}",             "myEvent": "{{ \$event }}"         }     } } }, "chains": {     "myTransformChain" : {         "variables": {             "myEvent": {                 "type": {                     "foo": "string"                 }             },             "myPrevious": {                 "type": " {                     "bar": "string"                 }             }         },         "actions": { ... },         "returnType": {             "bar": "string"         }     },     "fireIt": {         "root": "fire",         "actions": {             "fire": {                 "module": "vb/action/builtin/ fireCustomEventAction",                 "parameters": {                     "name": "{{ \$chain.variables.name }}",                     "payload": "who cares?"                 },                 "outcomes": {                     "success": "getResult",                 }             },             "getResult": {                 "module": "...",                 "parameters": {                     "name":                     "{{ \$chain.results.fire.result }}"                 }             }         }     } } } } </pre>

Behavior Type	Description
Dynamic component	The "dynamicComponent" behavior type is usable <b>only</b> in the context of an action chain invoked directly from a dynamic component. You'll need to choose this behavior type if you want to use a component in a layout to trigger action chains defined in a page, for example, to use a link in a layout to navigate to another page. This behavior is the only way a page can listen to a layout event. The event listener is defined on the component in the layout, and it must be lowercase (example: "on-myeventname"), like other Oracle JET custom event listeners.

## Use Events Defined in the Base App

The base app might define some events that you can use in your app extension to start action chains. In the Events tab of your page or layout, you can see which base app events are *Triggerable* and/or *Listenable* by your app 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	<p>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.</p> <p>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.</p>
Listenable	<p>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.</p> <p>For example, you can use listenable events to:</p> <ul style="list-style-type: none"><li>• execute an app extension action chain asynchronously or synchronously</li><li>• cancel an event in the app extension action chain</li><li>• transform a return value that is passed between the layers of event listeners</li></ul>

---

## 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 app extension. Any JavaScript code that you add will have a defined scope based on the editor where you write the code. If your code will only be called from within a specific layout artifact, for example, to assign data to a variable, you can write your code in the layout's JavaScript editor.

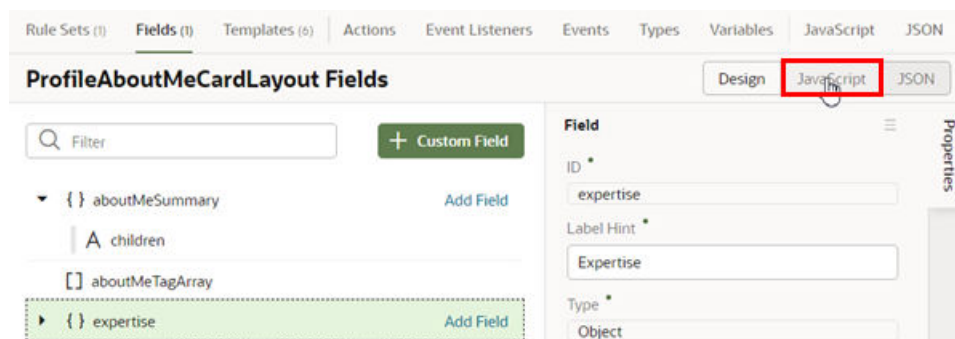
Each page and layout has a separate JavaScript file that you open in the artifact's JavaScript editor. For example, you open the JavaScript file for the `Leads` layout by opening it in the Designer and clicking the **JavaScript** tab, or by locating the layout's JavaScript file (`layout-x.js`) in the Source view in the Navigator. You can then call the functions from the layout's action chains and components.

```

1  define([], function() {
2      'use strict';
3
4      var LayoutModule = function LayoutModule() {};
5
6
7      /**
8       * @param {String} arg1
9       * @return {String}
10     */
11     LayoutModule.prototype.getSubtotal = function (arg1) {
12     };
13
14     return LayoutModule;
15 });
16

```

Layouts also have a JavaScript file for custom functions for the layout's field properties. You can edit this file by selecting a field in a layout's Fields tab and then selecting the JavaScript tab in the Properties pane, or by locating the artifact's JavaScript file (`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.

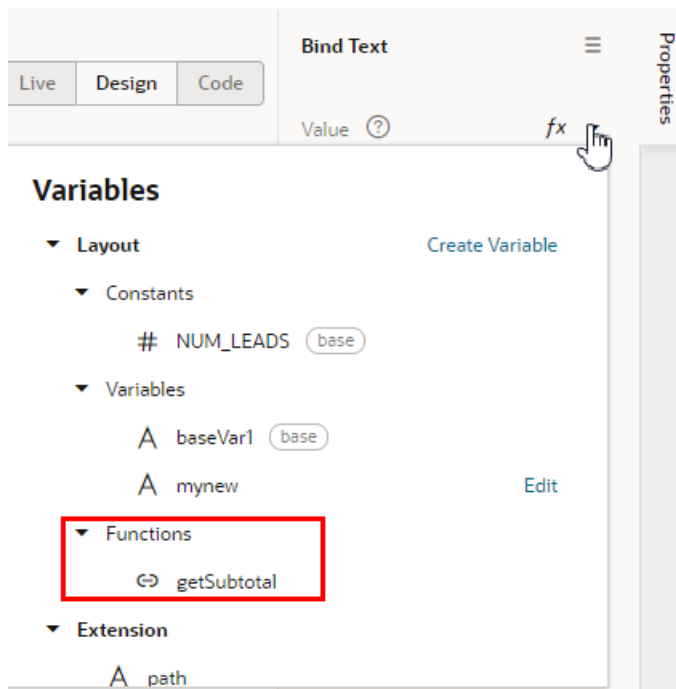


#### Note:

The auto-save function will not save a JavaScript file that has invalid code.

You can use custom JavaScript functions in your app extension:

- In an action chain, using the Call Function action. For details, see *Add a Call Function Action in Building Web and Mobile Applications with Visual Builder Studio*.
- In a component or field property, by selecting the function in the Expression editor or Variables picker in the Properties pane.

