

Oracle Fusion Cloud Applications

Configuring Applications Using Application Composer

25A



Oracle Fusion Cloud Applications
Configuring Applications Using Application Composer

25A

G17875-01

Copyright © 2011, 2023, Oracle and/or its affiliates.

Author: Iqtidar Hussain

Contents

Get Help	i
<hr/>	
1 About This Guide	1
<hr/>	
Audience and Scope	1
Related Guides	2
Additional Common and Advanced Setup Tasks	2
2 Before You Begin	5
<hr/>	
Get Started with Application Composer	5
About Application Composer	5
What's Required for Testing Configurations in the Sandbox	7
Enable Sales Administrators to Test Configurations in the Sandbox	8
Assign Yourself Additional Job Roles Required for Testing	9
3 Add Objects and Fields	11
<hr/>	
Overview of Using Application Composer	11
Objects	12
Security for Custom Objects	25
Fields	37
How do I enable tagging of service requests?	91
Actions and Links	93
Overview of Smart Actions	97
Direct Page Links for CX Sales	105
Deep Links for Service Requests	114
How to View Application Composer Changes	118
How to View a Diagnostic Report of Your Application Composer Changes	120
Import and Export Custom Objects	124
FAQs for Using Application Composer	125
4 Extend Application Pages	131
<hr/>	
Overview of Application Pages	131

Modify Application Pages Using Application Composer	132
Application Pages for Standard Objects	132
Application Pages for Custom Objects	132
Search and Select Dialog Boxes	134
Dynamic Page Layouts	139
Configure the Summary Table on a Landing Page	152
Enable or Disable Drill Down Fields in Application Pages	153
Specify Drill-Down Fields for Custom Dynamic Choice List Fields	154
Create and Add Custom Links to Application Pages	155
Subtabs	156
Mashups	173
Modify Work Area Lists	187
FAQs for Extending Application Pages	191
5 Groovy Scripts	193
Overview of Groovy Scripts	193
Groovy Scripting	194
Server Scripts	196
Object Functions	201
Global Functions	207
View Objects in Scripts	210
Classes and Methods Supported in Groovy Scripts	214
Groovy Scripting Examples	231
Call REST Web Services from Groovy Scripts	236
Call SOAP Web Services from Groovy Scripts	254
Groovy Performance and Web Services	274
Using Fields in Web Service Requests	275
Runtime Messages	276
Debug Your Groovy Scripts	278
FAQ for Using Groovy Scripts	280
6 Object Workflows	283
Overview	283
About Object Workflows	283
Commonly Used Object Workflow Terms	284
Expression Builder for Object Workflows	286
Examples of Object Workflows	287

Update Fields Using Object Workflows	290
Object Workflows and Email Notifications	291
Object Workflows and Tasks	296
Object Workflows and Outbound Messages	299
Object Workflows and Business Processes	304
Object Workflows and Groovy Scripts	333
Configure Object Workflows	337
Configure Notification Delivery Methods from Object Workflow	345
Delete Unpublished Object Workflows	346
FAQs for Creating Object Workflows	347
7 Custom Subject Areas	349
About Custom Subject Areas	349
Custom Subject Area Objects	350
Custom Subject Area Fields	352
Measures	354
Custom Subject Areas Dates	356
Secure Custom Subject Areas	358
Create Custom Subject Areas	358
Publication Statuses of Custom Subject Areas	363
How to Report on Custom Fields	364
Hierarchies in Analytics	364
Custom Subject Areas: Frequently Asked Questions	365

Get Help

There are a number of ways to learn more about your product and interact with Oracle and other users.

Get Help in the Applications

Some application pages have help icons  to give you access to contextual help. If you don't see any help icons on your page, click your user image or name in the global header and select Show Help Icons. If the page has contextual help, help icons will appear.

Get Support

You can get support at [My Oracle Support](#). For accessible support, visit [Oracle Accessibility Learning and Support](#).

Get Training

Increase your knowledge of Oracle Cloud by taking courses at [Oracle University](#).

Join Our Community

Use [Cloud Customer Connect](#) to get information from industry experts at Oracle and in the partner community. You can join forums to connect with other customers, post questions, suggest *ideas* for product enhancements, and watch events.

Learn About Accessibility

For information about Oracle's commitment to accessibility, visit the [Oracle Accessibility Program](#). Videos included in this guide are provided as a media alternative for text-based topics also available in this guide.

Share Your Feedback

We welcome your feedback about Oracle Applications user assistance. If you need clarification, find an error, or just want to tell us what you found helpful, we'd like to hear from you.

You can email your feedback to oracle_fusion_applications_help_ww_grp@oracle.com.

Thanks for helping us improve our user assistance!

1 About This Guide

Audience and Scope

This guide provides information on how administrators and implementors can make application changes using Application Composer. Application Composer is a browser-based tool that business analysts and administrators, not just programmers, can use to make application changes.

Using this tool, you can make the types of data model changes which previously could only be made by application developers. For example, you can create a new object and related fields and then create new user interface pages to expose that object to users.

Application Composer is a design time at runtime tool, which means that you can navigate to Application Composer directly from an application, make your changes, and see most changes take immediate effect, without having to sign back into the application. (For end users to see your changes, however, you must first publish your sandbox and then they must sign out and sign back in.)

Note: Application Composer isn't supported for use with iPad devices.

You can use Application Composer to configure supported product areas only, such as Oracle's Sales and Fusion Service, Supply Chain Management, and Project Management applications. This guide covers common Application Composer task flows. For product-specific Application Composer examples and use cases, refer to the implementation documentation for your specific product area.

Additional Configuration Tools

Application Composer is just one tool that administrators can use to make common application changes. Other tools in the configuration toolset includes Page Composer, BI Composer, and Business Process Composer.

- **Page Composer** is a page editor that you can use to edit individual UI pages. You can do things like:
 - Create work area lists (saved searches) for use by the whole organization or by individual roles.
 - Reorder columns in search result tables.
 - Configure infolet pages.

Note: Aside from the tasks listed above, don't use Page Composer in Oracle CX Sales. Use of Page Composer in CX Sales is limited to only these few tasks.

To learn more about Page Composer, refer to the Oracle Applications Cloud Configuring and Extending Applications guide.

- **Oracle Business Intelligence (BI) Composer** is a tool you can use to build reports. When building reports, you select a report subject area from within BI Composer. A report subject area is a set of entities, attributes, and measures that represent information about the areas of an organization's business. To build reports, use either the predefined report subject areas that are delivered for you, or create your own subject area using a wizard available in Application Composer.

To learn more about BI Composer, refer to the Custom Subject Areas chapter in this guide.

- When working with object workflows in Application Composer, you can define an object workflow that will trigger an approval flow if certain conditions are met. Approval flows are defined as business processes using **Oracle Business Process Composer**. Business Process Composer lets you orchestrate predefined components such as human-workflow tasks, services, and BPEL flows.

To learn more about Business Process Composer, refer to the Object Workflows chapter in this guide.

Related Guides

You can refer to the following guides to learn more about making changes to your applications.

Guide	Description
Configuring and Extending Applications	Describes the concepts of configuring Oracle Applications Cloud, and the tools that you can use.
Groovy Scripting Reference	Explains the basics of how to use the Groovy scripting language to enhance your application functionality.
Implementing Sales	Describes tasks to configure and set up Sales, including product-specific procedures for configuring Sales pages using Application Composer.
Configuring and Extending Product Lifecycle Management	Describes how to use Application Composer to configure and extend objects and application interfaces in PLM solutions such as Oracle Innovation Management Cloud, Oracle Product Development Cloud, and Oracle Quality Management Cloud.

Related Topics

- [Oracle Help Center](#)
- [Groovy Scripting Reference](#)

Additional Common and Advanced Setup Tasks

Besides the tasks explained in this guide, Application Composer includes several other tasks that are available as nodes under Common Setup and Advanced Setup.

You can refer to the following sections for a list of these tasks and where you can find more information about them.

Common Setup

The following table lists the additional common setup nodes that are explained in detail in other guides:

Node	Reference
Mobile Application Setup	See the "Get Started" and "Configure the App" sections of the Mobile Applications chapter in the Implementing Sales guide.
Productivity Applications Setup	See the "Configure Oracle CX Sales for Microsoft 365" topic of the Extend Microsoft 365 chapter in the Implementing Sales guide.
Oracle Sales Assistant Setup	See the "Sales Assistant" chapter of the Implementing Sales and Using Sales guides.
Notification Preferences	See the "Set Notification Triggers and Preferences" section of the Configure Notifications chapter in the Implementing Fusion Service guide.

Advanced Setup

The following table lists the additional advanced setup nodes that are explained in detail in other guides:

Node	Reference
Copy Maps	See the "Use Copy Maps" chapter of the Implementing Sales guide.
Data Quality Rules	See the "Configure Predefined Data Quality Rules to Your Requirements in Application Composer" topic of the Duplicate Resolution Setup chapter in the Implementing Customer Data Management (CDM) guide.

2 Before You Begin

Get Started with Application Composer

Before you begin, read this chapter to learn about Application Composer, how to access the tool, and how you can test the application changes you make once you're done. You should also be familiar with how to work with sandboxes.

Sandboxes set apart untested configuration changes from the mainline environment. So, wherever possible, make changes to the application in a sandbox rather than making direct changes in the mainline environment, thus avoiding any impact on other users in the environment. See the Configuration Life Cycle chapter in the Configuring and Extending Applications guide.

This chapter includes:

- A brief introduction to Application Composer, and how to access the tool
- What you need to test your changes
- How to let administrators assign themselves the job roles they need for testing
- How to assign yourself any additional job roles required for testing

Related Topics

- [Overview of Sandboxes](#)

About Application Composer

Application Composer is a browser-based configuration tool that enables business analysts and administrators, not just application developers, to extend their applications. Make the type of data model changes which, in the past, could only be made by application developers.

For example, easily create a new object and related fields, then create new user interface pages where that object and its fields are exposed to users.

Application Composer is a design time at runtime tool, which means that you can navigate to Application Composer directly from any application, make your changes, and see most changes take immediate effect in real time, without having to sign out and sign back in. (For end users to see your changes, however, you must first publish your sandbox and then they must sign out and sign back in. Read more about sandboxes later in this topic.)

Note: To see your changes in real time, always use the Navigator to navigate to the runtime page that you changed. Then navigate back to Application Composer to continue making changes. In other words, when making application changes (and testing them), restrict your usage to a single tab. Don't work across multiple browser tabs, because Application Composer doesn't support this type of usage.

You can use Application Composer to configure Oracle's Sales and Fusion Service, Supply Chain Management, and Project Management applications.

Application Composer isn't supported for use with iPad devices.

Application Changes for Nondevelopers

Application Composer hides the complexity of modifying applications by leveraging a set of standard design patterns and wizards. You focus on the application changes that your business requires (object model extensions and layout changes, for example), and Application Composer creates the underlying object artifacts for you.

Using Application Composer, you can make application changes such as the following:

- Modify objects by adding new fields, or create entirely new objects.
- Create foreign key-based relationships between two objects.
- Modify user interface pages by exposing your newly created fields for an object, or create an entirely new work area for your custom objects.

Expose object relationships in the form of subtabs on pages.

- Write application logic, such as triggers, validation rules, and workflows, for an object, or for use across multiple objects.
- Implement functional and data-level security for custom objects.
- Enable objects for custom reporting.

Working in a Sandbox

To make most application changes, you should work in a sandbox. In fact, many functions in Application Composer aren't available until you enter into an active sandbox. You use sandboxes to make application changes and test them without impacting other users in the environment. Wherever possible, make changes to the application in a sandbox rather than making direct changes in the mainline environment.

Sandboxes set apart untested configuration changes from the mainline environment. This lets you test your changes in a sandbox first, before publishing it. After publishing, your changes become available in the mainline metadata, or other sandboxes after they're refreshed, so that everyone can see your changes. Mainline metadata is the primary branch of metadata a sandbox is published to. Note that end users must sign out and sign back in to see the changes from a published sandbox.

Tip: As a best practice for using sandboxes in Application Composer, follow the below sequence to make your application changes:

1. Publish a sandbox with model-level changes, like creating an object and creating relationships for that object.
2. Publish a sandbox with the UI-level changes for the object, like adding layouts, subtabs, etc.

This order can minimize conflicts while publishing your sandboxes.

To learn more about sandboxes, see [Overview of Sandboxes](#).

Accessing Application Composer

Access Application Composer at runtime by using the Navigator menu, and selecting **Application Composer** under the Configuration category.

The first view of Application Composer is the main Overview page, which is the entry point into all your task options.

Getting Started

From the main Overview page:

- Use the object tree to select the object you want to modify. Or, click the New icon to create a new object.
- Use the links in main Overview page, also known as the local area, to select a task. Or, use the links in the Common Setup pane.

Related Topics

- [Define Objects](#)
- [Define Fields](#)
- [Overview of Sandboxes](#)

What's Required for Testing Configurations in the Sandbox

If you're creating configurations for a specific job role or creating your own custom objects, then you must be provisioned with additional job roles to view and test those configurations in the sandbox.

Enable the testing of both types of configurations using the steps described in this section.

What's Required for Role-Specific Configurations

If you're creating configurations for a specific job role in either Application Composer or Page Composer, then you must assign yourself that same job role to be able to test the configurations in the sandbox. For example, if you're creating your own page layout for the Sales Manager job role, then you must have the Sales Manager job role to view and test the layout. If you later create a different layout for salespeople, then you must deprovision the Sales Manager job role and provision yourself with the Sales Representative job role instead.

What's Required for the Objects You Create

If you're creating your own objects, then you must assign yourself the Custom Objects Administration (ORA_CRM_EXTN_ROLE) role. The application automatically generates this object role the first time you create an object in the application. Unless users have this role, they can't view or test the objects they create.

Setup Overview

1. While signed in as a setup user or the initial user you received when you signed up with Oracle, edit the role-provisioning rule for sales administrators and add the required job roles. Here is a summary of the steps:
 - a. In the Setup and Maintenance work area, use the following:
 - Offering: Sales
 - Functional Area: Users and Security
 - Task: Manage HCM Role Provisioning Rules
 - b. Search for all role-provisioning rules containing the Sales Administrator job role.

- c. For each rule, you add the job roles required for testing. Selecting the **Self-requestable** option makes it possible for individual users to assign themselves each job role when needed.
 - d. If you're creating custom objects, then you must also add the Custom Objects Administration role. You must select both the **Self-requestable** and the **Autoprovision** option for this role. This object role is required for all objects you create, so you want to provision it automatically for future to sales administrators.
2. Sales administrators, who are resources with the Sales Administrator job role, navigate to the Resource Directory and assign themselves the job roles they need. Setup users, who are not resources, can edit their own user records in the Manage Users work area and assign themselves the roles there.

For details on how resources can assign themselves job roles in the Resource Directory, see the Assign Yourself an Additional Job Role topic.

Related Topics

- [Assign Yourself Additional Job Roles Required for Testing](#)
- [Enable Sales Administrators to Test Configurations in the Sandbox](#)
- [Enter Setup Data Using Assigned Tasks](#)

Enable Sales Administrators to Test Configurations in the Sandbox

Modify the security role-provisioning rules to make it possible for administrators to assign themselves the job roles they need for testing custom configurations in the sandbox.

For viewing and testing the custom objects they create, administrators must have the Custom Objects Administration (ORA_CRM_EXTN_ROLE) role. To test job role-specific configurations, they must have the same job role. In this example, we are looking at sales administrators.

Modify the Provisioning Rule for Sales Administrators

1. Sign in as a setup user or the initial user you received when you signed up with Oracle.
2. In the Setup and Maintenance work area, use the following:
 - o Offering: Sales
 - o Functional Area: Users and Security
 - o Task: Manage HCM Role Provisioning Rules
3. On the Manage Role Mappings page, search for the role mapping for sales administrators:
 - a. In the Search region, click the **Role Name** list and select the **Search** link.
 - b. In the Search and Select window, enter `Sales Administrator` in the **Role Name** field and click **Search**.
 - c. Select the role name and click **OK**.
 - d. Click **Search**.
4. On the Manage Role Mapping page, click **Search**.

The Search Results display the mappings with the Sales Administrator job role.
5. Click the mapping name of each mapping and make the following edits:

- a. In the Associated Roles region, click **Add Row** (the plus sign icon) and add the job roles required for testing.
 - b. For each job role, select the **Requestable** and the **Self-requestable** options and deselect **Autoprovision**. You don't want the job roles assigned to the sales administrators automatically.
 - c. If you're creating your own objects, then you must also add the Custom Objects Administration role. The application automatically generates this object role the first time you create an object. For this job role select all of the options: **Requestable**, **Self-requestable**, and **Autoprovision**. All users creating their own objects must have this role.
 - d. Click **Save and Close**.
6. When you have added the job roles to all the provisioning rules, click **Done**.

Related Topics

- [Assign Yourself Additional Job Roles Required for Testing](#)
- [Enter Setup Data Using Assigned Tasks](#)

Assign Yourself Additional Job Roles Required for Testing

Administrators can use the procedure in this topic to assign themselves the roles they need to test role-specific modifications in the sandbox.

For example, if you're a sales administrator testing UI modifications for sales managers, you must assign yourself the Sales Manager job role. If you're creating your own custom objects, you must assign yourself the Custom Objects Administration role, if this role isn't already assigned to you. The Custom Objects Administration role is required for testing your objects in the sandbox.

Note: You can only assign yourself job roles that are made self-requestable in the role-provisioning rules created by a setup user. A setup user has the privileges to create other users and manage application security.

1. Navigate to the **Resource Directory**.
2. Select **View Resource Details** from the **Actions** menu in your record.
3. On the Resource page, click the Roles tab.
4. Click **Add Role**.
5. In the Add Role window, search for the role you want to use for testing by name or partial name, select it, and click **OK**.

For testing objects you created, you must add the Custom Objects Administration role.

Note: Available roles include only those that were set up as self-requestable during provisioning rule setup.

The application returns you to the Resource page and displays the requested role in the Roles Requests region.

6. You can remove a role you no longer need for testing by selecting it and clicking **Remove**.
7. Click **Save and Close**.

The new role becomes available for your use in a few minutes, pending the completion of a background process. The role displays in the Current Roles region the next time you navigate to this page.

3 Add Objects and Fields

Overview of Using Application Composer

Use Application Composer to create fields, objects, and relationships. Then, modify user interface pages for all users, or only some.

This chapter explains:

- How to define custom objects
- How to define custom fields for either a custom object, or a standard object
- How to secure custom object data

Other chapters in this guide describe additional tasks flows that are available in Application Composer. For example, learn how to modify user interface pages, create object workflows and custom subject areas, and how to write Groovy scripts. Refer to the table of contents for these other chapters.

Also refer to the Groovy Scripting Reference guide, available in the documentation library, for specific examples and use cases about changing your implementation.

You can access Application Composer by selecting Application Composer from the Navigator menu, under the Configuration category. To test your changes, use the Navigator to switch to the desired application.

Tip:

Navigate quickly and easily between your application's runtime pages and Application Composer design time pages using the Favorites and Recent Items menu.

Note: Application Composer doesn't support recursive navigation and may cause errors. For example, if you try to access the Account object from a custom object record, which has the same custom object subtab, and then attempt to access the Account record again from the custom object subtab, you'll receive an error.

Related Topics

- [About Application Composer](#)
- [Define Objects](#)
- [Define Fields](#)

Objects

Define Objects

Using Application Composer, modify an application's object model so that you can track and store any additional data you might need. For example, add new fields to an existing object (standard objects), or create entirely new objects (custom objects).

Standard objects are objects that are delivered with an application, and made available to Application Composer for application changes. Custom objects are objects that you create using Application Composer. You can create either top-level objects (objects without a parent) or child objects (objects created in the context of a parent).

Read this topic to learn about these tasks:

- Browsing the object tree
- Creating a custom object
- Deleting a custom object
- Using the Object Overview page
- Editing an object's attributes
- Selecting the display icon for the object's set of UI pages
- Viewing child and related objects

Use Application Composer's Object Tree

Access Application Composer from the Navigator menu. The first view of Application Composer is the main Overview page, which is the entry point into all your application change options.

On the main Overview page, the regional pane at left displays the object tree, which lets you browse an application's existing object configuration in a tree format. The object tree reflects the latest configuration of the application: both standard objects as well as custom objects.

To use the object tree:

1. Select **Application Composer** from the Navigator menu, under the Configuration category.
2. For each object node, whether standard or custom, expand it further to view and edit object details.

For example, look at object details such as fields and UI pages where the object is exposed.

Note: At the top of the object tree, you can also click the New icon to create a new custom object.

For both standard and custom objects, you can view and edit the following details:

- Fields
Add new fields to an object.
- Pages
Modify the pages on which an object appears.

- o Actions and links

 - Add actions or links to pages.

- o Server scripts

 - Write application logic that controls the behavior of an object's records.

For custom objects, you can also view and edit details to security. For example, you can implement functional and data-level security for an object and its records.

Create a Custom Object

Create a custom object if you want to track data about an object that's not already delivered with your cloud service. After you create the object, you then add custom fields and design user interface pages where your users can enter object records. There's no fixed limit on the number of custom objects that you can create.

To create a custom object:

1. On the main Overview page of Application Composer, select the **Custom Objects** node in the object tree, or click the icon in the local area of the main Overview page.
2. On the resulting summary table, click the New icon, or at the top of the object tree, click the New icon.
3. Complete the primary identifying attributes for a custom object:

- a. **Display Label**

An object's display label is the user-friendly label for an object, and also becomes the default page title for the object's work area.

- b. **Plural Label**

The plural label is used as the title of the object's work area. The label is also used as the search string in the regional search, as well as in the saved search on the object's runtime overview page.

- c. **Record Name Label**

Use the **Record Name Label** field to specify the display label for the object's RecordName attribute. The RecordName attribute stores the user-entered "name" of the record. For example, if you're creating a custom object, Book. In the **Record Name Label** field for this object, you would enter something like "ISBN Number." At runtime, for each new record, your users would use the **ISBN Number** field to enter

the book's International Standard Book Number (ISBN), which uniquely identifies books published internationally.

Typically, this field is the object's primary user-recognizable identifier that runtime users drill down on, from the landing page to the detail page. For example, at runtime, your users would click any ISBN to drill down to review details about the book, such as book title and author.

d. Record Name Data Type

Select either Text or Automatically Generated Sequence.

For record names of Text data type, the maximum length that users can enter is 32 characters. For record names of Automatically Generated Sequence data type, the sequence number is based on a display format which is up to 28 characters and has at least one number token: {0}.

- e.** Select the **Prevent duplicate values?** check box to prevent users from entering records with duplicate names.
- i.** If the **Prevent duplicate values?** check box is selected, then the **Treat "ABC" and "Abc" as distinct values** check box is enabled.

Select this check box if you want the assessment for duplicate records to be case sensitive.

f. Object Name

The object name is the internal name for the object.

Note: You can use a custom object's internal name only once across the mainline code and all existing sandboxes. If you previously used an object's internal name in a sandbox, you can reuse that same internal name, but you must first delete all other sandboxes where the internal name was previously used. You can use a custom object's display name as many times as you want across sandboxes. The restriction applies only to the internal name.

g. Description

4. Click OK.

Once your custom object is created, you must add fields and then create the UI pages where your users can create actual records. See Define Fields and Create a Set of Application Pages for Custom Objects.

Tip: To create a custom child object, click the Create Child Object button from the parent object's Object Overview page. See the next section in this topic. Once created, a child object can't be changed to a parent object. Similarly, a parent object can't be changed to a child object. Child objects are discussed below.

Delete a Custom Object

Application Composer doesn't support the deletion of either standard or custom objects. If you no longer need an object that was already published to the mainline metadata, optionally enter a note in the description that the object is no longer used.

Use the Object Overview Page

The Object Overview page provides a high-level overview of a standard or custom object. The Object Overview page displays the primary attributes for an object, plus a list of child objects and related objects, if any.

To access the Object Overview page:

1. On the main Overview page, select the object in the object tree or select the **Standard Objects** or **Custom Objects** node in the object tree, or the icon in the local area of the main Overview page.
2. Select the object from the resulting summary table, and click the **Edit** icon.

From the Object Overview page, you can:

- Edit the object's primary attributes, described in the previous section. For example, change the Display Label or Record Name Label.
- Change the display icon for the object.

This process is described as follows:

- View the parent child relationships that were created for this object.

You can also create new child objects from this page, which implicitly creates a new parent child relationship.

- View the non-parent child relationships that were created for this object.

Edit an Object's Attributes

After an object has been created, you can edit its attributes from its Object Overview page.

To edit an object's attributes:

1. Access the Object Overview page for the object, as described earlier.
2. On the Object Overview page, click **Edit**:
 - Change the object's primary attributes, such as display label, description, and record name, at any time.
 - You can't change the Object Name and API Name after the object has been created.

A custom object's API name is automatically derived using the logical name followed by `_c`. You use the API name in Groovy expressions that you build with the expression builder, when writing business logic for the object.

Select the Display Icon for Objects

From the Object Overview page, you can select the display icon to use for the object's UI pages. You can select the display icon for custom objects (although a default icon is provided), and you can change the icon for standard objects. The icon you select determines which icon and theme display to your end users in a variety of locations, such as on the Navigator, subtabs, mobile pages, and the springboard strip on application pages.

Tip: The icon selected for standard objects is inherited across your applications. For example, if you change the display icon for the Opportunity object, then all UI pages are automatically updated to the new icon. This includes even custom subtabs that you added using Application Composer.

To select the icon:

1. Click the object's node in the object tree to view the object's Overview page.
2. On the Overview page, set the icon for the object in the Display Icon region.

View Child and Related Objects

The Object Overview page displays a list of child objects and related objects, if any. You can also create new child objects from this page.

- A child object is an object with a cascade delete relationship to a parent object. This means that if you delete the parent object record, then all its child records are automatically deleted. A child object doesn't exist outside the context of the parent object, and doesn't have its own work area. You can't change a child object to a parent object after the child object has been created.
- Related objects can exist independently of each other, even if one object is deleted. Related objects are connected in a foreign key-based relationship between two top-level objects, not as parent and child. These types of relationships include reference relationships and dynamic choice list relationships.

Related objects can have either a one-to-many or a many-to-one relationship with the current object. Note that an object can be related to itself to model a hierarchy of the object. In this case, the object itself is displayed on its Object Overview page as a related object. For example, the Department and Sub-department objects would be displayed in this way.

Note: You don't create these types of relationships from this page. Instead, manage relationships from the Relationships page, which you can access from Application Composer's main Overview page. Or, create a dynamic choice list relationship by creating a dynamic choice list field for an object, which derives its choice list values from another object.

To create a child object for a standard or custom object:

1. Navigate to an object's Object Overview page.
2. Click the **Create Child Object** button. Creating a child object is the same as creating a custom object, except:
 - The current object is automatically displayed as the parent object.
 - Specify the **Child Collection Name** field to specify the internal name for this set of child object records, which can be used later when writing Groovy scripts.

Related Topics

- [Application Pages for Custom Objects](#)

Object Relationships

Your end users will often need to associate one object's records with the records of another object. To enable this type of association between records, you must first create a relationship between those two objects.

For example, maybe your users want to track the opportunities that get created for an account.

In this example, you will create a one-to-many relationship between the account and opportunity objects, and then expose a list of opportunities as a subtab on the account's details page. This lets users search for and add one or more opportunities to a single account record. When creating relationships between objects, there are four types of relationships that you can pick from in Application Composer. Each type of relationship has its characteristics and advantages. In general, they all let you use a subtab to create or assign one or more records from one object to a record from another object.

Review these aspects of using and managing relationships in Application Composer before you begin to create relationships between objects:

- Relationship types
- Adding subtabs

Relationship Types

Application Composer lets you create either a one-to-many relationship, or a many-to-many relationship. Across these two categories, there are four types of relationships that you can pick from when creating a relationship.

- **Parent child relationship.** A parent child relationship is a one-to-many relationship: one parent record can have many children records. When you create a child object, it's created specifically in the context of its parents. A child object doesn't have its own work area, and the child object's records are deleted if the parent objects is deleted.
- **Dynamic choice list relationship.** A dynamic choice list field provides a list of values from a source object, which your users can select and associate with a target object. When you define the dynamic choice list field, Application Composer automatically creates a one-to-many relationship between the source object and target object.
- **Reference relationship.** You can also manually create a one-to-many relationship, where you can specify a source object and target object. Thus, this type of relationship is similar to a dynamic choice list relationship. The only drawback is that you don't get a dynamic choice list field to add on the target object's work area. A reference relationship only gives you the ability to add a subtab to the source object's details page, showing a list of all the target object records that are associated with a single source object record.

Once you create a one-to-many relationship, you can't delete it.

- **Many-to-many relationship.** Create a many-to-many relationship where, similar to a one-to-many relationship, you can specify a source object and target object. However, with one-to-many relationships, you can add a subtab only to the source object's details page. Many-to-many relationships let your users add one or more records from one object to one or more records from another object.

Once you create a many-to-many relationship, you can't delete it.

Adding Subtabs

After you create relationships between objects, you can then expose one object's records on a subtab that's displayed on the other object's details page.

When adding a subtab to an object's details page, you select to add a Child or Related Objects subtab from the object's Pages Overview page. Application Composer lets you add a subtab based on any target object that has a relationship with the current object as the source object. Subtabs are discussed in related topics.

Related Topics

- [One-to-Many Relationships](#)
- [Many-to-Many Relationships](#)
- [Overview of Dynamic Choice Lists](#)

One-to-Many Relationships

Using Application Composer, you can create one-to-many relationships between two objects within the same application, where one object's primary identifier is stored in another object's table.

A relationship must exist before you can expose the "many" objects on a subtab that's displayed on the "one" object's details page or tree. For example, an account can have multiple service requests associated to it. To expose a list of service requests associated with a specific account as a subtab on the account's details page, you must first create a one-to-many relationship between the account and service request objects. Create these relationships implicitly by creating a child object or by creating a dynamic choice list. Or, create relationships explicitly on the Create Relationship page.

Once you create a one-to-many relationship, you can't delete it.

Parent Child Relationships

Parent child relationships are implicitly created when a custom object is created as a child of a top-level object.

When a child object is created, it's created specifically in the context of its parent. A child object doesn't have its own work area, and the child object is deleted if the parent object is deleted.

View parent child relationships in the object tree, where child objects appear as sub-nodes beneath their parent objects. If a parent child relationship exists, then the child object is listed on the parent's Object Overview page in the Child Objects region. A top-level object can have many child objects. A child object can have only one parent object.

Relationships that are implicitly created from parent child relationships are also displayed on the Relationships page. The relationship name is automatically generated for you.

Dynamic Choice List Relationships

Choice list relationships are implicitly created between two objects when you create a dynamic choice list field.

A dynamic choice list is a field that contains a list of values which are populated from the actual data of another object. For example, you might want to expose on a page a dynamic choice list which lets users specify the HR representative of a given department. The HR Representative choice list is a field that you're adding to the department object, but the list of values is populated by actual employees from the employee object.

When you select an object and create a dynamic choice list field based on a related object, you're implicitly creating a one-to-many foreign key relationship where the current object is the "many" object and the related object is the "one" object. This implicit creation of a relationship lets you later add a related object subtab for the "many" object on the "one" object's details page. You can view these implicitly created choice list relationships on the Relationships page.

View dynamic choice list relationships on an object's Object Overview page. If such a relationship exists, then the related object is listed on the object's Object Overview page in the Related Objects region.

These objects are related objects, not parent child objects. Related objects aren't deleted if the current object is deleted.

Relationships that are implicitly created from dynamic choice list relationships are also displayed on the Relationships page. The relationship name is automatically generated for you.

Note: Generally, the dynamic choice list that you create results in the implicit creation of a choice list relationship. The exception is if you're in a global single instance environment and you create a dynamic choice list between an Oracle CX Sales and Fusion Service object and a common object: resource, customer contact profile, account, address. In such cases, relationships aren't implicitly created.

Creating Reference Relationships

Create a foreign key-based, one-to-many relationship between two top-level objects explicitly using the Create Relationship page. This type of relationship is called a reference relationship.

To explicitly create a relationship between two top-level objects within the same application:

1. Select **Relationships** in the Common Setup pane.
2. On the Relationships page, click the **New** icon.
3. Select the source object and target object.
 A child object can't be the source object or target object.
 The Note common component isn't available for selection as either a source object or target object.
 Once you create a relationship, you can no longer edit the source and target objects.
 This relationship adds a field to the target object to store the foreign key details. If the source object is ever deleted, the target object records remain.
4. Enter the relationship name and description.
 Once you create a relationship, you can no longer edit the relationship name.
5. Optionally add the target object in a subtab to the source object's detail page.
6. Optionally specify data filter criteria for both the source and target objects.
 The filter criteria that you specify here controls which records are available for association at runtime with a record from the other object in this relationship.
 Read: "Configuring a Search and Select Dialog for Custom Objects".

Note: You can create multiple relationships between the same source and target objects. For example, create both a Primary Contact and Secondary Contact relationship between the contact and opportunity objects.

Groovy Script Syntax

Once you have created a one-to-many relationship between objects, a foreign key field is created on the child object or on the "many" object. Use the following API names to access those foreign keys in your scripts.

Relationship Type	Foreign Key API Name	Pattern Used
Parent/child relationship	If the parent object name is ParentObj_c, then the foreign key API name (added to the child object) is ParentObj_Id_c.	<Name of the parent object>_Id_c
Dynamic choice list relationship	If the dynamic choice list field name is DynChoice1_c, then the foreign key API name is DynChoice1_Id_c.	<Name of the dynamic choice list field>_Id_c

Relationship Type	Foreign Key API Name	Pattern Used
Reference relationship (one-to-many)	If the source object name is SourceObj_c, the target object name is TargetObj_c, and the relationship name is relation_Mto1, then the foreign key API name (added to the target object) is SourceObj_Id_relation_Mto1.	<Name of the source object>_Id_<Name of the relationship>

Related Topics

- [Object Relationships](#)

Many-to-Many Relationships

In addition to one-to-many relationships between objects, objects can also have a many-to-many relationship between each other. For example, a service request can have multiple employees working on it. At the same time, a single employee can work on multiple service requests.

In this scenario, you would create a many-to-many relationship between the Service Request and Resource objects, where the related records from both objects store their primary identifiers in an intersection object. Many-to-many relationships aren't supported in desktop work areas.

Once you create a many-to-many relationship, you can't delete it.

Creating Many-to-Many Relationships: Example

To create a many-to-many relationship using Application Composer:

1. Select **Relationships** in the Common Setup pane.
2. On the Relationships page, click the **New** icon.
3. Select the source object and target object.

A child object can't be the source object or target object.