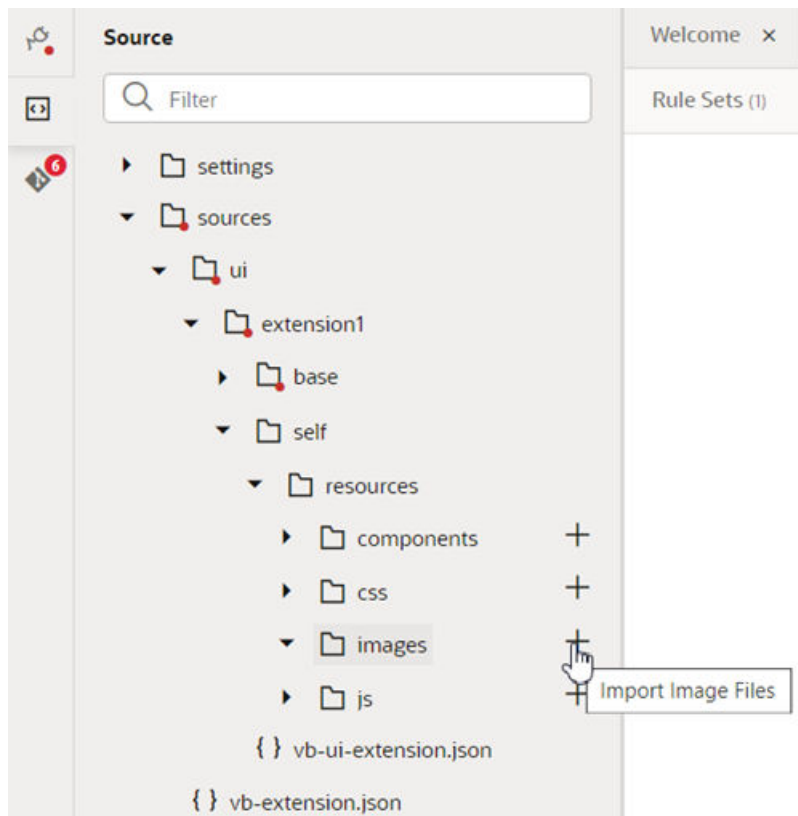


## Add Custom Resources

You can add custom resources such as images or CSS files to your app extension to replace or override resources if the base app allows it. For example, if a component uses a constant to define the path to an image, you could edit the constant to point to your custom image instead.

To add a custom image resource:

1. Open the Source View pane in the Navigator and expand the `resources` folder (your path to the folder might look like `sources/ui/extension1/self/resources`).

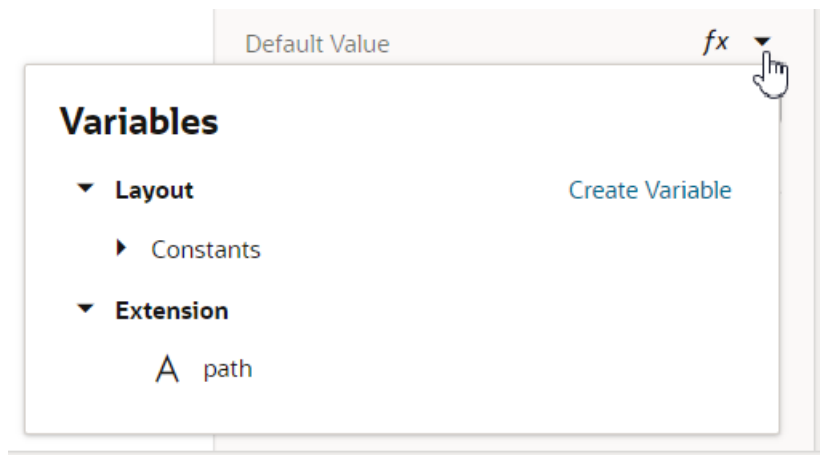


2. Click **+** to open the Import Resources dialog box.
3. Drag the resource into the drop area in the dialog box, or click the area and locate the resource on your local system. Click **Import**.  
The new resource is added to the folder and visible in Source View if you expand the folder.
4. Open the artifact containing the constant you want to edit, then open the Variables tab and select the constant that stores the path to the image.
5. Edit the constant's Default Value field in the Properties pane.

You can click *fx* to open the Expression Editor, and build the path to the image (for example, `$extension.path + 'resources/images/myNewImage.png'`).



You could also click  $\blacktriangledown$  to open the Variables picker and select the extension path, then type the location in the Default Value field.



## Add Pages To an Oracle Cloud Application

While you can't add native pages to an Oracle Cloud Application created by Oracle, you **can** create an *App UI*, then link to that App UI from your Oracle Cloud Application.

An App UI is simply an application that includes user interface components in the form of Visual Builder pages and flows. As long as your version of Oracle Cloud Applications is running 22B or later, you can build your own App UI within an extension and deploy the App UI as a peer to your Oracle Cloud Applications ecosystem.

Using Build an App UI or Fragment as a guide, you can create an App UI to contain your own pages with data from Oracle Cloud Application REST APIs, or even from custom business objects. By default your App UI will be available from Oracle's new Ask Oracle menu, but you might also choose to make it available from the Oracle Cloud Application itself. From a customizable component on your original Oracle Cloud Application page, like a dynamic container, you might add a link to the new App UI in the form of an icon or hyperlink, or create a button to navigate to the new content when it's clicked.

# 6

## Preview and Share Your Application Extension

The Preview and Share actions enable you and other team members to see how your app extension will look and behave in the Oracle Cloud Application without building and deploying it to your development environment.

### Preview Your Application

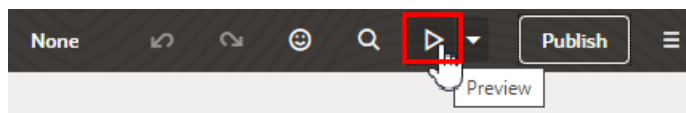
Before committing your changes to the branch, you can use the Preview action to see how your pages look and behave in a browser.

As the name implies, Preview shows you how your pages are displayed and gives you a chance to test the behavior of your app extension before sharing it with others. In the preview you can check the rule sets, layouts, the navigation between pages, and what happens when you add and edit data. While the data in the preview comes from your Cloud Application, you can freely make changes in the preview because the data is only stored in the test environment used for the preview. So you could test a form and edit fields and it wouldn't affect the data stored in the Oracle Cloud Application's data storage.

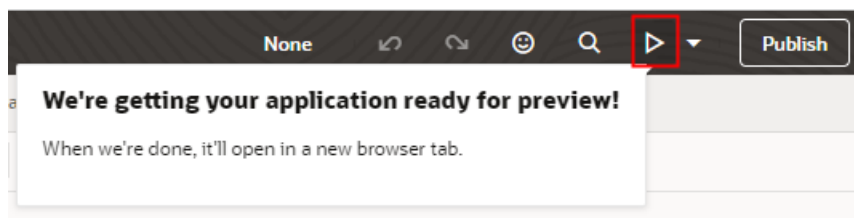
When you use Preview to view your application, you'll see the pages as they look based on your user role. This means that if the application has any rule sets that evaluate the user's role, this will affect the data you see in the pages. If you want to apply a different layout, you'll have to exit Preview mode and follow the instructions in [Preview Different Dynamic Layouts](#).

To preview your application:

1. Open a page in your application.
2. Click **Preview** in the header to open the application preview in your browser.



When you click Preview, you'll see a notification to let you know that the preview will soon be ready:



The application preview opens to the page you were working on in the Page Designer.

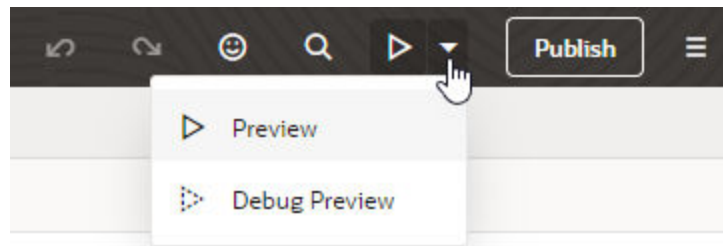


You can't share an application preview URL with your team members. If you want team members to see your changes, you'll need to commit the changes to a branch and then use the Share action to get a URL you can share.

## Preview in Debug Mode

When testing your app extension, you typically preview it to see it the way your user would. Sometimes though, you might want to preview the app extension in debug mode to troubleshoot issues with the Visual Builder runtime and Oracle JET libraries—runtime dependencies that make sure your app works as intended.



The **Preview** option in the header facilitates both modes:



The default Preview mode uses the optimized variants of the VB runtime and JET libraries. In this mode, all unnecessary characters (such as whitespaces and comments) in the application's source code are removed and variable names shortened to minify code and optimize performance.

The Debug Preview mode, on the other hand, uses the debug variants of the VB runtime and JET libraries. In this mode, line breaks and white spaces are preserved, allowing you to view the application's source code in a more readable format. You can then use your browser's debugging tools to step through your code and figure out exactly where an error occurred.

Because no performance optimization is done in debug mode, previewing an app in this mode can be misleading about how quickly—or slowly—an app opens. As a result, you might want to use debug mode only in a development environment.

- To view your app like a user would with the pages and data displayed, click .
- To view your app in debug mode, click the Preview drop-down, select Debug Preview, then click .

## Share Your Application Extension

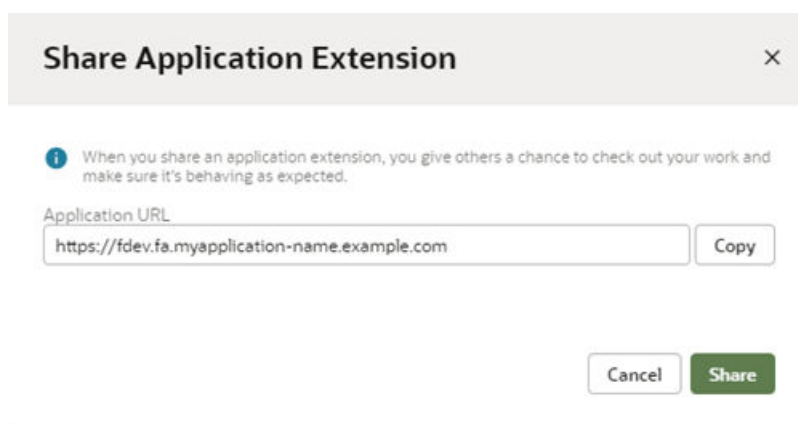
Application previews can only be seen by you in your workspace, but you can use the Share action to share your application extension with others. The Share action generates a URL for the application extension that you can share with your team members.

If you made changes to the app extension that involved changes to the data model, when you share the URL you might need to also share the name of the sandbox so your team members can check the changes in the app extension using the correct data model. The URL of the application extension you are sharing might not be the

same as the URL you see when you are previewing the application extension in the Designer.

To share your edited application extension:

1. Open your workspace in the Designer.
2. If you are using a sandbox, confirm that your current sandbox is the one you want to share.
3. Open the User Menu and click **Share**.



4. Click **Copy** to copy the generated URL for the application extension.

If you are using a sandbox, you will need to share the URL and the name of the sandbox with your team members.

5. Click **Share**.
6. Send an email with the generated URL, and include the name of the sandbox if one was used.

## Export Your Workspace as an Archive

If you need to share your code for your app extension 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.

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

To export an app extension as an archive:

1. In VB Studio's left navigator, click **Workspaces**.
2. Locate the workspace that you want to export, then select **Export Application Extension** in the workspace's options menu.

# 7

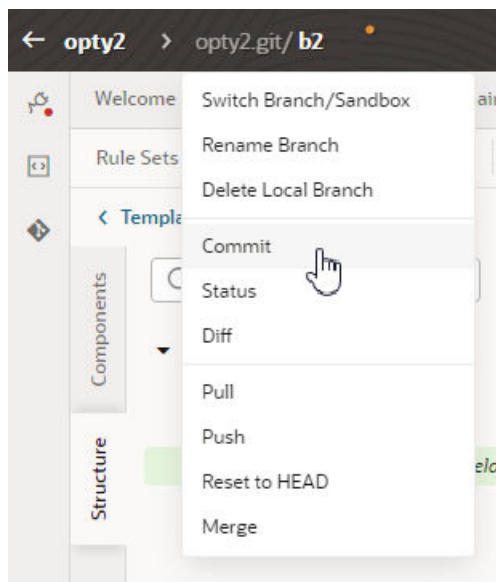
## Publish the Application Extension

To publish your app extension, you first save your local changes to a remote branch, then merge that branch into the main branch. You can do all these steps at one time using the Publish action, or you can use Git commands in your workspace to do them individually.

### Review the Process

If you're unfamiliar with Git commands, it may be helpful to review the relationship between your workspace, your project, and the Oracle Cloud Applications development environment.

- In most cases, the trunk or *main* branch of your project's Git repository is the source from which your artifacts (like an app extension) are built. Your project's Git repository is also called the *remote repository*, while the clone in your workspace is called the *local repository*.
- When you create a workspace (or one is created for you), you get a clone of the project's Git repository and all its *branches*. Branches are usually created from the main branch, but might also be created from a different branch. Your workspace will contain a branch (which might be new or might already exist), but no one else can see the changes you make to the branch in your workspace until you choose to make them visible. This local branch is where you work on your application extension.
- As you make your changes, you frequently use the *Commit* action from the Designer's drop-down menu to mark the changes in your local repository branch that you want to copy to the remote repository:

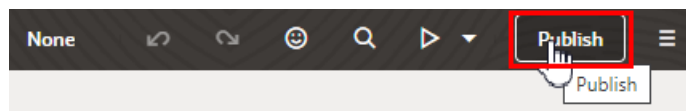


- You can also add a commit message describing the reason you updated those files (for example, "Added new layout X for field Y"). These commit messages can help you later to identify which files were changed and why.

- You use the *Push* action from the Designer's drop-down menu to copy changes in your branch in the local repository to the branch in the remote repository. All the changes that you've committed since your previous push are copied to the branch in the remote repository.
- Once your changes are pushed to the remote branch they are available to everyone else in your project. Your team members can clone the branch in their workspaces and view the changes. They can also view the source code in the repository, including all branches, by opening Git in VB Studio.
- If other team members are working on the same application extension you are, you might all be working off the same branch. For this reason, you should update the remote branch with your changes frequently using the Commit and Push commands.
- You use the *Pull* action to update your local branch with any changes that other members push into the remote branch. It's a good idea to do this frequently to ensure that the branch in your workspace has the most recent changes. This can help you to avoid accidentally 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.
- When you're completely done with your work and you've pushed all your changes into the remote branch, the next step is to *merge* the changes in the remote branch into the source of truth, the main branch. You might be required to submit a *merge request* before the merge is permitted. A merge request describes the reason and scope of the changes to be merged, and often requires some team members to explicitly approve the request after they have reviewed the changes.
- You can use **Publish** in the Designer's header when you're ready to publish the work you've done in the private branch in your local repository. The Publish action commits any uncommitted changes, pushes the changes into the remote branch, and then either merges the branch into the main branch or creates a merge request.