The Note common component isn't available for selection as either a source object or target object.

Once you create a relationship, you can no longer edit the source and target objects.

Note: You can create only one many-to-many relationship for a particular set of objects.

4. Enter the relationship name and description.

Once you create a relationship, you can no longer edit the relationship name.

5. Indicate the cardinality of the relationship:
 - o M:M
 - Many-to-many

6. Enter the name of the intersection object.

The intersection object's table records two foreign keys: one for the Service Request object and the other for the Resource object. This enables the many-to-many relationship.

The intersection object is available as an extensible, top-level object in Application Composer. Optionally extend the intersection object. Custom fields that you add to the intersection object are available for display on the subtabs you create, which is discussed in the section below.

Intersection objects also have a corresponding web service automatically published.

You can optionally enable auditing for this intersection object so that changes to this object's attributes display in the Audit History work area. To set up audit for intersection objects, use the **Manage Audit Policies** task in the Setup and Maintenance work area and select **CRM Application Composer** as the product.

7. Optionally specify data filter criteria for both the source and target objects.

The filter criteria that you specify here controls which records are available for association at runtime with a record from the other object in this relationship.

Read: "Configuring a Search and Select Dialog for Custom Objects".

Adding Subtabs

After you create the many-to-many relationship, you can now add related object subtabs on each object's application details page:

Note: You can add subtabs for a many-to-many relationship to application details pages only. Many-to-many relationships aren't supported in desktop work areas.

- Create an Employee subtab on the service request's details page.
The subtab displays all employees that are working on a specific service request. At runtime, your end users can add or remove employees who are working on a specific service request.
When creating the subtab, you can select which Resource fields to display, such as Employee Name and Title. You can also select which intersection object fields to display, such as Primary Service Request Owner.
- Create a Service Requests subtab on an employee's details page.
The subtab displays all service requests that an employee is working on, since each employee can work on multiple service requests. At runtime, your end users can add or remove service requests that an employee is working on.
When creating the subtab, you can select which Service Request fields to display, such as Service Request Abstract and Date Logged. You can also select which intersection object fields to display, such as Primary Service Request Owner.

When selecting the fields for display on a related object subtab, join fields aren't available for selection if the relationship is a many-to-many relationship.

Configuring a Search and Select Dialog for Custom Objects

A Search and Select dialog lets your end users search for and select object records when assigning one record to another, such as an employee to a service request. These dialog boxes are launched from the related object subtabs that you create, after creating relationships.

Search and Select dialog boxes are automatically provided for standard objects, and aren't extensible. However, if you're creating a many-to-many relationship that involves a custom object, then you must configure the Search and Select dialog boxes for those custom objects.

The filter criteria that you specify in the relationship definition applies to the Search and Select dialog, and controls which records are available for association at runtime with a record from the other object in this relationship.

For example, you can define filter criteria that lets your end users select only "unassigned" service requests for association with an employee.

Groovy Script Syntax

Once you have created a many-to-many relationship, two foreign key fields (one for each object) are created on the intersection object. Use the following API names to access those foreign keys in your scripts.

Relationship Type	Foreign Key API Name	Pattern Used
Reference relationship (many-to-many)	If the source object name is SourceObj_c, the target object name is TargetObj_c, and the intersection object is IntersectionObject_c, then the two foreign key API names (added to the intersection object) are TargetObj_Id_Tgt_TargetObj_cToIntersectionObject_c and SourceObj_Id_Src_SourceObj_cToIntersectionObject_c.	<Name of the object>_Id_Tgt_<Name of the object>_cTo<Intersection object name>, <Name of the object>_Id_Src_<Name of the object>_cTo<Intersection object name>

Related Topics

- [Object Relationships](#)
- [About Search and Select Dialog Boxes](#)
- [Audit Configuration for Business Object Attributes](#)

Activate Global Search on Objects You Created or Deactivated

Oracle activates the global search on all application objects where search is available. Use this procedure to activate search on any objects you deactivated in the past or for objects you created.

You can activate search only on the objects you created, not on their child objects.

To make an application object available for global search, you must do the following:

1. Activate the object.
2. Specify the frequency with which the object will be indexed.
3. Optionally, you can modify the list and order of fields indexed in the search and displayed in the search results.

Activating an Object for Search

Let's look at a Sales object, for example. To activate an object for search, do the following:

1. Navigate to the Setup and Maintenance work area, and use the following:
 - o Offering: Sales

- Functional Area: Sales Foundation
 - Task: Manage Search View Objects
2. Select the object you want to enable for search.
 3. Click **Activate**.

The status for the object changes to **Active**.

Tip: Make sure you deactivate any object that's not needed for global search to maximize system resources.

Setting the Indexing Frequency and Schedule

After you have activated the object, you must specify how frequently you want the object records indexed.

Oracle recommends that you index objects daily during off-hours. You should stagger the indexing times for the different objects to minimize performance impacts.

Specifying the fields to be indexed and displayed in the search results is optional because these are already set up for you.

1. Select the **Display Name** link of the object.

The Edit Search View Object page appears.

2. In the Index Schedule region, select the **Frequency Type** and enter the number of days between index runs and the time, if appropriate. Oracle recommends staggering the indexing schedule to maximize available system resources.
3. You can change which fields the application indexes and which fields display in search results as described in the Specifying Which Fields Are Indexed and Displayed in Search Results section.
4. When you're done, click **Save and Close**

The application returns you to the Manage Search View Objects page where you can monitor the status of the index generation for each object.

The first time your scheduled indexing process runs or any time you modify the list of fields in the object, the application generates a complete index of all the existing records. Subsequently, the process indexes only records that have changed.

If you end up with many inactive records in your system over time, you can improve the efficiency of your searches by periodically regenerating the full index. This can be accomplished by selecting the object and clicking **Full Reindex**.

Specifying Which Fields Are Indexed and Displayed in Search Results

In the Edit Search View Object page, you can also change which fields the application indexes and which fields display in search results. You must index the fields you created if you want them to be available for searches.

- The **Title** and **Fixed Content** fields let you specify which fields are displayed in search results and in what order.
 - **Title** is the linked heading of each search result.
 - **Fixed Content** is the text which appears under the heading.

For example, the titles starting with the word **Opportunity** are links which permit users to drill down to the record. The rest of the fields are the fixed content.

- The **Body** field lists the fields that are indexed by the application. The most relevant fields are displayed in the search results, space permitting. While the **Body** field includes all of the standard fields for indexing, you must add the fields you created to the list if you want them available for searches.

To make changes, click **Edit** (the pencil icon) and make your changes in the Edit Search View Object window.

Custom Object Public Unique IDs

When users create records in the database, the applications use document sequencing to generate a unique ID for each business object record. Understandably, these numbers aren't easy to read or use because of their length and complexity.

To overcome the issue of long IDs, you can set up the applications to make these IDs more user-friendly and readable. This user-friendly value is called the public unique ID (PUID). Custom object PUID values are stored in the Record Number (**RECORD_NUMBER**) field.

You can accept the default format of a PUID wherein the application automatically generates a unique 15-digit numeric ID for each record. Or, you can change the PUID format using the basic or advanced setup. For custom objects, Oracle recommends that you follow the advanced setup. For each custom object that you define, set a prefix so that new PUID values are unique and don't conflict with other data that might have been previously imported or part of any planned future imports. This will prevent PUIDs automatically generated during import from overlapping with PUIDs automatically generated for records created in the UI or via REST.

To configure the prefix for each custom object's PUID format:

1. Sign in as a setup user.
2. In Setup and Maintenance, go to the following:
 - Offering: Sales
 - Functional Area: Sales Foundation
 - Task: Manage Public Unique Identifier Sequence Generation
3. On the Manage Public Unique Identifier Sequence Generation page, if you're configuring these values for the first time, add a row to the table.
4. In the **Object** list of values, select the object for which you're configuring the public unique ID.
5. In the **Radix** list of values, select the base numbering to use. This list of possible radix values is the same list of values that are provided in the ZCA_PUID_RADIX profile option.
6. In the **Prefix** box, enter the prefix you want to use for the object and unique ID.
 - Enter a maximum of five characters. For example, if your custom object is **TroubleTicket_c**, then enter **TICK_**.
 - The allowed characters are: 0-9, A-Z, a-z, and these special characters: period, hyphen, comma, and underscore.
7. Repeat for all other custom objects.
8. Click **Save and Close**.

For more information about the setup for PUIDs, see Overview of Public Unique IDs, Implementing Sales guide.

Related Topics

- [Overview of Public Unique IDs](#)

Security for Custom Objects

Make Custom Object Pages Visible to Users

After creating custom objects and their respective user interface (UI) pages, you must indicate which end users can view the pages (functional security) and enter data (data security). You do this in Application Composer using the custom object's Security node, or the Role Security link in the Common Setup pane.

You can provision data security for custom object records and restrict users who have privileges to view, update, or delete records. You can give access to all users or only to members of access groups, to owners of records or to an owner and his management hierarchy, and to user-defined roles.

Review these aspects of the custom object security process in Application Composer before you begin to define your security policies:

- Who can create custom objects?
- Who can see your custom object?
- Understanding security policies
- Custom vs. predefined roles
- Application Composer and the Security Console

Who Can Create Custom Objects?

By default, a custom object and its records are visible and editable only to users who are provisioned with the Custom Objects Administration (ORA_CRM_EXTN_ROLE) role. Users with any one of the three following job roles can also create custom objects and use all other Application Composer functions:

- Customer Relationship Management Application Administrator
- Application Implementation Consultant
- Master Data Management Application Administrator

Oracle recommends provisioning the user with the Customer Relationship Management Application Administrator job role (for performing the application changes) and the Custom Objects Administration job role and an Administrator job role (for testing the application changes in the UI).

Who Can See Your Custom Objects?

When you create custom objects, by default their UIs are visible only if you have the Custom Objects Administration (ORA_CRM_EXTN_ROLE) role. The application creates this custom role automatically. The UI pages you create for custom objects aren't visible to additional users unless you provide access in Application Composer using the object's Security node. Use the Security node to specify not only which job roles can access the UIs, but the levels of access. Provision data security for custom object records and restrict users who have privileges to view, update, or delete records. You can give access to all users or only to members of access groups, to owners of records or to an owner and his management hierarchy, and to user-defined roles. For example, you can make it possible for owners to update object records, while managers can only view records.

To manage who can see your custom objects at runtime, do the following. In this procedure, we will look at sales managers and salespeople.

1. Ensure that you have assigned the Custom Objects Administration role to yourself and to other users who make application changes.

The Custom Objects Administration role is automatically assigned to new custom objects, as well as existing custom objects, if you have upgraded from a previous release.

Note: If you're creating your own objects, then you must also add the Custom Objects Administration role. The application automatically generates this object role the first time you create an object.

See "Enable Sales Administrators to Test Configurations in the Sandbox."

2. For each custom object, use the Security node to specify which roles can view that object's UI pages, and their level of access: view, update, and delete. This is called a security policy. See the following "What's a Security Policy" section.

When granting access to custom object UIs, you can select only custom job roles. For example, if you want to create a custom object for sales managers, then a custom sales manager job role must first exist (instead of the predefined Sales Manager job role provided by Oracle), before you can grant access to sales managers. If you want to create a custom job role, then copy the predefined Sales Manager job role in the Security Console as described in the Securing Sales and Service guide.

Granting access to custom job roles means that your custom object won't be affected by future upgrades. See the "Custom vs. Predefined Roles" section later in this topic.

3. Alternatively, you can enable the custom object to use access group security, which means that only members of a specific access group can see the custom object records, depending on the rules that you define. See "Enable Access Group Security for Custom Objects."
4. If you're creating custom objects for a specific job role, then you must also assign yourself that job role to view and test the application changes in a sandbox. For example, if you're creating a custom object for sales managers, then you must assign yourself the sales manager job role to test how that object works for sales managers. If you later create a different object for salespersons, then you will have to deprovision the sales manager job role and provision yourself with the salesperson job role instead, so that you can accurately test your new object.
 - o Setup users who have the permission to create and update users can grant themselves the appropriate job roles by editing their user record in the Manage Users work area.
 - o Sales administrators who are resources can request the job role they need for testing by following the procedure described in "Assigning Yourself an Additional Job Role."

To make job roles requestable for sales administrators, a setup user must create a special role-provisioning rule, as described in "Creating the Provisioning Rule for the Job Roles Used in Testing."

5. If you're adding a custom object subtab to a standard object, then you must also assign yourself the job role that can view the standard object's UI.

For example, if you add a custom object subtab to the Edit Opportunity details page. In this case, you will need the role required to access the Edit Opportunity page, in addition to the role granted to the custom object.

What's a Security Policy?

For each custom object, you will need to update its security policy. A security policy specifies which users are authorized to access an object's data, and what type of access they have. Access includes both function security as well as data security. For example, does a user have view only access, or can the user create and update an object's record, as well?

As previously mentioned, custom objects are automatically assigned the Custom Objects Administration (ORA_CRM_EXTN_ROLE) role and creating a custom object record automatically grants you the owner role. Next, grant additional access to your custom object so that your end users can enter data.

For each custom object, you can grant access to multiple roles for a single object, or you can grant access to multiple objects for a single role.

- Define security policies **for an object**.

Authorize the various custom roles whose users can access that object's data.

You must define security policies for child objects, as well.

See "Managing Security by Object: Explained."

- Or, define security policies **for a role**.

Specify the role's access levels across multiple custom objects.

See "Managing Security by Role: Explained."

Define the security policy for a custom object using the Security node in Application Composer, on the Define Policies page. On this page, you can manage an object's data security, which applies to a record of an object and an object's functional security, which applies to the object as a whole. The page provides options for the following security types:

- **Create**

Users with the corresponding role can create a record of the object.

- **Read**

Users with the corresponding role can view the object's work area pages.

- **Update**

Users with the corresponding role can update a record of the object.

- **Delete**

Users with the corresponding role can delete a record of the object.

The Read, Update, and Delete security types provide data security options for the following along with the functional security:

- Owner
- Owner and Management Chain
- All

The Owner field is available on all pages for custom objects. When you create a record, by default, you're the owner. With this security provisioned, you can filter records owned by you or your subordinates.

When you select any of the above data security options, the corresponding functional security is automatically selected.

Custom vs. Predefined Roles

The Define Policies page (for both custom objects and roles) displays custom roles. Custom roles are copies of the predefined roles that Oracle provides for all customers. You can't modify predefined roles, so they aren't displayed here.

However, you can modify custom roles. Modifying a custom role means adding access to one or more custom objects so that role can view the custom object at runtime.

If you don't see a list of roles on the Define Policies page, then you must first copy the predefined roles that you need using the Security Console:

1. Use the Security Console to make copies of the predefined roles you need. These copied roles are known as custom roles.

Refer to your product's security documentation for additional information. For example, for Oracle CX Sales and Fusion Service, see these topics in the *Securing Sales and Service* guide:

- [Copying Sales Roles: Points to Consider](#)
- [Copying Job or Abstract Roles: Procedure](#)

2. Navigate back to Application Composer, open the Security node for your custom object, then define the security policy across roles for your custom object.

If you upgraded from a previous release, then you might have made changes to predefined roles in an earlier release. During the upgrade to the current release, Oracle automatically copies those modified predefined roles for you, so they will appear as custom roles on the Define Policies page. See "Custom Roles and the Upgrade Process: Explained" in the *Securing Sales and Service* guide.

Application Composer and the Security Console

The Security Console manages the security policies that control access based on roles. However, you define the security policies for custom objects in Application Composer's object-centric and role-centric Define Policies pages. This is outside the Security Console.

Security policies defined in Application Composer can be modified in Application Composer. Don't use the Security Console to modify these policies.

Related Topics

- [Assign Yourself Additional Job Roles Required for Testing](#)
- [Manage Security by Object](#)
- [Enable Sales Administrators to Test Configurations in the Sandbox](#)
- [Guidelines for Copying Roles](#)
- [Copy Job Role and Abstract Role](#)

Manage Security by Object

When you create custom objects, by default their UIs are visible only if you have the Custom Objects Administration (ORA_CRM_EXTN_ROLE) role. Use each custom object's Security node to specify the additional job roles that can access that custom object's UIs.

Use the Security node to provision data security for custom object records and restrict users who have privileges to view, update, or delete records. You can provision this type of security to all users or only to members of access groups, to owners of records or to an owner and his management hierarchy, and to user-defined roles. For example, you can make it possible for owners to update records, while managers can only view records.

Alternatively, you can update the security policy for a custom role, across multiple custom objects, using the Role Security link in the Common Setup pane. See "Manage Security by Role."

Managing Object Security

The object-centric Define Policies page displays a list of the custom roles available for selection. Use this page to manage access to either a top-level or child custom object by specifying a security policy for one or more custom roles. When you do this, users with the corresponding custom roles can access the custom object and its data, depending on the security policies you define. Alternatively, you can enable the custom object to use access group security, which means that only members of a specific access group can see the custom object records, depending on the rules that you define.

To access the object-centric Define Policies page:

1. Ensure that you're in an active sandbox session.
2. Navigate to Application Composer and on the main Overview page, select a custom object in the object tree.
3. Select the **Security** node. The page that displays is the object-centric Define Policies page.
4. Enable data security across multiple roles. For each role in the table, indicate if that role can create records, and indicate the level of access for viewing, editing, and deleting records.

If data security is selected, the corresponding functional security is automatically selected.