## Publish the Application Extension

The easiest way to publish your changes is by opening the Publish Changes dialog box from the header in the Designer. The Publish operation combines Git operations (commit, push, merge) to merge the changes in your local repository branch into the main branch. In the dialog box you simply enter a commit message describing the changes (and add reviewers and issue if you're also creating a merge review request) and click Publish. If you don't create a merge review request, the changes are merged immediately.



Depending on how your project is configured, you might be able to publish your changes without creating a merge review request, but it's good practice to ask your team members to review the changes you made before you publish it. A code review is usually required before changes are merged from a branch into the main branch.

If you create a merge review request, the changes in the branch can be merged into main branch after they've been reviewed and approved in the Merge Requests page in VB Studio. For the steps to do this, see [Merge Remote Branches](#).

 **Note:**

Don't use the Publish action if you aren't ready to merge the remote branch into main branch. You can't use a remote branch after it's been merged into the main branch.

When you're ready to merge your changes, publish the sandbox you used in your Oracle Cloud Application, if you used one. Publishing the sandbox before merging your application extension changes to the main repository 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*.

 **Note:**

When you change your data model in App Composer, you need to publish those changes before you can use them in your app extension. For example, if you create some custom fields in App Composer, an app extension published to a test environment can use the new custom fields after the new data model is published to the test environment. When you publish your app extension to a different environment, for example, your production environment, you'll also need to use App Composer to publish the new data model to the production environment.

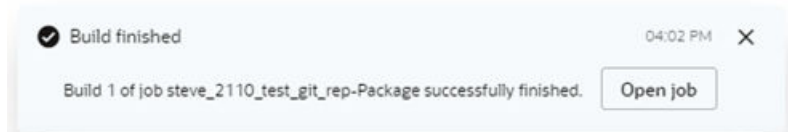
Once you merge a branch into the main branch, you can no longer use it (or your local copy of it). A new branch is created automatically for you when you publish your changes, and you can use it to make additional changes or you can create your own branch.

Your project is probably set up so that each time changes are merged into the master branch, the application extension artifacts are built from the repository and automatically deployed to the Oracle Cloud Application's test environment for further tests.

 **Note:**

To deploy your application extension, the pipeline's deployment job must be configured with the credentials of a user who has permissions to connect and deploy to the Oracle Cloud Application environment. (These **must** be Oracle Cloud Application credentials, as opposed to VB Studio or SSO credentials.) The privilege granting permission to deploy is `FND_ADMINISTER_SANDBOX_PRIV`. If your project owner hasn't provided this information, you'll be prompted to provide it each time you click **Publish**. If your credentials don't allow you to deploy to the Visual Builder instance, talk to your project owner or an administrator to get credentials that you can use. The credentials that you enter when prompted, if valid, will be permanently saved in the deployment job. Your administrator can also choose to enter the credentials directly in the build job, so you aren't prompted for them when you try to deploy your app. You won't be able to publish without these credentials. If your user credentials don't allow you to deploy to the environment, talk to your project owner or an administrator. See *Configure the Deployment Job* in *Administering Visual Builder Studio* for details on how an admin can add the missing credentials.

You'll see notifications like the following in your workspace about the status of your jobs, including when a job has started, when it has finished, and when one fails.



## Publish Changes Without Creating a Review

If your project is configured to allow you to merge changes from your branch without creating a merge review request, you can use the Merge Now tab in the Publish Changes dialog box.

To publish your application extension without creating a merge review request:

1. Click Publish in the Designer toolbar.
2. Select the **Merge Now** tab and type a commit message.

### Publish Changes ×

You are about to publish your changes, which means they will be deployed to the **Development** environment. This might take a few moments.

Merge Now Merge After Review

**⚠** Some build jobs, associated with the merge action of your master branch, are missing authorization credentials. These build jobs will fail until fixed.  
[opty2-Deploy](#)

Commit Message: \*

▼ Changed Files (1)

[M sources/ui/extension1/base/dynamicLayouts/leads/layout-x.json](#)

Cancel Publish Changes

3. Click **Publish Changes**.
4. Click **Close**.

## Publish Changes that Require Review

If you don't see the Merge Now option in the Publish Changes dialog, that means that a merge request is required before you can merge your changes. If you see both tabs, that means the option to create a merge request is yours. If your project requires a review before changes can be merged, the Publish Changes dialog will not have the Merge Now tab.

When you create a merge request you can specify which team members you want to review the changes. The requirements of the merge request will depend on how your project is set up, for example, a certain member might be required to approve the request before the changes can be merged. For details on how to work with merge requests, see Review Source Code with Merge Requests in *Using Visual Builder Studio*.

To create a merge request in the Publish Changes dialog box:

1. Click **Publish** in the Designer toolbar.
2. Select the **Merge After Review** tab and type a commit message.

## Publish Changes ×

You are about to publish your changes, which means they will be deployed to the **Development** environment. This might take a few moments.

Commit Message: \*

▼ Changed Files (1)

**M** sources/ui/extension1/base/dynamicLayouts/leads/layout-x.json

Reviewers \*

Linked Issues

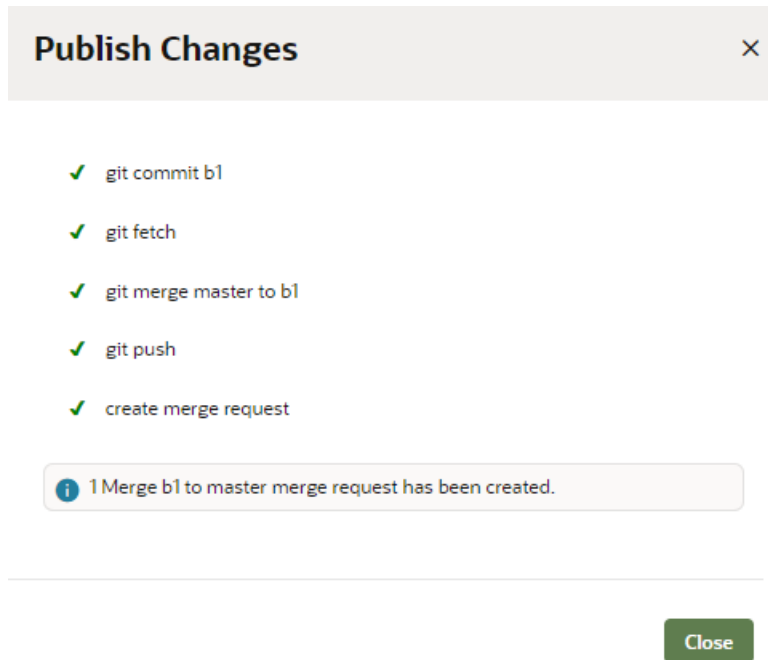
3. Select the team members you want to review your changes.