5. If you want to enable the custom object to use access group security instead, then select the **Enable Access Group Security** check box. For more information about enabling access group security, see "Enable Access Group Security for Custom Objects."

Note: Even if you select the **Enable Access Group Security** check box, you must still configure functional security in the Roles section so that the right roles can see the custom object's user interface pages.

6. You can optionally configure owner security instead of access group security. With owner security, for example, you can provide create and read access to all users, update access to the record's owner and owner management chain, and delete access to only the owner.

If you configure both owner and access group security, then your users will see data from both their owner management chain as well as from access groups that they're members of.

See "Manage Security for Owner and Owner Management Chain."

7. Select the Change History check box for each role that can view the custom object's Change History subtab.

For the Change History subtab to be visible to users on a custom object record, you must do two things:

- a. Check the Change History check box.
- b. Add the Change History subtab to the custom object's details page layout.

Refer to your product's implementation documentation for more information. For example, for Oracle CX Sales and Fusion Service, see "Enable the Change History Subtab" in the Implementing Sales guide.

See "Make Custom Object Pages Visible to Users" to learn about function security and data security.

Related Topics

- [Make Custom Object Pages Visible to Users](#)
- [Manage Security by Role](#)
- [Display the Change History Subtab](#)

Manage Security by Role

When you create custom objects, by default their UIs are visible only if you have the Custom Objects Administration (ORA_CRM_EXTN_ROLE) role. For other roles, you can provide access to multiple custom objects using the Role Security link in the Common Setup pane.

Use the Role Security link to specify not only which custom objects and pages that a single role can access, but also the levels of access. Provision data security for custom object records and specify whether or not the role can view, update, or delete records. You can provision this type of security to all the users, owners of records, owner and management hierarchy, and user-defined roles. For example, you can make it possible for owners to update records, while managers can only view records. The Owner field is available on all pages for custom objects. When you create a record, by default, you're the owner. With this security provisioned, you can filter records owned by you or your subordinates.

Alternatively, you can update the security policy for a custom object, across multiple custom roles, using each custom object's Security node. See "Manage Security by Object."

Managing Role Security

The Role Security page displays a list of the custom roles available for selection. Click a custom role name to navigate to the role-centric security policies page, which displays a list of the custom objects for your implementation. Use this page to manage access for users with the corresponding custom role by specifying a security policy for one or more top-level or child custom objects. When you do this, users with the corresponding custom role can access the custom objects and related data, depending on the security policies you define.

To access the role-centric security policies page:

1. Ensure that you're in an active sandbox session.
2. Navigate to Application Composer and in the Common Setup pane, select the **Role Security** node.
Or, select the **Role Security** hyperlink in the local area of the main Overview page.
Or, from the object-centric Define Policies page, select a role.
3. Click a custom role name to navigate to its role-centric security policies page.

Custom Objects Administration Save Save and Close Cancel

Objects

Actions View

Display Name	Create	Read	Update	Delete	Change History
ActualChild_c	<input checked="" type="checkbox"/>	All	All	All	<input type="checkbox"/>
ApplicationAsset_c	<input checked="" type="checkbox"/>	Read All,Functional Read	Update All,Functional Update	Delete All,Functional Delete	<input checked="" type="checkbox"/>
ApplicationExpe...	<input checked="" type="checkbox"/>	Read All,Functional Read	Update All,Functional Update	Delete All,Functional Delete	<input checked="" type="checkbox"/>
ApplicationInco...	<input checked="" type="checkbox"/>	Read All,Functional Read	Update All,Functional Update	Delete All,Functional Delete	<input checked="" type="checkbox"/>
Childobjcu_c	<input checked="" type="checkbox"/>	All	All	All	<input type="checkbox"/>
Consignment_c	<input checked="" type="checkbox"/>	Read All,Functional Read	Update All,Functional Update	Delete All,Functional Delete	<input checked="" type="checkbox"/>
CustParent10Jul_c	<input checked="" type="checkbox"/>	Read All,Functional Read	Update All,Functional Update	Delete All,Functional Delete	<input checked="" type="checkbox"/>
CustomObj_01_c	<input checked="" type="checkbox"/>	Read All,Functional Read	Update All,Functional Update	Delete All,Functional Delete	<input checked="" type="checkbox"/>
Item1_c	<input checked="" type="checkbox"/>	All	All	All	<input type="checkbox"/>

4. Enable data security across multiple objects. For each object in the table, indicate if the role can create records, and indicate the level of access for viewing, editing, and deleting records.

If data security is selected, the corresponding functional security is automatically selected.

5. Select the Change History check box for each object whose Change History subtab should be visible to the current role.

For the Change History subtab to be visible to users on a custom object record, you must do two things:

- a. Check the Change History check box.
- b. Add the Change History subtab to the custom object's details page layout.

Refer to your product's implementation documentation for more information. For example, for Oracle CX Sales and Fusion Service, see "Enable the Change History Subtab" in the Implementing Sales guide.

Related Topics

- [Make Custom Object Pages Visible to Users](#)
- [Manage Security by Object](#)
- [Display the Change History Subtab](#)

Enable Access Group Security for Custom Objects

You can use access groups to provide resources with access to custom object data. To do this, you must first enable access group security for each custom object.

To enable access group security for custom objects, complete these steps:

1. Navigate to Application Composer and confirm that you're in an active sandbox.
2. Navigate to the Security node of the custom object that you want to enable access group security for.
3. On the Define Policies page, select the **Enable Access Group Security** check box.

CAUTION: You can't disable access group security once enabled, but you can disable specific groups or rules on the Access Groups page in the Sales and Service Access Management work area.

4. Next, enable that custom object for access group object sharing rules. To do this, navigate to the Access Groups page in the Sales and Service Access Management work area.
5. Click the Object Rules tab.
6. On the Object Sharing Rules page, select the **Synchronize Custom Objects and Fields** item from the **Actions** menu. The custom object and its attributes are now available when defining object sharing rules for access groups.
7. In Application Composer, set functional security for required roles.

Navigate to the custom object's Security node, and configure functional security in the Roles section of the Define Policies page. This step isn't related to access group security (data security), but it's a required step so that the right roles can see the custom object's user interface pages (functional security).

After you enable access group security for a custom object, you work with it just like a standard object. Create your object sharing rules for access groups, and all group members are given access to that custom object's data according to the rules.

Tip: When configuring data security, you can optionally configure owner security instead of access group security. With owner security, for example, you can provide create and read access to all users, update access to the record's owner and owner management chain, and delete access to only the owner. You configure owner security in the Roles section of the Define Policies page. If you configure both owner and access group security, then your users will see data from both their owner management chain as well as from access groups that they're members of.

Related Topics

- [Create Custom Object Sharing Rules](#)

Enable Team-Based Access to Custom Objects

You can provide resources with access to custom object data, where access is based on the resource's membership in a team, also known as team-based access group security. With this type of security, team members as well as their management hierarchy can access custom object records.

To enable team-based security for custom objects, complete these steps in Application Composer:

1. Create a relationship between your custom object and the Resource object.

In Application Composer, create a many-to-many relationship between your custom object and the Resource object, where your custom object is the source object.

2. Create a subtab so that your users can add resources to custom object records at runtime.

Add a Team subtab to the custom object details page layout, where the Team subtab is based on the intersection object created from your many-to-many relationship.

3. Configure security so that the team member on the custom object record as well as his management hierarchy have access to the record.

To do this, set security for both the intersection object as well as the custom object.

For the intersection object:

- a. Navigate to the Security node for the intersection object.
- b. On the Define Policies page, select each role that needs access and, for each column (Read, Update, Delete), select **All**.

For the custom object:

- a. Navigate to the Security node for the custom object.
- b. On the Define Policies page, select the Enable Access Group Security check box.
- c. Select the Configure Team for Access Group Security check box and select the many-to-many relationship that you just created.

4. Configure functional security for the required roles.

This step isn't related to access group security (data security), but it's a required step so that the right roles can access the custom object's user interface pages at the appropriate level (functional security).

- a. Navigate to the Security node for the custom object.
- b. On the Define Policies page, select each role that needs access and, for each column (Read, Update, Delete), select the access level for reading, updating, and deleting records: **Functional Read, Functional Delete, or Functional Update.**

5. Publish your sandbox.

Finally, enable your custom object for access group object sharing rules. You do the next set of steps in the Sales and Service Access Management work area.

1. Navigate to Access Groups in the Sales and Service Access Management work area.
2. On the Object Sharing Rules page, select the **Synchronize Custom Objects and Fields** item from the **Actions** menu.

After you sync, your custom object displays in the Object list.

3. Select your custom object from the Object list to configure object sharing rules.

In the Rules region, the (Custom Object) Team and (Custom Object) Team Hierarchy predefined rules display, in addition to the rules for (Custom Object) Owner and (Custom Object) Owner Hierarchy.

4. Click each rule to assign a custom access group and access level.

Note that access groups are automatically created based on roles created using the Security Console.

For more information, see the Access Groups chapter in the Oracle Fusion Cloud Customer Experience Securing Sales and Fusion Service guide:

5. On the Access Groups Monitor page, optionally schedule and run the Perform Object Sharing Rule Assignment process to assign access group object sharing rules to your custom object.

By default, the process runs automatically at scheduled intervals to make sure you have the required access to all object data for your selected access groups. But you can submit the process manually if, for example, you want immediate access to new records and objects.

Related Topics

- [Overview of Access Groups](#)
- [Overview of the Access Groups UI](#)
- [Edit Object Sharing Rules](#)
- [How do I run the Perform Object Sharing Rule Assignment Process?](#)

Enable Territory-Based Access to Custom Objects

You can provide resources with access to custom object data, where access is based on the resource's membership in a territory, also known as territory-based access group security. With this type of security, the territory that the user assigns to a record controls who can see, edit, and delete the record.

Overview of Setup Steps

To enable territory-based access group security for a custom object, complete these steps:

1. Set up your territories and territory hierarchy.
See *Steps to Implement Territories*.
2. Configure the custom object in Application Composer.
3. Configure the access group object sharing rules for the custom object in the Sales and Service Access Management work area.

Let's look at the setup steps in more detail.

Application Composer Setup

In Application Composer, complete these steps inside a sandbox.

1. For your custom object, create a dynamic choice list field using the below configuration.

Sales Territory Dynamic Choice List Setup

Dynamic Choice List Attribute	Value
Display Label	Sales Territory
Related Object	Sales Territory
List Selection Display Value	Name
Data Filter > Advanced Filter	Add the following filter condition so that users can associate only active territories with custom object records: <code>(EffectiveEndDate > :todayDate) AND (StatusCode = 'FINALIZED')</code>
Additional List Display Values	<ul style="list-style-type: none"> ○ Owner ○ Type ○ Territory Function ○ Status
Additional List Search Fields	<ul style="list-style-type: none"> ○ Owner ○ Type ○ Territory Function ○ Status

2. Add this Sales Territory field to the details page layout for the custom object, so that at runtime, your users can add a territory to custom object records.

3. Configure security so that the territory team member on the custom object record as well as his management hierarchy have access to the record.

To do this, set security on the custom object:

- a. Navigate to the Security node for the custom object.
 - b. On the Define Policies page, select the Enable Access Group Security check box.
 - c. Select the Configure Territory for Access Group Security check box and then select the dynamic choice list field that you just created, **Sales Territory**.
4. Configure functional security for the required roles.

This step isn't related to access group security (data security), but it's a required step so that the right roles can access the custom object's user interface pages at the appropriate level (functional security).

- a. Navigate to the Security node for the custom object.
 - b. On the Define Policies page, select each role that needs access and, for each column (Read, Update, Delete), select the access level for reading, updating, and deleting records: **Functional Read, Functional Delete, or Functional Update**.
5. Publish your sandbox.

Object Sharing Rules Configuration

In the Sales and Service Access Management work area, enable your custom object for access group object sharing rules.

1. Navigate to Access Groups in the Sales and Service Access Management work area.
2. On the Object Sharing Rules page, select the **Synchronize Custom Objects and Fields** item from the **Actions** menu.

After you sync, your custom object displays in the Object list.

3. Select your custom object from the Object list to configure object sharing rules.

In the Rules region, these predefined rules display:

- (Custom Object) Territory Owner
- (Custom Object) Territory Owner Hierarchy
- (Custom Object) Territory Team
- (Custom Object) Territory Team Hierarchy

In this context, hierarchy refers to territories and not resources.

4. Click each rule to assign a custom access group and access level.

(Access groups are automatically created and populated based on the roles created using the Security Console.)

5. On the Access Groups Monitor page, optionally schedule and run the Perform Object Sharing Rule Assignment process to assign access group object sharing rules to your custom object.

By default, the process runs automatically at scheduled intervals to make sure you have the required access to all object data for your selected access groups. But you can submit the process manually if, for example, you want immediate access to new records and objects.

For more information, see the Access Groups chapter in the Oracle Fusion Cloud Customer Experience Securing Sales and Fusion Service guide.

Object Sharing Rules Example

When a territory is assigned to a record, the object sharing rules dictate what the territory owner and team members (as well as owners and team members in the territory hierarchy) can see and do with the record.

Let's say you have two sales roles: one for sales managers and one for sales representatives. This means that the automatically created access groups are Sales Manager Custom Group and Sales Representative Custom Group. You might configure object sharing rules as described in the below table.

Based on the below rules:

- If the resource owns the assigned territory (Pacific Northwest) and is in the Sales Manager Custom Group, then they can see and edit the record.
- If the resource owns the assigned territory and is in the Sales Representative Custom Group, then they can only see the record.
- If the resource owns the territory (United States) that's part of the territory hierarchy and is in the Sales Manager Custom Group, then they can see, edit, and delete the record.
- If a resource is a sales manager or sales representative but isn't part of either the assigned territory (Pacific Northwest) or territory hierarchy (United States), then the resource can't access the record at all.

Object Sharing Rules Example

Rule Name	Group Name and Access Level
(Custom Object) Territory Owner	<ul style="list-style-type: none"> • Sales Manager Custom Group: Read and Update • Sales Representative Custom Group: Read
(Custom Object) Territory Owner Hierarchy	<ul style="list-style-type: none"> • Sales Manager Custom Group: Read, Update, and Delete
(Custom Object) Territory Team	<ul style="list-style-type: none"> • Sales Representative Custom Group: Read
(Custom Object) Territory Team Hierarchy	<ul style="list-style-type: none"> • Sales Manager Custom Group: Read, Update, and Delete • Sales Representative Custom Group: Read

Related Topics

- [Overview of Access Groups](#)
- [Overview of the Access Groups UI](#)
- [Edit Object Sharing Rules](#)
- [How do I run the Perform Object Sharing Rule Assignment Process?](#)

Manage Security for Owner and Owner Management Chain

When you create a custom object record, by default, you become the owner. As an owner, you can filter records owned by you and your subordinates using the Record Set filter. See FAQ: What record sets is an owner permitted to search?

When you create a custom object, an **Owner** field is automatically created. You can access this field in Application Composer under the Custom tab of the Fields page. To access this page click the **Fields** node of the custom object.

This example illustrates how an owner can define the following security policies for the Trouble Tickets object:

- All users can create and read the record
- Only the owner and owner management chain can update the record
- Only the owner can delete the record

Managing Role Security as an Owner

1. Ensure that you're in an active sandbox session.
2. Navigate to Application Composer and on the main Overview page, select a custom object in the object tree.
3. Select the **Security** node. The object-centric Define Policies page appears.
4. Navigate to the row of the role whose levels of access you want to edit.
5. Select the **Create** check box.
6. Click the **Read** drop-down list, and then select **Read All**. The Functional Read option gets selected automatically.
7. Click the **Update** drop-down list, and then select **Owner and Owner Management Chain**. The Functional Update option gets selected automatically.
8. Click the **Delete** drop-down list, and then select **Owner**. The Functional Delete option gets selected automatically.
9. Click **Save and Close**.

The security policies for the Trouble Tickets object are now updated to provide create and read access to all users, update access to owner and owner management chain, and delete access to only the owner.

Fields

Define Fields

Use Application Composer to extend your applications by adding new fields to both standard or custom objects. The fields that you add to an object are custom fields.

When creating a custom field, Application Composer provides a set of field types that you can choose from. For example, you can create a check box field, or create a long text field.

View an Object's Fields

Note: An object can have a maximum of 625 fields. For more information on the number of field types per object, see the "Field Types and Field Properties" topic in this guide.

1. Navigate to Application Composer from the Configuration category in the Navigator.
2. Expand the object that you want to add custom fields to.
3. Select the Fields node to navigate to the Fields page.
Click the Standard Fields or Custom Fields tabs to view the standard or custom fields available for the object.
 - On the Standard Fields tab:

Review the list of standard fields that are delivered for an object, and optionally modify the display label and help text for a field.

The list of standard fields includes all the fields that are delivered by Oracle for an object, as well as system fields, which could include:

- CreatedBy
- CreationDate
- Id
- LastUpdateDate
- LastUpdatedBy
- RecordName

Note: The custom objects that you create also contain these same system fields, among others.

- o On the Custom Fields tab:

Review the list of custom fields that you created specifically for your implementation, and create new custom fields.

Add Fields to Objects

To create a custom field:

1. Confirm that you're in a sandbox session, before making any application changes.
2. In Application Composer, select the object that you want to make changes to, then select the object's Fields node.
3. On the Custom Fields tab, click **New**.

Application Composer provides a set of field types that you can choose from when creating new fields:

- o Check box
 - o Currency
 - o Date
 - o Datetime
 - o Dynamic choice list
 - o Fixed choice list
 - o Formula
 - o Long text
 - o Number
 - o Percentage
 - o Record Type
 - o Text
4. Select the type of field you want to create, and then specify the required field attributes to create the custom field.
 5. After you create custom fields, you must expose those fields on the right user interface pages, before your end users can see them. See *Modify Application Pages Using Application Composer*.

When you create custom fields for objects and expose the fields on desktop pages, Application Composer automatically creates all the underlying object artifacts for you, and provides full Web service support for those new fields, as well. Application Composer also makes it easy to enable your object model extensions for importing and exporting.

Note: When you create a new field, its value takes effect only for new records. You must update any records that existed prior to the creation of the new field. You can use Import and Export Management to do so. See Import and Export Custom Objects in this chapter.

Delete Fields

You can delete custom fields from both custom and standard objects if you don't need them, as long as your sandbox is still unpublished.

To delete a field:

1. Make sure that you're in an unpublished sandbox.
2. Select the **Fields** node of the custom or standard object whose field you want to delete.
3. On the Fields page, select the field you want to delete and click the delete icon.

Note: Before deleting a field, make sure to first remove the field manually if it was used in any BI reports, Outlook, Copy Maps, Data Quality Rules, or Oracle CX Cloud Mobile.

Related Topics

- [Field Types and Field Properties](#)
- [Field Display Labels](#)

Field Types and Field Properties

Fields can be one of several types, such as a number field or formula field. And, each field type has a set of standard properties. Read this topic to learn about field types and common field properties.

This topic explains:

- The types of fields that you can create
- How fields work with other components
- The common set of field properties that you can specify for a field

Most properties are common across all types of fields, although some are specific to the type of field you're creating. For example, all field types have a display label, but only some field types can be indexed for search.

- How to index fields for faster searches
- Modifying standard fields

Field Types

Application Composer provides a set of standard field types that you can choose from when creating a new field for an object.

The types of fields that you can create are:

- **Check box**
Select to indicate a true or false attribute of a record.
- **Currency**
Enter a currency amount.
- **Date**
Enter a date, or select one from a calendar.
- **Datetime**
Enter a date, or select a date from a calendar, and enter a time of day. During field creation, you choose whether to show the date or time.
- **Dynamic choice list**
Select from a list of values populated from another object's set of records.
- **Fixed choice list**
Select from a list of static values populated from an FND lookup type.
- **Formula**
Calculated in the runtime application using the Groovy-based expression included in the formula field's definition. This is a read-only field that users in the runtime application don't update. However, the application logic that you write can update these fields directly.
- **Long text**
Enter a combination of letters, numbers, or symbols. This field type supports 32,000 characters.
- **Number**
Enter a number in this field.
- **Percentage**
Enter a percentage. The percent sign is automatically added.
- **Record Type**
Select from a list of static values populated from an FND lookup type. You can associate each choice list value with a role or a page layout.
- **Text**
Enter a combination of letters, numbers, or symbols. This field type is limited to 1500 characters.

How Fields Work with Other Components

When you create new fields for objects, Application Composer limits you to a set of standard field types. The field types that you can select from are already integrated with other components of the Extensibility Framework to provide you with the maximum flexibility when extending your implementation:

- All field types correspond to API data types. Each field type has an API name, such as `customfield_c`.

When writing a server script using the expression builder, use this `_c` field name to reference fields.

- All field types correspond to your web service XSD payload.
- All field types correspond to your import ODI mappings when using Application Composer's import and export feature.
- Most field types correspond to available fields that you can use to create a custom subject area for reporting. Exceptions include long text and formula fields.

Common Properties for Custom Fields

When you create a custom field, you first select the field type. You can't change the field type after the field is created. The specified field type controls which field properties you must define when creating the field. Some properties are common across field types, while other properties are unique to a specific field type.

The common field properties that you can define for a custom field are listed in this table, along with the regions on the field configuration pages where they appear and a list of the applicable field types that you must set these properties for. Use this table to understand the common properties that you must define when creating a new field.

Field Property	Description	Field Property Region	Related Field Types
Label	Specify the display label for the field. You should limit the label to a maximum length of 80 characters, although no maximum length is enforced.	Appearance These properties control how the field appears to your users in the runtime application.	Set this property for all field types.
Help Text	Displays when users hover over the field in the runtime application. You should limit the label to a maximum length of 80 characters, although no maximum length is enforced.	Appearance	Set this property for all field types.
Display Width	Do not use. A field's display width is dynamically calculated based on the field type and resolution.	Appearance	Do not use.
Name	A unique field name for internal use only. The field name is automatically populated based on the field label you enter, but without spaces. Field names can contain only underscores and alphanumeric characters. They must:	Name	Set this property for all field types.

Field Property	Description	Field Property Region	Related Field Types
	<ul style="list-style-type: none"> • Begin with a letter • Not contain spaces • Not end with an underscore • Not contain consecutive underscores • Not include special characters. <p>This might cause issues while generating clients using SOAP web services.</p> <ul style="list-style-type: none"> • Be limited to a maximum of 28 characters if the characters are single byte. <p>If the characters are multibyte, such as Japanese or Chinese, then the maximum character limit is 28/number of bytes per multibyte character. For example, if characters are 2 bytes, then the name is limited to a maximum of 14 characters.</p> <p>If a mix of characters is used, then 28 is the maximum sum of character bytes that's supported.</p> <p>You can't change this property after the field is created.</p> <p>Tip: It is possible to create custom fields with different names, but the same display label. Avoid this scenario, however, to avoid seeing two fields with the same display label when configuring a user interface page.</p> <p>The API name, used in your Groovy scripts, is also automatically generated for a field by taking the logical name and appending <code>_c</code>. Don't use special characters in the API name. Also, the API name must be in English. Otherwise you won't be able to add the field to any page.</p>		
Description	A unique field description for internal use only.	Name	Set this property for all field types.

Field Property	Description	Field Property Region	Related Field Types
Required	<p>Indicate if the field is required. You can also optionally use the expression builder to write an expression that specifies the conditions that must apply for this field to be required.</p> <p>If you write an expression to control whether a field is required, then you must also configure the Depends On choice list. This choice list includes fields from the current object, and is used in the evaluation of your expression at runtime.</p> <p>Default values aren't necessary for required fields. However, you must expose all required fields on the object's creation and details (update and edit) pages wherever those pages appear (such as in the application, mobile, or Outlook UI). Required fields without any default values are automatically added to an object's creation pages. However, required fields with default values must be added manually to creation pages. Also, required fields aren't automatically added to details pages; you must add them manually. This lets your users populate the fields at runtime.</p> <p>The object's web services also reflect the required fields when your sandbox is published to the mainline metadata.</p>	<p>Constraints</p> <p>Specify constraints, which let you control the runtime behavior of the field.</p>	<p>Set this property for all field types except for formula fields.</p>
Updatable	<p>Indicate if the field is updatable. You can also optionally use the expression builder to write an expression that specifies the conditions required for this field to be updatable.</p> <p>If you write an expression to control whether a field is updatable, then you must also configure the Depends On choice list. This choice list includes fields from the current object, and is used in the evaluation of your expression at runtime.</p> <p>Here are a few things to note about the updatable property:</p> <ul style="list-style-type: none"> The updatable property is read only in the custom detail page layouts of Account and 	<p>Constraints</p>	<p>Set this property for all field types except for formula fields.</p>

Field Property	Description	Field Property Region	Related Field Types
	<p>Contact objects for security reasons.</p> <ul style="list-style-type: none"> If a field isn't marked as updatable, you can't update it at the user interface level or through web services. You can, however, update it through the import and export functionality, and using server scripts. If a field isn't marked as updatable, then it isn't applicable for mass update. 		
Searchable	<p>Indicate if you want this field to be made available for selection as an additional search criterion from the Add Fields choice list in the Advanced Search mode.</p>	Constraints	<p>Set this property for all field types except for long text and formula fields.</p> <p>See Creating Searchable Fields.</p>
Indexed	<p>Index the field to speed up the performance of saved searches in the different object work areas. The global search isn't affected.</p> <p>Use this option only on the most frequently searched custom fields to ensure optimal search performance.</p> <p>See Indexing Fields.</p>	Constraints	<p>Set this property for the following field types, either during creation or when editing them:</p> <ul style="list-style-type: none"> Currency Number Percentage Text <p>You can't index these field types at all:</p> <ul style="list-style-type: none"> Check Box Date Datetime Dynamic Choice List Fixed Choice List Formula Long Text
Include in Service Payload	<p>Specify whether the field value can be included in a web service request or response.</p>	Constraints	<p>Set this property for all field types.</p>
Fixed Value	<p>Specify a literal default value for the field.</p> <p>Don't assign a literal default value to fields that are both required and intended to be unique, as a runtime error can occur.</p>	Default Value	<p>Set this property for all field types except for formula fields and dynamic choice lists.</p>

Field Property	Description	Field Property Region	Related Field Types
Expression	Use the expression builder to write an expression that dynamically sets the default value for a field at runtime.	Default Value	<p>Set this property for all field types except:</p> <ul style="list-style-type: none"> • Check Box • Formula • Fixed Choice List • Dynamic Choice List <p>For these field types, write server scripts.</p>

Modifying Standard Fields

You can modify the properties of some standard fields, as well. The properties that you can modify are listed in this table.

Field Property	Description	Field Property Region
Required	<p>Indicate if the field is required. You can also optionally use the expression builder to write an expression that specifies the conditions that must apply for this field to be required. If a standard field is already set to required by Oracle, however, then you can't change the field to be not required.</p> <p>If you write an expression to control whether a field is required, then you must also configure the Depends On choice list. This choice list includes fields from the current object, and is used in the evaluation of your expression at runtime.</p> <p>Default values aren't necessary for required fields. However, you must expose all required fields on the object's creation and details (update and edit) pages wherever those pages appear (such as on the desktop, application, mobile, or Outlook UI). Required fields without any default values are automatically added to an object's creation pages. However, required fields with default values must be added manually to creation pages. Also, required fields aren't automatically added to details pages; you must add them manually. This lets your users populate the fields at runtime.</p> <p>The object's web services also reflect the required fields when your sandbox is published to the mainline metadata.</p>	Constraints
Updatable	Indicate if the field is updatable. You can also optionally use the expression builder to write an expression that specifies the conditions required for this field to be updatable.	Constraints

Field Property	Description	Field Property Region
	<p>If you write an expression to control whether a field is updatable, then you must also configure the Depends On choice list. This choice list includes fields from the current object, and is used in the evaluation of your expression at runtime.</p> <p>Here are a few things to note about the updatable property:</p> <ul style="list-style-type: none"> • The updatable property is read only in the custom detail page layouts of Account and Contact objects for security reasons. • If a field isn't marked as updatable, you can't update it at the user interface level or through web services. You can, however, update it through the import and export functionality, and using server scripts. • If a field isn't marked as updatable, then it isn't applicable for mass update. 	

Number of Field Types per Object

An object can have a total of 625 fields. The following table describes how this number is distributed among the various field types:

Number of Fields	Reserved for...
350	<ul style="list-style-type: none"> • Check boxes • Dynamic choice lists • Fixed choice lists • Text fields • Record type fields <p>Note: Of the 350, you can add one record type field per object.</p>
200	<ul style="list-style-type: none"> • Currency fields • Number fields • Percentage fields
50	<ul style="list-style-type: none"> • Date fields • Datetime fields
25	Long text fields
No limit	Formula Fields

Number of Fields	Reserved for...
	<p>Note: A formula field is a computed attribute, and exists only at runtime. This is a transient type of attribute that doesn't persist in the database as a table column. Hence, no maximum number of formula fields exists for an object.</p>

Related Topics

- [Define Fields](#)
- [Groovy Scripting](#)
- [Field Display Labels](#)

Indexing Fields

Index your custom fields to speed up the performance of saved searches in the different object work areas. The global search isn't affected. To ensure optimal search performance, index only the most frequently searched custom fields.

What can you do with indexing?

- You can index only these types of fields:
 - Currency
 - Number
 - Percentage
 - Text
- Once you index a field, you can't un-index it.
- You can index fields either when you first create them, or when you later edit them. But, there are different limits in place when creating fields vs. editing them.

Let's look at the index limits when creating a field:

- For standard objects, you can index two text fields and three number fields (shared among number, percentage, and currency fields) during creation.
- For custom parent objects, you can index 10 text fields and 15 number fields (shared among number, percentage, and currency fields) during creation.

For custom child objects, you can index 10 text fields and 14 number fields during creation.

There is a separate index limit in place when editing a field. You can index fields after creating them, whether for standard or custom objects, but you are limited to indexing a maximum of 5 fields across your entire implementation. This is in addition to the fields that you can index per object, during creation.

Note: You can index a greater number of fields for certain Oracle Sales standard objects. The exceptions are listed in the following table.

Index Limits for Oracle Sales Standard Objects

Oracle Sales Standard Object	Indexing Options
Account	<ul style="list-style-type: none"> • 10 number fields • 10 text fields • 10 case-insensitive text fields (UPPER)
Account custom child	<ul style="list-style-type: none"> • 15 number fields • 10 text fields
Contact	<ul style="list-style-type: none"> • 10 number fields • 10 text fields • 10 case-insensitive text fields (UPPER)
Contact custom child	<ul style="list-style-type: none"> • 15 number fields • 10 text fields
Opportunity	<ul style="list-style-type: none"> • 2 text fields • 8 number fields
Sales Lead	<ul style="list-style-type: none"> • 2 text fields • 16 number fields
Service Request	<ul style="list-style-type: none"> • 2 text fields • 1 number field

Things to consider when indexing fields:

- Dynamic choice list fields and relationships automatically use 1 indexed number field.
- If all indexed number fields are already taken, then Application Composer uses a non-indexed number field.
- If a tab or BI analysis is based off a dynamic choice list field or other relationship, then create that relationship first to ensure you obtain an indexed number field. This ensures optimal performance for your tab or analysis.

Creating Searchable Fields

When creating custom fields in Application Composer, you can indicate if you want them to be searchable using the Searchable checkbox. In this context, searchable means available for selection on a list page from the **Add Fields** choice list in Advanced Search mode. For best performance, consider the following details when using the Searchable checkbox:

You can select the **Searchable** checkbox for all field types except for long text and formula fields.

When you create a custom field, this checkbox is selected by default. However, if you keep the Searchable checkbox selected for all custom fields for an object, then this could potentially degrade list page performance for that object, especially if the object has many custom fields.

Here's how list page performance can be negatively impacted:

- When a user navigates to a list page and expands the Advanced Search, the Add button displays the list of searchable attributes as a drop-down list.

If a very large number of custom fields have the Searchable property set to true, then the performance of rendering the Advanced Search window can be impacted because it needs to populate all searchable attributes in the list.
- If a user keeps the Advanced Search in its expanded mode on a list page, then when the user searches for or drills into a record and then clicks **Save and Close** to return to the list page, the searchable attributes list will be rendered again.

Therefore, you can reduce the performance impact on a list page by reducing the number of searchable fields. To do this effectively, determine which fields your users will most often use to perform searches for the object. Then, when you create a custom field, deselect the Searchable checkbox unless it's one of those necessary search fields. This will reduce the performance impact of needing to populate the searchable attributes list whenever Advanced Search is used.

Field Display Labels

You can change the display labels (strings) that appear in your applications using Application Composer and User Interface Text Tool. These tools enable the administration of strings. Your tool selection depends on the scope of the changes that you want to make.

For large-scale changes or changes to labels in locales other than English, use the User Interface Text Tool. For changes on a single page or to all labels for a custom object in the English locale, use Application Composer. Read this topic to learn more about best practices for changing display labels.

Display Labels

A single field label, such as Products, can occur across your applications on a variety of pages. However, just because the same field label appears multiple times doesn't mean that it's the same field across all pages. In fact, it's probably not. Note that there is a difference between a field label instance and a field label value. The value is what is seen on a page. The instance is the actual field location where the label value is retrieved from. Typically, multiple display label instances may exist that have the same value. This is especially true for common labels such as Account and Customer.

For example, you want to change Products to Our Products, but the field label Products appears throughout your applications 500 times. You don't have to change the Products display label 500 times. However, you will probably need to change it more than once. Again, this is due to the underlying architecture of your applications where multiple display label instances can (and do) exist with the same value.

Display Labels and Multiple Languages

Your applications are fully globalized and support multiple languages. English is installed by default and additional languages can be requested. When additional languages are installed, any user can change the current session locale (language) of their login session through the Preferences menu. However, string administration tools can change display labels to any language, regardless of the session locale. For example, the active session locale can be French, but you can use any string administration tool to enter Korean display label values. As a result, Korean display label values will be seen, even if the session locale is French. Keep this in mind when making changes to field display labels.

To modify display labels in multiple language refer to the topic, Changing a Specific Field Label.

Changing Display Labels Across Multiple Pages

The User Interface Text Tool, sometimes referred to as the String Editor, administers the vast majority of display labels. It is designed for bulk display label search and replace. This tool supports all installed languages and can be used to create and modify locale-specific label values.

Note: You cannot modify list of value fields (also known as choice lists) using the User Interface Text Tool. Instead, use Setup and Maintenance. In some cases, you can also use Application Composer to modify these special types of fields.

For example, you want to change the display label Products to Our Products wherever it appears throughout your applications.

To make a large-scale change of a display label value across multiple pages:

1. First, sign in as an application implementation consultant and confirm that you are working in an active sandbox.
2. Using the Navigator, select **User Interface Text** from the Configuration category.
3. Click **Search and Replace**.
4. In the Search For and Replace With boxes, enter the word or phrase that you want to change and the corresponding replacement.