The members you select will be listed in the Merge Requests page as reviewers. A reviewer can approve or reject a merge request, but it is not required. You can merge the changes even if they have not been reviewed or approved.

4. Select the VB Studio issues that you want to link to the merge review request, assuming that your project is using them.
5. Click **Publish Changes**.

When you click Publish Changes, the dialog box displays the Git commands that are automatically performed for you.





6. Click **Close**.

After the reviewers have reviewed and approved the changes, go to the Merge Requests page in VB Studio and use the Merge action to merge the changes into the main branch. For the steps to do this, see [Merge Remote Branches](#).

## Video: Overview of Your Git Repositories

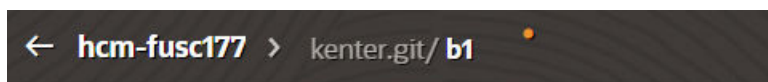
This video illustrates the relationship between your workspace and the Git repository in your project.



## How Do I Add My Changes to a Remote Branch?

After you edit files in your workspace you need to save them to a remote branch in order to share the changes in your app extension with your team members. Until you save those changes to a remote branch, they only exist in the local branch in your workspace, which means that only you can see them.

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



You can also open the Git panel in the Navigator to view all the changes in your workspace that haven't been committed.

Saving your changes to a remote branch is a two-step process: **commit** and **push**. The first step you must perform is a **commit**. When you make a commit, you are basically grouping the files in your local branch that you want to save to the remote branch, and providing a description of the group. This may seem redundant, but it helps document the process of saving files to the repository, which can be very helpful later if you need to track down which files were changed.

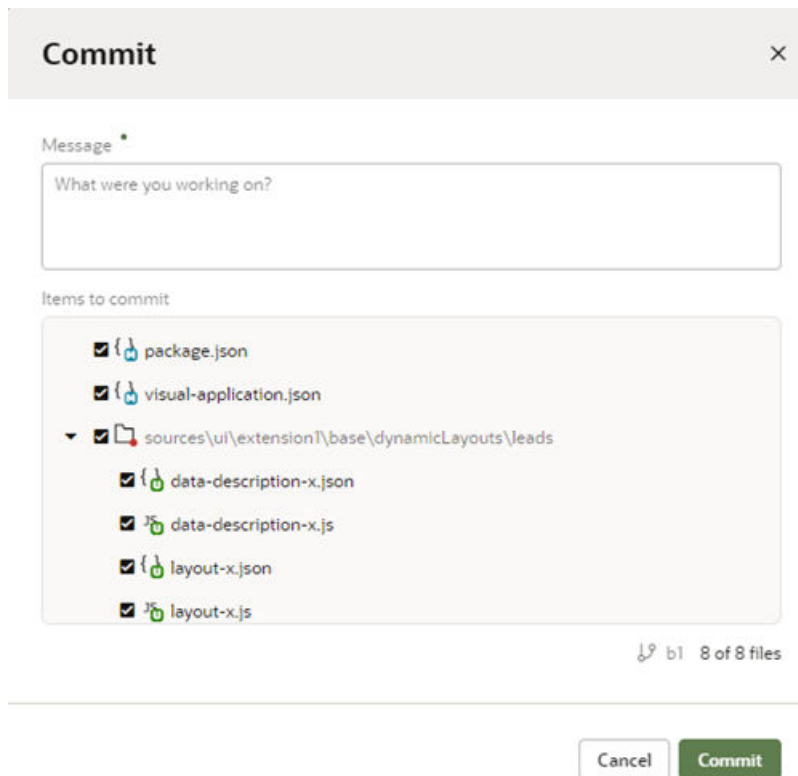
For example, if you make some changes to add a new layout to include a field, when you "commit" the changes you can add a commit message describing the reason you updated those files ("Added new layout X for field Y"). You then might change some other files to add a rule for the new layout, and add a commit message describing those changes ("Added rule Z for layout X"). When you eventually save these files to the repository, those commit messages can help you identify which files were changed and why. When you look at the list of changes made to the repository, you can easily see what files were changed and the reasons for changing them.

The second step is **push**, which saves all the files in the groups that you've "committed" to the remote branch, so a single "push" might save several groups of files. If you have edited some files but haven't committed them, you are prompted to commit them when you "push" your changes to the repository.

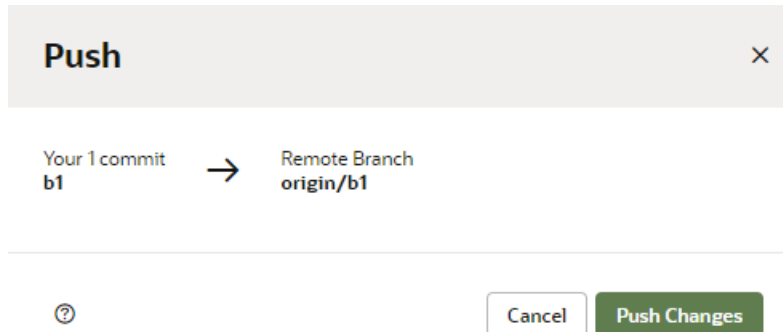
Files and artifacts can have the status Edited, Committed, or Pushed (the default status). The name of each file in the Navigator is color-coded to indicate its status.

To commit and push changes from your local branch to a remote branch:

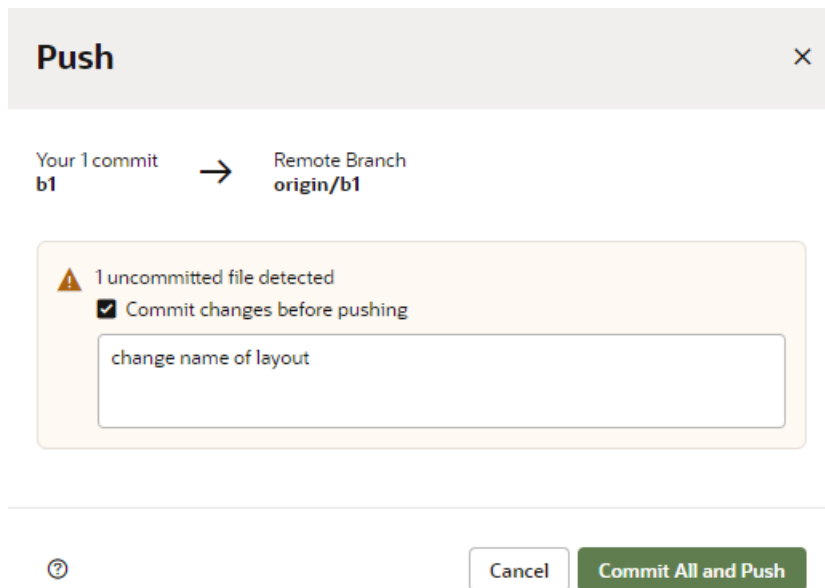
1. Click the branch name in the toolbar and select **Commit** in the menu.
2. Enter a commit message to describe the changes you are committing. Click **Commit**.