Tip: For optimal results, enter search criteria that produces the smallest possible number of search results. This makes it simpler to preview all possible matches in the next step. To do this, instead of entering **Products**, enter **^Products\$**. Behind the scenes, the search engine finds label values where Products is the entire display label value.

5. Click **Match Case**.
6. Click **User Interface Text** and **Global Menu Label Text** to search only the display label category that contains the vast majority of UI display labels. All other check boxes can be deselected.
7. Click **Preview Changes** to review, modify, and exclude individual occurrences before you save your changes.

Changing Specific Display Labels

Sometimes you will only want to change a specific instance of a field label, rather than the labels of all the fields with the same name. For instance, the Opportunity page includes multiple instances of the label Account, but you might only want to change one. If you're working in an English locale, make this change using Application Composer. If you're in a non-English locale, use the User Interface Text tool.

To change an individual field name using Application Composer, look at the object's fields, then change each one with the same name to a unique value. For example, if you want to change the Account field and there are four instances of it, then change each one to something like Account 123, Account 456, and so forth. Then you can look at the object's UI to identify the one you want to change. After making the change, update the others to their original names, and make a note in the Description field for each one to help others in the future.

For a non-English-localized environment, you can't just use Application Composer to change the field name, because Application Composer is supported only for an English locale. Instead, access the User Interface Text tool after changing each field label to a unique name. In the User Interface Text tool, specify the language you want to change to in the Language list, then enter the unique name of the field you want to change in the Search box and the name you want to change to in the Replace box.

See Changing a Specific Field Label: Explained.

Changing Custom Object Display Labels

For example, you want to change all the display labels for a custom object, Trouble Ticket. In this case, use Application Composer.

Use Application Composer to change a display label value for custom object and custom field display labels. The change is displayed everywhere the original label value was displayed, across all UI channels.

Note, however, that if you use Application Composer to change display labels for standard object and standard object standard fields, the change is not displayed everywhere. This is because Application Composer changes only one instance of a label, while there are typically multiple instances of a label in use for every standard object and standard object standard field. In this case, Application Composer and Page Composer are similar tools: both tools change one and only one instance of a display label. Again, only the User Interface Text tool has the potential to change every instance of a display label.

Application Composer is an English only tool. It reads and writes only to English label data sources. Application Composer reads and writes in English display label data sources only, even if a non-English locale is the active session locale.

Tip: To add non-English display label values for custom objects, custom fields, standard objects, and standard fields, use the User Interface Text tool.

User Interface Properties of Fields

Define Required, Updatable, and Hidden Properties for Fields

Use Application Composer to define a field as Required, Updatable, or Hidden at the layout level. Setting these user interface (UI) field properties helps reduce the number of layouts.

You can also use expressions (rules) to mark a field conditionally required, conditionally updatable, or conditionally hidden. Fields without conditional hide and show capabilities may result in a large number of layouts. You can change the field properties for individual layouts or perform a mass update of the field properties across multiple layouts.

The advanced UI property changes that you make to the fields are only applicable at the layout level and not at the model level. Also, if there are already field-level security settings in place, then you won't be able to change those settings using the advanced UI properties.

Note: You can't define or edit the advanced UI properties of action links, aggregate fields, and grouping fields. For formula fields and join fields, you can only set the hidden property. For check boxes, you can only set the updatable and hidden properties.

The following scenarios explain the different UI properties you can define for a field:

- **Required:** For example, when a contact is created for a field sales representative, a phone number or an email is required. However, if the contact is created by a web service or imported from an external system, such enforcement may or may not be in place. You can't make the Email field a required field at the model layer, but you can choose to enforce it in the layout by setting the Required property. See "Set the Required Field Property" for an example on how to define the Required property.
- **Updatable:** For example, you can make certain fields read-only for partner users by having separate layouts for partner and internal users, with the Updatable property defined differently for each layout.

Note: If a field already has an EL expression defined for it, then the Updatable property is disabled at the layout level and you can't change it. For example, due to data security constraints, all fields on the Edit Account > Profile subtab and Edit Contact > Summary subtab already have EL expressions defined. This means that you can't set the Updatable property for any standard or custom field on those subtabs. Also, a field can't have an expression at the layout level in the following cases:

- If the updatable property contains a user-defined expression.
- If there's a predefined expression added to the field. For example, Account and Contact can have an expression added for extra security when the fields are moved to page layouts.

See "Set the Updatable Field Property" for an example on how to define the Updatable property.

- **Hidden:** For example, consider a field named Shipping Method with values: Air, Land, and Sea. When the Shipping Method field is set to Land, a Land Transportation field appears with values: Train or Truck. The Land Transportation field doesn't appear if the Shipping Method field isn't set to Land. You can add the fields to all layouts and make the Land Transportation field conditionally hidden or shown using the Hidden property. This way you can avoid creating multiple layouts.

See "Set the Hidden Field Property" for an example on how to define the Hidden property.

Related Topics

- [Edit the User Interface Properties of a Field](#)
- [Edit User Interface Properties of a Field Across Layouts](#)

Edit the User Interface Properties of a Field

To edit the user interface (UI) properties of a field:

1. Navigate to Application Composer.
2. Expand the object whose page layout you want to make changes to.
3. Click the **Pages** node.
4. In the Details Page Layouts section, select the page layout that you want to update.
5. Click the field whose properties you want to set or edit.

Note: Fields that already have property settings configured display an icon next to them.

The Edit UI Properties page appears with the current properties of the field listed in the Field Definition section.

6. In the Layout Properties section, specify or edit the **Required**, **Updatable**, and **Hidden** properties as required for the current layout.

Values available for the properties are listed below:

Property	Values
Required	<ul style="list-style-type: none"> ○ Yes ○ No

Property	Values
	<ul style="list-style-type: none"> ○ Expression
Updatable	<ul style="list-style-type: none"> ○ Yes ○ No ○ Expression
Hidden	<ul style="list-style-type: none"> ○ No ○ Expression

7. To set a a rule for a property using an expression:
 - a. Select **Expression** as the value.
An expression builder icon appears next to the property.
 - b. Click the Expression Builder icon to open the Advanced Expression dialog and enter the expression.

Note: Field properties are evaluated many times while rendering a page. This means that any expression you enter here could impact performance. In general, when entering an expression to make a field conditionally required or updatable, avoid using object functions and don't set any object attribute values.
 - c. Click **OK**.
The expression you enter appears as read-only text next to the property on the Edit UI Properties page. If you select expression and don't enter any expression, an error message appears.
8. Click **Save and Close**.
9. Click **Done** on the Details Layout page.

The edited field is updated for the current layout.

Edit User Interface Properties of a Field Across Layouts

To perform a mass update of the UI properties of a field across multiple layouts:

1. Navigate to Application Composer.
2. Expand the object whose page layouts you want to make changes to.
3. Click the **Pages** node.
4. On the Application Pages tab, click **Actions** from one of the page layout sections, and then select **Edit UI Properties** from the drop-down list.
The Edit UI Properties page appears.
5. Select the field whose properties you want to change.
The Field Definition section displays the current properties of the field at the model level.
6. The Layout Properties section displays all the active layouts for the field, where you can select multiple layouts on which the field properties can be edited. Select the layouts that you want to update.
7. Click **Update**.
The Update Layouts page appears.
8. In the Layout Properties section, specify or edit the **Required**, **Updatable**, and **Hidden** properties as required for the selected layouts.

9. Click **Submit**.

The Edit UI Properties page appears.

10. Click **Done**.

The field is updated across the selected layouts.

For a worked example on how to edit field properties across layouts, see [Editing Field Properties across Multiple Layouts: Worked Example](#).

Set the Required Field Property

You can make a field conditionally required using the Required property. For example, consider a custom object named Deal Registration that contains a Partner Justification field.

Your requirement is that the Partner Justification field always appears as a mandatory field, but only for partner users.

The following example illustrates how to set the Partner Justification custom field as conditionally required for partner users.

Make a Field Conditionally Required

1. Navigate to the Details Page Layouts section of the Deal Registration object.
2. Select **Default Custom Layout**.

The Details Layout page appears.

3. Click the **Partner Justification** field.

The Edit UI Properties page appears.

4. In the Layout Properties section, select **Expression** from the drop-down list of the **Required** property.

The Expression Builder icon appears next to the property.

5. Click the Expression Builder icon.

The Advanced Expression page appears.

6. Enter the following expression in the Edit Script section:

```
def secCtx = adf.context.getSecurityContext()
if (secCtx.isUserInRole('ORA_ZPM_PARTNER_SALES_REPRESENTATIVE_JOB'))
return true
else
return false
```

Note: Field properties are evaluated many times while rendering a page. This means that any expression you enter here could impact performance. In general, when entering an expression to make a field conditionally required or updatable, avoid using object functions and don't set any object attribute values.

7. Click **OK**.

The Edit UI Properties page appears with the expression next to the **Required** property.

8. Click **Save and Close**.
9. On the Details Layout page, click **Done**.

The Partner Justification field now appears as a required field in the edited layout for partner users.

Set the Updatable Field Property

You can set fields of a layout to be read-only or updatable. For example, consider a custom layout created for the Ticket object, named Closed Ticket, which only applies when the status of the ticket is closed. Here, Ticket is a custom object.

The following steps define how to set the **Ticket Name**, **Priority**, and **Status** fields of the custom layout as read-only and to set the **Close Comments** field to updatable.

Note: By default, the fields are updatable

Define Fields as Read-Only

To change the status of the Ticket Name, Priority, and Status fields to Read-Only:

1. Navigate to the Details Page Layouts section of the Ticket object.
2. Select **Closed Ticket Layout**.

The Details Layout page appears.

3. Click the **Ticket Name** field.

The Edit UI Properties page appears.

4. In the Layout Properties section, select **No** from the drop-down list of the **Updatable** property.
5. Click **Save and Close**.
6. Repeat for **Priority** and **Status** fields.
7. Click **Done** on the Details Layout page.

The **Ticket Name**, **Priority**, and **Status** fields are now read-only for the edited layout and the **Close Comments** field is updatable by default.

Set the Hidden Field Property

Consider a new custom layout created for the Opportunity object with an Opportunity Type custom field.

The Opportunity Type field is a fixed choice list with values: Purchase, Lease, and Trade in. When the Opportunity Type is Trade in, a Trade in value is required. So, the Trade in Value field should appear only when the Opportunity Type field is set to Trade in. You can do so by defining the Hidden property.

Make a Field Conditionally Hidden

To set the Hidden property for the Trade in Value field:

1. Navigate to the Details Page Layouts section of the Opportunity object.
2. Select **New Custom Layout**.

The Details Layout page appears.

3. Click the **Trade in Value** field.

4. In the Layout Properties section, select **Expression** from the drop-down list of the **Hidden** property.

An Expression Builder icon appears next to the property.

5. Click the Expression Builder icon.

The Advanced Expression page appears.

6. From the **Depends On** field drop-down list, select **Opportunity Type**.

7. Enter the following expression in the Edit Script section:

```
if (OpportunityType_c == 'TRA') return false;  
else  
return true;
```

8. Click **Ok**.

The Edit UI Properties page appears with the expression next to the **Hidden** property.

9. Click **Save and Close**.

10. Click **Done** on the Details Layout page.

The Trade in Value field now appears only when the Opportunity Type field is set to Trade in for the current layout.

Edit Field Properties across Multiple Layouts

Consider a custom object named Ticket for which you want your users to always enter an account while creating a ticket, but not when they're importing a ticket from an external source.

To do this, you can set the Account field as required across all the creation page layouts. Here, Ticket is a custom object that can be created. To implement this worked example, use a custom object created in your own environment.

This example illustrates how to perform a mass update of the UI properties of the Account field across the creation page layouts.

Perform a Mass Update of the UI Properties of a Field

1. Navigate to the Creation Page Layouts section of the Ticket object.
2. Click **Actions** and select **Edit UI Properties** from the drop-down list.

The Edit UI Properties page appears.

3. Select the **Account** field from the drop-down list.

The Field Definition section displays the current properties of the field at the model level. Note that the **Required** property is set to **No**.

The Layout Properties section displays all the active layouts for the field.

4. In the Layout Properties section, select all the layouts.
5. Click **Update**.

The Update Layouts page appears.

6. In the Layout Properties section, set the **Required** property to **Yes**.
7. Click **Submit**.

The Edit UI Properties page appears.

8. Click **Done**.

The Account field is now updated as a required field across the selected layouts.

View Field Properties

To view all the various settings defined for a field:

1. Navigate to Application Composer.
2. Expand the object whose field settings you want to view.

3. Click the **Pages** node.
4. On the Application Pages tab, click **Actions** from one of the page layout sections, and then select **Edit UI Properties** from the drop-down list.

The Edit UI Properties page appears.

5. Select the field whose settings you want to view.

The Field Definition section displays the current properties of the field at the model level. The Layout Properties section displays all the active layouts for the field.

Here you can edit the field properties across multiple layouts. See [Editing User Interface Properties of a Field across Multiple Layouts: Procedure](#).

Check Box Fields

Using Application Composer, you can extend your application's object model by adding fields to both standard or custom objects. One such field is a check box: users in the runtime application can select it to indicate a record's true or false attribute.

Check Box Field Properties

Create a check box field by specifying values for the common set of field properties, such as display label and field name.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Name	Name
Description	Name
Required	Constraints Note: By default, the check box field is a required field and can't be edited during its creation. You must provide a value for it at runtime to indicate its state as true or false.
Updatable	Constraints
Searchable	Constraints
Fixed Value	Default Value

Field Property	Field Property Region

Additional Check Box Field Specifications

Additional specifications for this field type include the following details:

- Data type is VARCHAR2.
- An object can have a total of 625 fields. Of these, 350 are reserved for text and check box fields, and fixed and dynamic choice lists.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)

Currency Fields

Using Application Composer, you can extend an application's object model by adding new fields to both standard or custom objects. One such field is a currency field, where users in the runtime application can enter a currency amount.

Currency Field Properties

Create a currency field by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the currency field type.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution.
Name	Name
Description	Name
Required	Constraints

Field Property	Field Property Region
Updatable	Constraints
Searchable	Constraints
Indexed	Constraints
Fixed Value	Default Value
Expression	Default Value

The following table shows properties that are unique to certain field types, including currency fields:

Field Property	Field Property Region
<p>Maximum Length</p> <p>Specify how many digits a user can enter in the field.</p> <p>The maximum length is the total number of digits that the currency field can have. Decimal places are validated against what's configured for the currency code in Setup and Maintenance. Refer to the following section for setup instructions.</p>	Constraints
<p>Minimum Value</p> <p>The minimum numeric value that a user can enter into this field.</p>	Constraints
<p>Maximum Value</p> <p>The maximum numeric value that a user can enter into this field.</p>	Constraints

Additional Currency Field Specifications

Additional specifications for this field type include the following details:

- Data type is NUMBER.
- An object can have a total of 625 fields. Of these, 200 are reserved for number, currency, and percentage fields.

Note: Each currency field uses two number type columns: one stores the amount itself, and the other stores the currency conversion rate that's calculated from the entered amount's currency code to the corporate currency code.

- An object includes the following fields to assist with currency conversion. These fields are automatically added to an object if the object's application allows the creation of currency fields. They are derived from the application's corporate currency setup.
 - Currency code
 - The currency code for all of an object's currency fields.
 - Corporate currency code
 - The currency conversion rate type.

Currency conversion for a currency field occurs as follows:

- At runtime, the user enters the currency amount.
 - When the user saves the record:
 - The currency amount is stored using the currency code specified for the object.
 - The application calculates the currency conversion rate using the object's currency code, corporate currency code, currency conversion rate type, and the currency field's specified exchange date, if any.
- In addition to the entered amount, only the conversion rate that's calculated from the entered amount's currency code to the corporate currency code is stored.
- If you later change either the currency code or exchange date, the application recalculates the currency conversion rate for the record. Note, however, that the currency amount displayed on the application page won't change.

Note: When you run a report based on a custom subject area that uses a currency field, the report does display your preferred currency based on the current exchange rate. Again, this is different from how currency amounts are displayed at runtime on application pages, because currency fields only ever display in the entered amount, even if the currency conversion rate for the record changes.

- Precision, or the number of decimal points, for a currency field is derived from the currency code itself. To set the precision for a currency code in Oracle CX Sales, for example:
 - a. In the Setup and Maintenance work area, select the following:
 - Offering: Sales
 - Functional Area: Sales Foundation
 - Task: Manage Currencies (under All Tasks)

The Manage Currencies page appears.
 - b. In the Currency Code field, enter a currency code, such as JPY.
 - c. In the Search Results region, expand the currency code and enter a number into the Precision field.

For example, to display two decimal places for currency fields based on JPY, enter 2 in the Precision field for the JPY currency code.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)

Date Fields

Using Application Composer, you can extend an application's object model by adding fields to both standard or custom objects. One such field type is a date field, where users in the runtime application can enter a date or select one from a calendar.

This type of field has no time component.

Date Field Properties

Create a date field by specifying values for the common set of field properties, such as display label and field name.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Name	Name
Description	Name
Required	Constraints
Updatable	Constraints
Searchable	Constraints
Indexed	Constraints
Fixed Value	Default Value
Expression	Default Value

Additional Date Field Specifications

Additional specifications for this field type include the following details:

- Data type is `TIMESTAMP`.
- An object can have a total of 625 fields. Of these, 50 are reserved for date and datetime fields.
- When you create a custom subject area for custom reporting, you can select fields with this type to use for date leveling.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)

Datetime Fields

Using Application Composer, you can extend an application's object model by adding new fields to both standard or custom objects.