3. Click the branch name in the toolbar and select **Push** in the menu.
4. Click **Push Changes** in the Push dialog box.  
The Push dialog box shows the names of the local branch with your changes and the remote branch where they'll be pushed to.



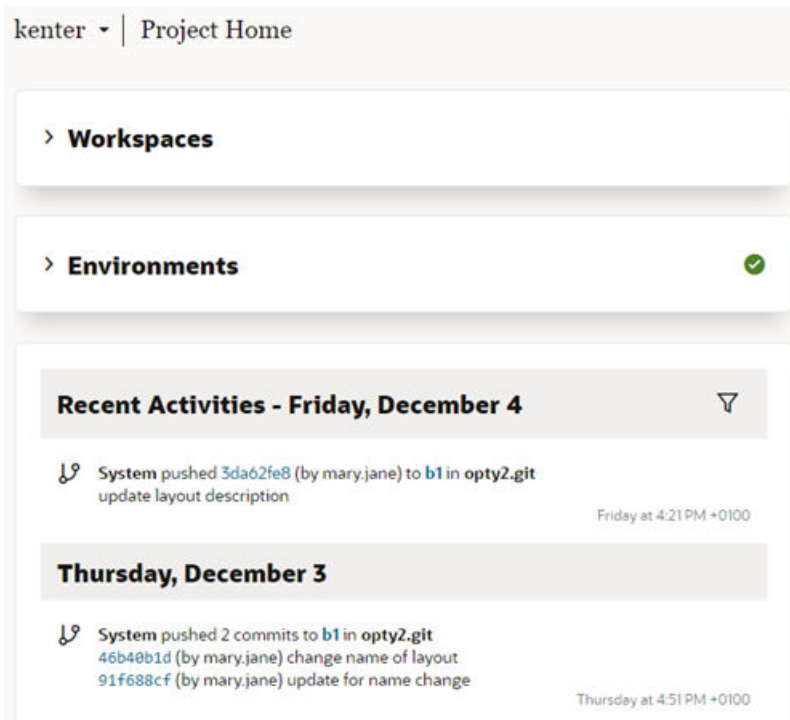
You'll see a message in the Push dialog box if any of your changes are still uncommitted when you attempt to push your changes. You can enter a commit message in the dialog box and click **Commit All and Push**.



Pushing your changes adds all the files that you've committed since your previous push to the remote branch.

A notification is displayed in the Designer each time you commit and push changes.

You can see the changes that were added to the remote branch in the Recent Activities panel on the Project Home page. The panel lists the commit message for each commit included in a push to the remote branch.



After you push your changed files to the branch in the remote repository, you can merge the remote branch into another branch, or into the main branch.

To merge a remote branch into a branch that isn't the main branch, use the Merge Requests tool in VB Studio. For details on how to create and work with merge requests, see Review Source Code with Merge Requests in *Using Visual Builder Studio*.

#### Note:

You can use the Publish action in the Designer header if you want to commit, push changes to a remote branch and merge the remote branch into the main branch in one operation. Don't use the Publish action if you aren't ready to merge the remote branch into the main branch.

## Merge Remote Branches

You use the Merge Requests tool in VB Studio to merge remote branches, for example, when you want to merge a remote branch with your changes into the main branch. Your team members can use the tool to review the changes you want to merge into the main branch, 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.

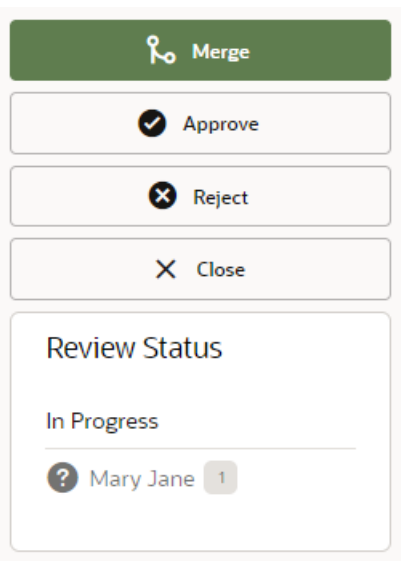
A merge request is created automatically when you use the Merge After Review option in the Publish Changes dialog box in the Designer. To publish the app extension, you'll need to open the Merge Requests page in VB Studio and merge the branch into the main branch.

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 VB Studio left navigator and open the merge request.
2. On the right side of the page, click **Merge**.



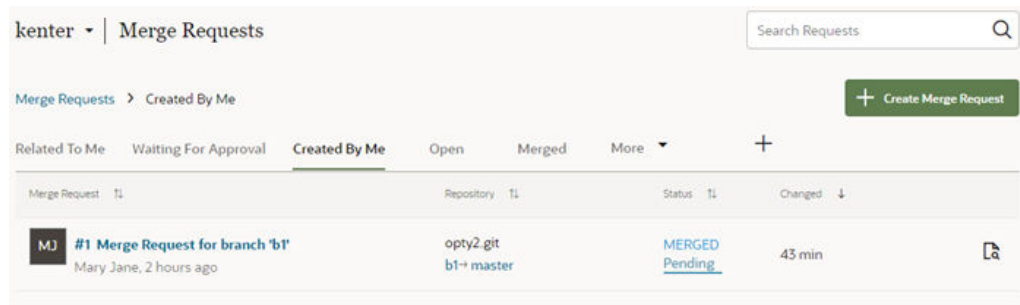
3. In the Merge dialog box, type a description of the merge and click **Create a Merge Commit**.

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.



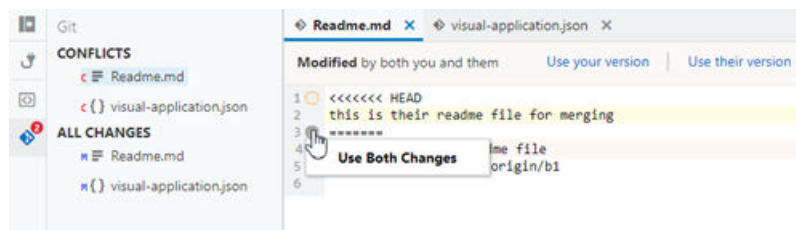
## Keep Your Local and Remote Branches in Sync to Avoid Conflicts

If you and another team member are working together on an app extension, you will be using the same Git repository and might be working off the same branch and editing the same files. This can result in a code conflict, where you and a team member make changes to the same line of code in a file.

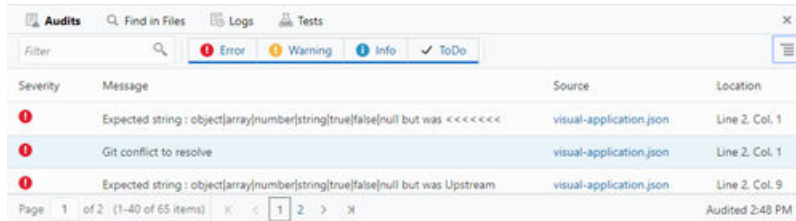
To help minimize the number of conflicts, the best practice is to make sure your local branch and the remote branch are in sync and up to date. You do this by committing and pushing often. Each time you commit or push changes to the branch, Git will check the remote branch to see if there were any updates since the last time you checked, and update your local branch with the changes.

If a code conflict occurs, you'll have to resolve it before you can update the branch with your changes. VB Studio will let you know if there is a conflict when you try to commit or merge your changes, and you can use the conflict resolution editor to fix the problems in each file by selecting whose changes you want to keep. You can continue when all the conflicts are resolved.

For details on resolving merge conflicts when merging a review branch, see *Resolove a Merge Conflict in Using Visual Builder Studio*.



Use the Audits pane in the Designer to scan and locate places in your code that are flagged as errors because a conflict has not been resolved.



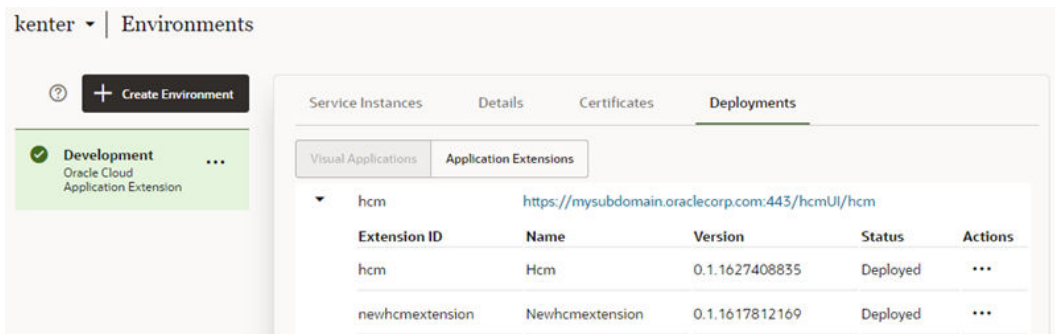
## View Your App Extension Deployments

After you've published your app extension, you can check your project's Environments tab to see if it's been deployed to your development environment.

To view the deployed app extensions for your base application:

1. Click **Environments** in the VB Studio left navigator to open your project's Environments page.
2. Select the development environment for your app extension.  
App extension projects usually only have one development environment that the extensions are deployed to.
3. Open the **Deployments** tab for your environment and click **Application Extensions**.
4. Expand the node for your base application to see a list of deployed app extensions.

There might be several app extensions deployed to your base application, so you'll need to look for the extension id and name in the list of deployed extensions to confirm that yours has been deployed.



If you think your app extension should have been deployed but you can't find it in the list, check the build jobs for your project to check if the job has finished, or contact your project administrator if you think there might be a problem with deploying the app extension.

# 8

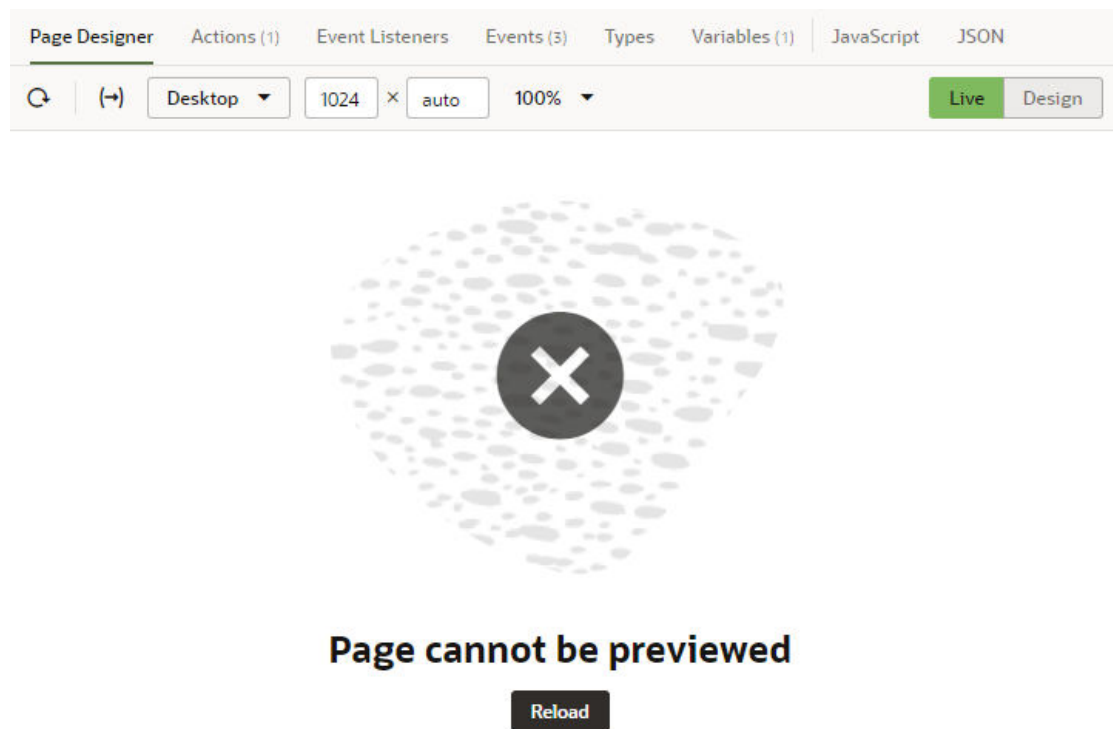
## Troubleshooting

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:



To resolve this page preview error, you'll need to edit some of your Chrome browser settings.

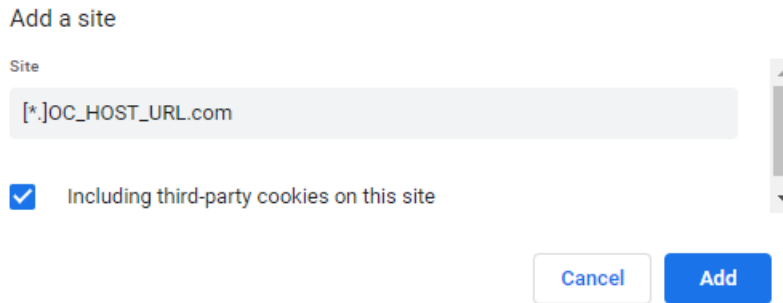
1. Note the host domain of your Oracle Cloud Application instance.  
For example, if the URL of your Oracle Cloud Application is something like `osl.mysubdomain.OC_HOST_URL.com`, the host domain is `OC_HOST_URL.com`.
2. Open the browser's Cookies settings page (`chrome://settings/cookies`) and disable **Block third-party cookies**, if it's not already disabled.

You can select **Allow all cookies** to disable the "Block third-party cookies" option.