One such field type is datetime, where users in the runtime application can enter a date or select one from a calendar, and enter a time of day.

Datetime Field Properties

Create a datetime field by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the datetime field type.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Name	Name
Description	Name
Required	Constraints
Updatable	Constraints
Searchable	Constraints
Indexed	Constraints
Fixed Value	Default Value Note: When you select a literal date for this field's default value, the selected date appears in the international standard notation for date and time of day; however, at runtime, the date and time of day are displayed according to the user's preferences.

Field Property	Field Property Region
Expression	Default Value

Additional Datetime Field Specifications

Additional specifications for this field type include the following details:

- Data type is `TIMESTAMP`.
- A object can have a total of 625 fields. Of these, 50 are reserved for date and datetime fields.
- When you create a custom subject area for custom reporting, you can select fields with this type to use for date leveling.
- This field type supports time zone conversion.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)

Dynamic Choice Lists

Overview of Dynamic Choice Lists

Using Application Composer, you can add new fields to both standard or custom objects. One field type that you can add to a custom or standard object is a dynamic choice list.

A dynamic choice list is a field that contains a list of values which are populated from the actual data of another object. For example, you might want to expose on a user interface page a dynamic choice list which lets users specify the account that they're logging a help request against. The **Account Name** choice list is a field that you're adding to the help request object, but the list of values is populated by actual names from the account object.

When creating dynamic choice lists, note the following:

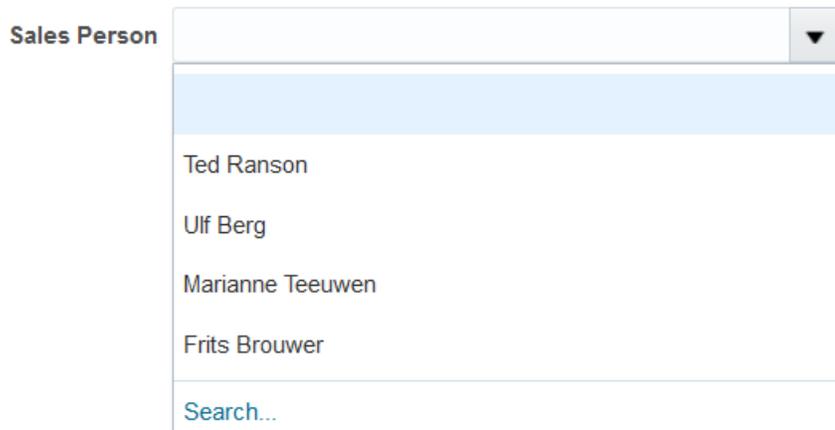
- Review the common set of field properties, as well as the dynamic choice list-specific properties, that you must specify.
- Review the options available in the List Data Source, Additional List Display Values, and Additional List Search Fields regions.
- Understand how a dynamic choice list results in the implicit creation of a relationship.

Note: You must create a Select and Search dialog box for a custom object, if you also create a dynamic choice list that's based on the same custom object. This is so that, at runtime, users can click the Search link to search for and select the value they need in the choice list field.

Using a Dynamic Choice List at Runtime

A dynamic choice list is a field that contains a list of values which are populated from the actual data of another object. At runtime, users can display the list of values, and then search for and make their selections. Let's look at an example.

Here's an example of a dynamic choice list field where users can add a salesperson to an opportunity. This is a custom field that's populated with records from the Resource object.



The image shows a web form with a field labeled "Sales Person". The field is currently open, displaying a dropdown menu. The menu contains a list of names: "Ted Ranson", "Ulf Berg", "Marianne Teeuwen", and "Frits Brouwer". At the bottom of the dropdown, there is a "Search..." link. The field has a small downward arrow icon on its right side.

Notice that each dynamic choice list field includes the following:

- A list of the most recently used records for greater efficiency.
- A search link that, when clicked, opens the Search and Select dialog where users can search for and select a record. Again, this improves efficiency by making it easier to find the right selection.

The most recently used list displays in dynamic choice list fields based on these objects: Account, Contact, Household, Opportunity, Lead, Asset, Campaign, Partner, Service Request, Product, Product Group, and custom objects. The list includes a small set of records displayed in alphabetic order. A list of values does display for custom dynamic choice list fields based on other standard objects, but the list is sorted by the sort order defined by those objects (which could be different across objects).

In the most recently used list, records are listed if the user previously selected it in one of these ways:

- By typing a value and tabbing out of the field
- By typing a partial value and selecting a record from one of the auto-suggested values
- By selecting a record from the Search and Select dialog box
- By selecting a record from the most recently used list

For Opportunity, the dynamic choice list field displays the recently searched and selected opportunities in the Search and Select dialog, but it won't display any recently created opportunities in the most recently used list. However, for Account, the dynamic choice list displays both, the recently searched and selected, and the most recently created ones.

Note: Any advanced filter condition set for the dynamic choice list isn't applied on the most recently used list.

Dynamic Choice List Properties

Create a dynamic choice list by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the dynamic choice list field type.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution.
Name	Name
Description	Name
Required	Constraints
Updatable	Constraints
Searchable	Constraints
Note: If you plan to expose this dynamic choice list as a subtab on the related object's details page, then you must make this field searchable. Fields prefixed with an * are not searchable.	

The following table shows properties that are unique to only certain field types, including dynamic choice lists:

Field Property	Field Property Region
Related Object	List Data Source
List Selection Display Value	List Data Source
Data Filter	List Data Source
Additional List Display Values	Additional List Display Values
Additional List Search Fields	Additional List Search Fields

Using the List Data Source, Additional List Display Values, and Additional List Search Fields Regions
When defining a dynamic choice list, use the following regions to determine what data will display in the list of values at runtime.

Note: Although you can configure what data will display in the list of values at runtime, you can't control the sequence of where those values display in the list.

- List Data Source region

- Related Object

The values in a dynamic choice list are populated from another object's data. Select the related object first, then use the **List Selection Display Value** choice list to select the field that you want to display within the dynamic choice list as the first column at runtime. Selecting the related object is possible only during field creation.

Note: The set of objects that are available for selection is constrained to top-level objects only. You can't select a child object as a related object.

In our example, the related object would be **Account**.

Tip: Once you create a dynamic choice list field, you can easily recognize the choice list's related object from the Fields page. The Fields page displays summaries of both standard and custom fields for the selected object. If a dynamic choice list was created, then the Type column includes the related object. In our example, the field type would be **Choice List (Dynamic) <Account>**.

- List Selection Display Value

From the **List Selection Display Value** choice list, select the related object's field that you want to display within the dynamic choice list as the first column at runtime. This is the primary field on the related object that your users will use to make the appropriate selection. In our example, the field might be something like **Name**.

Note: The fields available for selection include only the related object's required fields.

- Data Filter

You can further refine the set of data that appears within the dynamic choice list at runtime by using data filters. You can set up a simple filter to show only a subset of records in the list of values at runtime. Or, you can set up a more complex expression that dynamically filters the subset of records based on the object context. You can also use an existing filter that's predefined for some standard objects.

Ideally, set data filters on a dynamic choice list during the initial configuration of the field.

For example, use a simple filter to hide any accounts outside a particular region. Or, hide account records that are inactive. At runtime, only active accounts would appear in the dynamic choice list field.

Tip: To optimize performance, refine the list of values displayed in this field at runtime by including one or more filters based on indexed fields.

To learn more about data filters, see "Data Filters for Dynamic Choice Lists."

- Additional List Display Values region (not available for all objects; see note below)

You can further refine the look and feel of the dynamic choice list by selecting additional fields to display in the choice list.

Use the **Additional List Display Values** region to include additional related object fields in the dynamic choice list at runtime. These additional fields assist your users in making a selection from the choice list. The region doesn't include the field that you already selected in the **List Selection Display Value** choice list.

There's no limit on the number of additional fields that you can select.

Note: Fields prefixed with an * are not searchable.

- Additional List Search Fields region (not available for all objects; see note below)

You can indicate which additional related object fields can be added as search criteria in the dynamic choice list's Search and Select dialog.

Use the **Additional List Search Values** region to include additional related object fields in the dynamic choice list's Search and Select dialog, accessed using the **Search** link at runtime. The region doesn't include the field that you already selected in the **List Selection Display Value** choice list.

There's no limit on the number of additional fields that you can select.

Note: The Additional List Display Values and Additional List Search Fields regions don't display when configuring a dynamic choice list that's based on one of these objects:

- Account
- Asset
- Campaign
- Contact
- Household
- Opportunity
- Partner
- Product
- Product Group
- Resource
- Sales Lead
- Service Request

That's because these objects have corresponding search and select dialog boxes that you can configure. See the related topics "Configure a Search and Select Dialog Box for Custom Objects" and "Configure a Search and Select Dialog Box for Standard Objects."

Implicit Relationship Creation

When you create a dynamic choice list for an object based on a related object, you're implicitly creating a many-to-one foreign key relationship where the current object is the "many" object and the related object is the "one" object. This implicit creation of a relationship lets you later add a related object subtab for the "many" object on the "one" object's details page. You can view these implicitly created choice list relationships on the Relationships page.

Let's go back to our previous example where you want to let users specify the account that they're logging a help request against. You will add an **Account Name** dynamic choice list field to the help request object, and its values are populated from the account object. In this example, an account can be tied to multiple help requests. The relationship that's created is a many-to-one relationship between the help request and account objects, which enables users to store an account identifier in the help request object's table. In this relationship, the account object is the source object and

the help request object is the target object. If a source object is ever deleted, then at runtime, then the dynamic choice list will have no values in it.

Later, you might want to expose a related object subtab on the account details page which shows, for a single account, all the help requests that are related to it. You can create this Help Requests related object subtab on the Account details page because the relationship was already created when you created the Account dynamic choice list.

Note: Generally, the dynamic choice list that you create results in the implicit creation of a choice list relationship. The exception is if you're in a global single instance environment and you create a dynamic choice list between an Oracle CX Sales and Fusion Service object and a common object: resource, customer contact profile, account, address. In such cases, relationships aren't implicitly created.

Groovy Script Syntax

Once a many-to-one relationship is created between objects using a dynamic choice list field, a foreign key field is created on the "many" object. Use the API names listed in the following table to access those foreign keys in your scripts.

Relationship Type	Foreign Key API Name	Pattern Used
Dynamic choice list relationship	If the dynamic choice list field name is DynChoice1_c, then the foreign key API name is DynChoice1_Id_c.	<Name of the dynamic choice list field>_Id_c

Additional Dynamic Choice List Specifications

Additional specifications for this field type include the following details:

- Data type is VARCHAR2 (1500).
- A object can have a total of 625 fields. Out of those 625 fields, 350 fields are reserved for text and check box fields, and fixed and dynamic choice lists.
- For optimal performance, you can create a maximum of 20 dynamic choice lists fields for standard objects, and 19 dynamic choice lists fields for custom objects.

Note: Dynamic choice list fields can also have an impact on the performance of a page because they require additional queries to run. So, Oracle recommends that you limit the number of dynamic choice list fields exposed in landing and edit pages to 5.

- Dynamic choice list fields and relationships automatically use 1 indexed number field. If all indexed number fields are already taken, then Application Composer uses a non-indexed number field.

If a subtab or analysis is based off a dynamic choice list field or other relationship, then create that relationship first to ensure you obtain an indexed number field. This ensures optimal performance for your subtab or analysis.

- To report on custom dynamic choice list fields added to standard objects, you must create a custom subject area because you can't add custom dynamic choice list fields to prebuilt subject areas. You can then create a union report in Analytics Answers that joins the desired prebuilt subject area with your new custom subject area.

- When a dynamic choice list field is created, a reverse accessor isn't automatically created for the field's target object. This means that, when using a REST GET call to get the target object's data, the data won't be available in the output.

Related Topics

- [Data Filters for Dynamic Choice Lists](#)
- [What's the difference between fixed choice lists and dynamic choice lists?](#)
- [Object Relationships](#)
- [About Search and Select Dialog Boxes](#)
- [Configure a Search and Select Dialog Box for Custom Objects](#)
- [Configure a Search and Select Dialog Box for Standard Objects](#)

Data Filters for Dynamic Choice Lists

When creating a custom dynamic choice list, also known as a list of values, you can filter the values that display in the list at runtime. For example, set up a simple filter to show only a subset of records in the list.

Understanding Data Filters

When you first define a dynamic choice list, you indicate which related object's records populate the custom field's list of values. You can further refine the set of data that appears within the dynamic choice list at runtime by using data filters. Ideally, set data filters on a dynamic choice list during the initial configuration of the field.

To refine the set of data that appears within the dynamic choice list, select from one of three options:

- Simple mode
- Advanced mode
- Existing filter

Tip: When defining a simple or advanced filter, use indexed fields to optimize performance.

Defining Simple Data Filters

A simple data filter uses static values that you provide as search criteria to refine the list of values at runtime. For example, when editing an opportunity, the list of accounts that display in the Account dynamic choice list can be filtered to show only active accounts.

To define a simple data filter:

1. In Application Composer, create a custom dynamic choice list field.
2. After you select the related object and the object's field that you want to display in the list of values, navigate to the Data Filter region.
3. In the Data Filter region, click **Add Search Field**.
4. Select the field that you want to use as the simple filter, then specify the criteria to apply at runtime.

For example, if you select **City**, then specify a city name, such as New York.

You can select more than one field.

5. Complete the rest of the field's configuration, then click **Submit**.

The filter's static value is applied to the set of records in the dynamic choice list at runtime to refine the list of values.

Defining Advanced Data Filters

An advanced data filter uses dynamic criteria based on the object context to refine the list of values at runtime. As part of the advanced data filter, you can write Groovy expressions and accept bind variables.

To define an advanced data filter:

1. In Application Composer, create a custom dynamic choice list field.
2. After you select the related object and the object's field that you want to display in the list of values, navigate to the Data Filter region.
3. In the Data Filter region, click the **Advanced Mode** link that displays in the instructions.
4. Enter a search expression to refine the records that appear in the list of values. As you write your script:
 - o Click **Add Search Field** to select appropriate fields.
 - o Click **Add Bind Variable** to pass the target object context to the filtering expression.

For example, when editing a lead, the list of campaigns that display in the Campaign dynamic choice list can be filtered by adding bind variables that pass context from the lead record to the Campaign dynamic choice list.

5. Check the **When filtering data, ignore expressions involving null bind variable values** check box.

This check box defines the behavior of the expression when a bind variable evaluates to null.

- o If this box is checked, then any atomic expression ("FieldName Operator Operand") whose bind variable evaluates to null is ignored. The other atomic expressions that are part of the filter remain.
 - o If this box isn't checked and the bind variable evaluates to null, then the dynamic choice list will be empty.
6. Complete the rest of the field's configuration, then click **Submit**.

CAUTION: Don't define advanced data filters if you're working with a dynamic choice list that's based on one of these objects:

- Account
- Asset
- Campaign
- Contact
- Household
- Opportunity
- Partner
- Product
- Product Group
- Resource
- Sales Lead
- Service Request

That's because these objects have corresponding search and select dialog boxes that you can configure. Instead of defining data filters, prepopulate the field's corresponding search and select dialog box with the records you want your users to see first. See the related topic, "Prepopulate a Search and Select Dialog Box with Default Values."

Using Existing View Criteria Filters

When defining a data filter for a dynamic choice list, you can select from a predefined list of existing view criteria filters. These existing filters are the named view criteria that are already provided for some standard objects. Some standard objects may predefine named view criteria which you can use to simplify common searches.

To select an existing view criteria filter:

1. In Application Composer, create a custom dynamic choice list field.
2. After you select the related object and the object's field that you want to display in the list of values, navigate to the Data Filter region.
3. In the Data Filter region, click the **Existing Filter** link that displays in the instructions.
4. Select the desired filter.

5. Complete the rest of the field's configuration, then click **Submit**.

Related Topics

- [Overview of Dynamic Choice Lists](#)
- [Prepopulate a Search and Select Dialog Box with Default Values](#)

Fixed Choice Lists

Using Application Composer, you can extend an application's object model by adding new fields to both standard or custom objects. One such field type is a fixed choice list, a field that contains a list of static values populated from FND lookup types.

At runtime, users can select one or more values from this field, depending on the field's definition.

Fixed Choice List Properties

Create a fixed choice list by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the fixed choice list field type.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution.
Name	Name
Description	Name
Required	Constraints
Updatable	Constraints
Searchable	Constraints
Fixed Value	Default Value

Field Property	Field Property Region
<p>You can't set a default value for any fixed choice list that's constrained by another fixed choice list.</p> <p>If the choice list allows multiple values, then you can still write an expression to preselect multiple values by default.</p> <p>For example, if the field includes three lookup codes with (Code,Label) of (S, Small),(M,Medium),(L,Large), and (XL, Extra Large), then to preselect the Small and Extra Large selections by default, set the default value to the literal string (without quotes): S,XL.</p> <p>The defaulted values for the multiple-select fixed choice list is stored in a comma-separated format. In the previous example, "S,XL" will be stored in the database.</p>	

The following table shows properties that are unique to certain field types, including fixed choice lists:

Field Property	Field Property Region
<p>Display Type</p> <p>Indicate if users can select a single value or multiple values from the choice list at runtime. You can only select the display type during field creation.</p> <p>Note: If you create a multiple-select fixed choice list, then don't use commas in the actual lookup codes for the lookup type that populates this field.</p>	Appearance
<p>Lookup Type</p> <p>When creating a lookup type, don't use special characters or spaces in the lookup type itself. Underscores are permitted.</p> <p>Also, don't create a lookup type with a name ending in "LOOKUPTYPE". If you do, then you won't be able to see this extension in Analytics Answers and reporting.</p>	List of Values
<p>Constrain List by Parent Field Value Selection</p>	List of Values

Selecting the List of Values for the Fixed Choice List

The values in a fixed choice list are populated from FND lookup types. Select the lookup type with values you want to display in this choice list. The fixed choice list will display the lookup type's values in a single column; it's not possible to add more columns to a fixed choice list.