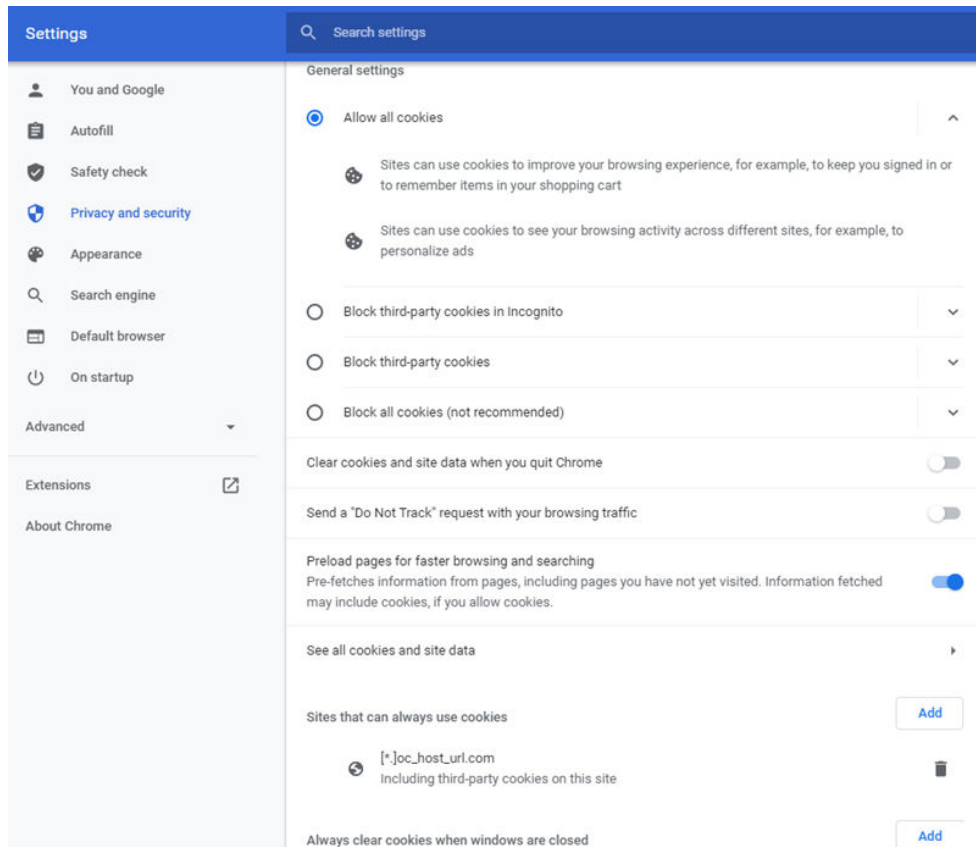


3. Click **Add** next to the "Sites that can always use cookies" option, then type your Oracle Cloud Application host domain in the Add a site dialog box and enable **Including third-party cookies on this site**. Click **Add**.

Use the syntax `[*.]OC_HOST_URL.com` for your host domain in the dialog box to allow cookies from your host's subdomains.

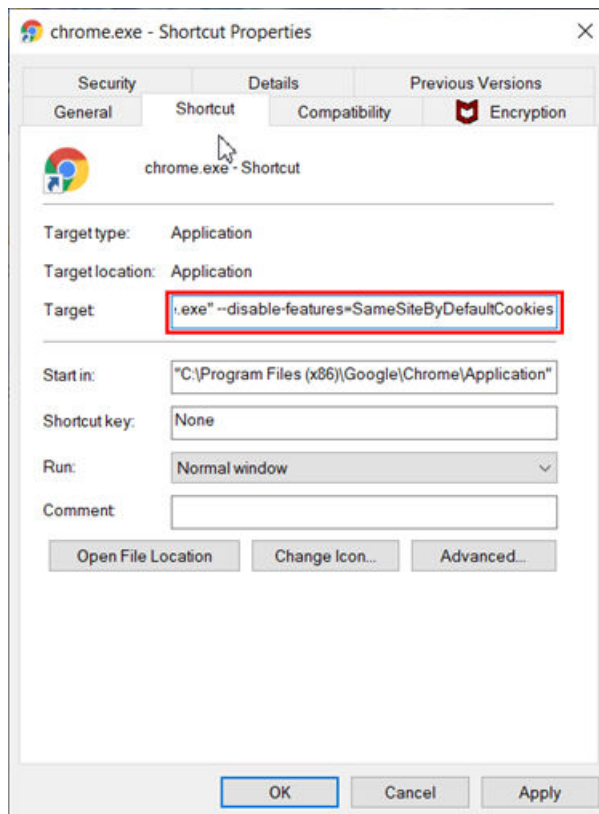


Confirm that your host domain is included in the list of sites that can always use cookies.



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.

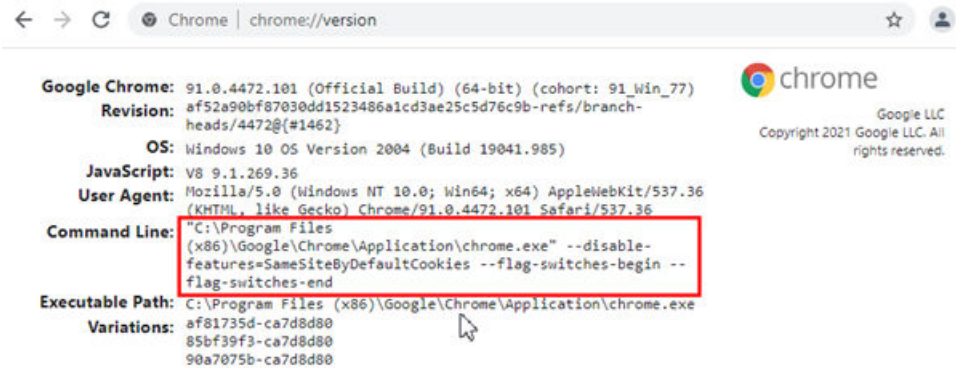


If you're using Chrome on MacOS, you can launch Chrome with the parameters from a Terminal window. To launch Chrome, open a Terminal window and type `'/Applications/Google Chrome.app/Contents/MacOS/Google Chrome' --disable-features=SameSiteByDefaultCookies`.

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 `chrome://version/` in your browser and check the Command Line properties to confirm that "SameSite by default cookies" is disabled.



← → ↻ Chrome | chrome://version ☆ 👤

**Google Chrome:** 91.0.4472.101 (Official Build) (64-bit) (cohort: 91\_Win\_77)  
**Revision:** af52a90bf87030dd1523486a1cd3ae25c5d76c9b-refs/branch-heads/4472@{#1462}  
**OS:** Windows 10 OS Version 2004 (Build 19041.985)  
**JavaScript:** V8 9.1.269.36  
**User Agent:** Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.101 Safari/537.36  
**Command Line:** "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --disable-features=SameSiteByDefaultCookies --flag-switches-begin --flag-switches-end  
**Executable Path:** C:\Program Files (x86)\Google\Chrome\Application\chrome.exe  
**Variations:** af81735d-ca7d8d80  
85bf39f3-ca7d8d80  
90a7075b-ca7d8d80

 **chrome**  
Google LLC  
Copyright 2021 Google LLC. All rights reserved.