You can select the lookup type only during field creation. A fixed choice list can have a maximum of 1,000 values.

Some additional points about selecting a lookup type for a fixed choice list:

- The set of FND lookup types available for selection is constrained to those lookup types related to this fixed choice list's object (by product code), as well as all custom lookup types that have been created for your implementation.
- When entering a lookup type, you can select the **Edit** icon to modify the existing values if required.
- You can also create a new lookup type from this page.

If creating a new lookup type, then don't use special characters or spaces in the lookup type itself. Underscores are permitted.

▲ List of Values
Configure the list of values you want to display in the choice list. Click the search icon to select a predefined lookup type, or create a new one.

* Lookup Type

Constrain list by parent field value selection

Select the parent field whose value selection will drive the contents of this field, and then define the mapping between the parent and child values.

* Parent Choice List

* Value Map

Mapping List Values to Parent Values

You can constrain the actual values that display in the fixed choice list at runtime by relating the fixed choice list to a parent fixed choice list. The value selected in the parent fixed choice list drives the values that display in this fixed choice list. This is usually referred to as a dependent fixed choice list, or dependent LOV.

For example, you might want your users to see two choice lists when creating a trouble ticket: one field for specifying the trouble ticket type and one field for specifying the trouble ticket area. If a user selects **Hardware** from the **Type** choice list, then the **Area** choice list should contain only hardware options that the trouble ticket can be logged against, such as Desktop or Laptop.

To do this, while creating the **Area** fixed choice list, select the **Constrain List by Parent Field Value Selection** check box, select the **Type** parent field, and then map the values between the parent lookup type and this field's lookup type.

To implement the previous example:

1. Define the **Type** fixed choice list.

2. Define the **Area** fixed choice list.

- a. Select the **Constrain List by Parent Field Value Selection** check box and select the parent field, **Type**.

You can select the **Constrain List by Parent Field Value Selection** check box only during field creation, and only if at least one other single-select fixed choice list has been defined. After field creation, however, you can always update the mapping between lookup values.

- b. Map the values between the Type and Area lookup types.

For example, map all relevant hardware values in the Area lookup type's set of values, such as **Desktop** and **Laptop**, to the value of **Hardware** in the Type's lookup type.

If the parent lookup type is set-enabled and you use multiple sets, then there's some additional setup to complete. Parent lookup values are recognized only if those values are assigned to the logged-in user's set ID. This means that, for parent lookup types that are set-enabled, you must assign the lookup type's common values to all set IDs. You do this in the Manage Set-enabled Lookups task in the Setup and Maintenance work area.

After your users start using these fields to enter data, don't change a lookup type's lookup code values. For example, don't change **LAPTOP_CODE** to **LAPTOP_CODE1**. If you change a lookup type's lookup code values, then you will need to manually re-map all records that already reference the original code, as well as re-map the values between lookup types in the child fixed choice list.

Note: You can't set a default value for any fixed choice list that's constrained by another fixed choice list.

Additional Fixed Choice List Specifications

Additional specifications for this field type include the following details:

- Data type is VARCHAR2 (1500).
- An object can have a total of 625 fields. Of these, 350 are reserved for text and check box fields, and fixed and dynamic choice lists.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)
- [What's the difference between fixed choice lists and dynamic choice lists?](#)
- [Overview of Lookups](#)

Formula Fields

Using Application Composer, you can extend an application's object model by adding fields to both standard or custom objects. One such field is a formula field, which is calculated in the runtime application using the Groovy-based expression included in the field's definition.

Formula Field Properties

Create a formula field by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the formula field type.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution.
Name	Name
Description	Name

The following table shows properties that are unique to certain field types, including formula fields:

Field Property	Field Property Region
Formula Type Specify the field's data type, such as text, number, or date. You can specify the type only during field creation.	Field Value Type
Display Type If the formula type is Text , then indicate if you want this formula field to render in the runtime application as a simple text box, or if the field should allow multiple lines where text can wrap.	Appearance
Depends On	Constraints

Using the Expression Builder and the Depends On Choice List

Use the **Depends On** choice list to indicate if the field should be automatically recalculated (using the expression you write) if another field's value changes.

Note: The **Depends On** choice list includes a list of fields that belong to the same object. If you want this formula field to automatically recalculate if the value of another field on a different object changes, then you must write a server script.

Use the expression builder to write an expression that calculates the field's value at runtime.

This list provides several examples of when to use the expression builder:

- If your expression calculates the value of an employee's annual bonus amount, set the expression to automatically recalculate that amount if the employee's salary changes.
- If your expression determines the correct customer phone number to use for an opportunity, set the expression to automatically reset the phone number if the opportunity's customer account changes.

Note: Use formula fields only for deriving a field's value, and not for setting any additional field values.

Additional Formula Field Specifications

Additional specifications for this field type include the following details:

- Data type is set by the **Formula Type** property.
- The formula field type isn't supported by subject areas. You can't add formula fields to a custom report.
- You can't search on a formula field.
- A formula field is a computed attribute, and exists only at runtime. This is a transient type of attribute that doesn't persist in the database as a table column. Hence, no maximum number of formula fields exists for an object.
- A formula field's Groovy script is evaluated every time the field's value is requested by any layer. Accordingly, don't write process-intensive scripts, such as calling external web services, as this could impact the page's performance.
- Don't use a formula field to set other fields' values because, due to the order of rendering, the order in which the fields are processed isn't guaranteed. If you want to write code that derives other field values when the value of some other field is changed, use the **After Field Changed** trigger documented in the Groovy Scripting Reference guide.
- Use simple computation logic inside formula fields.
- Use formula fields only for deriving and not for setting any additional field values.
- Formula fields are not editable, but don't use them for displaying a custom field as read-only.
- When writing a Groovy expression for a formula field, don't refer to transient attributes (for example, the display label of a dynamic choice list field) in the formula logic.

- Never use a 'setAttribute()' inside a formula field. This can cause a record lock issue due to the repeated evaluation of the formula field at various times of the page rendering.
- If a formula field calculation involves querying objects, then avoid exposing such formula fields on overview pages (Summary table) and on subtabs. If this type of formula field is exposed as a column in a table, the formula is evaluated and its query is fired for each row. This will impact the performance of a List page. If a formula involves a complex query, expose the formula field on the Object Detail page, not on the List page.
- Avoid including formula fields in REST calls if your specific business case doesn't require you to do so. You can ensure this by clearing the **Include in Service Payload** check box.
- When using formula fields to evaluate values in child collection in REST, all the formula fields in the child are evaluated for every child record causing performance degradation.

To improve performance, instead of using formula fields to implement logic like executing an SQL query, use a persistent field like a text field and add any logic to the insert or update trigger on that field and set the value to that field.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)
- [Groovy Scripting](#)

Joins and Join Fields

A join is a predefined association between an object and its related object. Joins use underlying, preexisting relationships already delivered with your cloud service. You use joins to add related object fields to an object's work area.

Before you can do that, however, you must register those fields, either custom or standard, by creating join fields. (Join fields aren't provided automatically for you.)

Understanding Joins

Joins are view links between an object and another top-level object, which are already related through an existing many-to-one or one-to-one relationship. Joins let you display a related object's fields on an object's work area.

Note: You can't create joins or edit existing joins.

Why Use Joins?

Joins use preexisting relationships between an object and its related object. Joins give you more flexibility than relationships provide. With relationships, you can only add child or related object subtabs to an object's details page. Joins, on the other hand, let you add related object fields to any page of an object's work area, not just the details page.

For example, in Oracle CX Sales, the Account object and the Opportunity object are related objects by default and are already delivered with a join. If you register the Account's Primary Phone field as a join field for the Opportunity object, then you can display that field anywhere on the Opportunity work area, such as on the Create Opportunity page or Opportunity landing page.

Tip: The other benefit of joins and join fields is that, at runtime, when your end users enter data into the Primary Phone field, the data is actually written to the underlying Account object.

Choosing a Join

To view the joins available for an object, expand the object and click the Joins node. For example, expand the Opportunity object, and click the Joins node. Select the desired join row and click Edit to navigate to the read-only Join Specification page, where you can review details about the join.

Joins are delivered by default for some objects. (Not every object has a Joins node.) For example, these CX Sales objects are delivered with one or more joins:

- Customer Contact Profile
- Forecast Item
- Opportunity
- Partner
- Product Group
- Program Enrollments
- Sales Account

Note: The Sales Account object is available for change only to existing users from prior releases. If the Sales Account object is read only for you in Application Composer, then this means that you can't extend the Sales Account object. Instead, use the Account object only.

- Sales Lead Contact

Registering Join Fields

Joins are delivered without join fields. Before you can add related object fields from a join to an object's work area, you must select the related object fields that you want to display, and then register them as join fields.

To register a join field:

1. Expand an object and click the Joins node.
2. Click the join name to navigate to the Join Fields page, where you can register join fields.

Working with Join Fields

Once you have registered a related object field as a join field, you can then show or hide those fields on an object's work area by using the configuration pages available from the object's Pages node. There are some caveats about working with join fields, listed below.

- Join fields that are based on a dynamic choice list field aren't exposed as searchable fields in Application Composer. This means that when you configure the local search, regional search, Search and Select dialog, or a context link subtab, join fields based on dynamic choice list fields aren't available for selection.

However, you can still filter the records that display in an object's summary table by using the Query By Example feature. At runtime, click the Query By Example icon on the table's toolbar, and enter a value for the join field column.

- Fields configured for an object as a join field don't appear in file-based import and bulk export.

- Join fields are computed attributes, and exist only at runtime. This is a transient type of attribute which doesn't persist in the database as a table column. Hence, there is no maximum number of join fields for an object.
- Generally, a join field on a record is automatically updated when a user updates that field elsewhere on the record. For optimal performance, however, this automatic update doesn't happen on Opportunity and Lead pages.

For example, let's say a user adds a new contact to an opportunity and saves the opportunity. Then, still on the Contacts subtab, the user adds a primary phone number. After saving the record, the Primary Phone field (the join field) on the opportunity's Summary tab doesn't automatically update. To see the new telephone number on the Summary tab, the user must requery the opportunity.

Related Topics

Long Text Fields

Using Application Composer, you can extend an application's object model by adding fields to both standard or custom objects. One such field type is a long text field, where users in the runtime application can enter a combination of letters, numbers, or symbols.

This field type supports 32,000 characters.

Long Text Field Properties

Create a long text field by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the long text field type.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution.
Name	Name
Description	Name
Required	Constraints
Updatable	Constraints

Field Property	Field Property Region
Fixed Value	Default Value
Expression	Default Value

The following table shows properties that are unique to certain field types, including long text fields:

Field Property	Field Property Region
<p>Display Type</p> <p>Indicate how you want this text field to render in the runtime application:</p> <ul style="list-style-type: none"> As a simple text box. Allowing multiple lines where text can wrap, or where the user can enter carriage returns. 	Appearance

Additional Long Text Field Specifications

Additional specifications for this field type include the following details:

- Data type is CLOB.
- A object can have a total of 625 fields. Of these, 25 are reserved for long text fields.
- For optimal performance, you can create a maximum of 5 long text fields for both standard and custom objects. To change this limit across all objects, contact Oracle Support.
- The long text field type is unavailable for use with custom subject areas and Oracle Social Network:
 - Long text fields aren't available for inclusion in custom reports.
 - Long text fields aren't available for sharing in OSN conversations.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)

Number Fields

Using Application Composer, you can extend an application's object model by adding fields to both standard or custom objects. One such field is a number field, where users in the runtime application can enter a number.

Number Field Properties

Create a number field by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the number field type.

The following table shows properties that are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution.
Name	Name
Description	Name
Required	Constraints
Updatable	Constraints
Searchable	Constraints
Indexed	Constraints
Fixed Value	Default Value
Expression	Default Value

The following table shows properties that are unique to certain field types, including number fields:

Field Property	Field Property Region
Decimal Places The number of digits that can be entered and displayed to the right of the decimal point. If at runtime, a user enters more digits after the decimal point, then Application Composer rounds up (using	Constraints

Field Property	Field Property Region
<p>the tie-breaking rule, round half up) to derive the field's value.</p> <p>For example, if you enter 2 for the number of decimal places, then at runtime, an entry of 4.986 is displayed as 4.99.</p>	
<p>Maximum Length</p> <p>The number of digits a user can enter in the field. This number should be greater than or equal to 1 and less than or equal to 38.</p> <p>Maximum Length - Decimal Places = the number of digits that can appear to the left of the decimal point.</p> <p>Don't set a maximum length shorter than the number of decimal places.</p>	Constraints
<p>Minimum Value</p> <p>The minimum numeric value that a user can enter into this field.</p>	Constraints
<p>Maximum Value</p> <p>The maximum numeric value that a user can enter into this field.</p>	Constraints

Additional Number Field Specifications

Additional specifications for this field type include the following details:

- Data type is NUMBER.
- An object can have a total of 625 fields. Of these, 200 fields are reserved for number, currency, and percentage fields.

Note: Of the 200 fields reserved for number, currency, and percentage fields, 15 fields are reserved for indexed number fields in a custom parent object (10 fields for custom child objects). This means you can create a total of 185 non-indexed fields. For standard objects, you can index two text fields and three number fields (shared among number, percentage, and currency fields). Note that you can index up to 8 number fields for the Opportunity object, and up to 16 number fields for the Sales Lead object.
- Leading zeros are removed.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)

Percentage Fields

Using Application Composer, you can extend an application's object model by adding fields to both standard or custom objects.

One such field type is a percentage field, where users in the runtime application can enter a percentage. Application Composer automatically adds the percent sign to the number.

Percentage Field Properties

Create a percentage field by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the percentage field type.

The following properties are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance <div style="border-left: 2px solid orange; padding-left: 10px; margin-left: 20px;"> Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution. </div>
Name	Name
Description	Name
Required	Constraints
Updatable	Constraints
Searchable	Constraints
Indexed	Constraints
Fixed Value	Default Value
Expression	Default Value

The following properties are unique to only certain field types, including percentage fields:

Field Property	Field Property Region
<p>Decimal Places</p> <p>The number of digits that can be entered and displayed to the right of the decimal point. If at runtime, a user enters more digits after the decimal point, then Application Composer rounds up (using the tie-breaking rule, round half up) to derive the field's value.</p> <p>For example, if you enter 2 for the number of decimal places, then at runtime, an entry of 4.986 is displayed as 4.99.</p>	Constraints
<p>Maximum Length</p> <p>The maximum digits a user can enter in the field.</p> <p>Maximum Length - Decimal Places = the number of digits that can appear to the left of the decimal point.</p> <p>Don't set a maximum length shorter than the number of decimal places.</p>	Constraints

Additional Percentage Field Specifications

Additional specifications for this field type include the following details:

- Data type is NUMBER.
- A object can have a total of 625 fields. Of these, 200 are reserved for number, currency, and percentage fields.
- Application Composer automatically adds the percent sign.
- For custom fields, the percentage value is stored as the absolute value. For example, 100%=1 and 60%=0.6. Application Composer multiplies it by 100 and displays the converted percentage value.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)

Record Type Fields

Using Application Composer, you can add a record type field to both standard or custom objects. A record type field contains a list of static values which are populated from FND lookup types.

This type of field is useful because you can associate each choice list value with a role or a page layout. Associating a choice list value with a role means that only users with that role can see the choice list value. Associating a choice list value with a page layout means that once that value is selected at runtime, the associated page layout displays. This makes a record type field more powerful than a fixed choice list field or a dynamic choice list field.

Using Record Type Fields

Create a record type field, so that you can associate each choice list value with a role or a page layout.

You can:

- Associate each choice list value with a role.
 - a. You do this while you're creating the field.
 - b. At runtime, when an end user views the list of values in the field, the available choices are constrained according to the user's role.

Custom roles, which are copies of the predefined roles that Oracle provides for all customers, are displayed by default. However, you can optionally choose to display predefined roles, as well.

- Associate each choice list value with a page layout.
 - a. You do this by adding the field to a application page layout, after you have created the field.
 - b. You must then assign a choice list value to the page layout.
 - c. At runtime, when an end user selects a value from the field, the page display changes to match the application page layout that you associated with the choice list value.

Note: You can create only one record type field per object, and only once the object has a work area. If the object's work area hasn't been created yet, then you must create the work area first, before creating the record type field.

Record Type Field Properties

Create a record type field by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the record type field type.

The following properties are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution.
Name	Name
Description	Name
Required	Constraints

Field Property	Field Property Region
Updatable	Constraints
Searchable	Constraints
Fixed Value	Default Value

The following properties are unique to only certain field types, including record type fields:

Field Property	Field Property Region
Lookup Type Selecting the lookup type is possible only during field creation.	List of Values
Available Record Types Indicate the choice list values that each role has access to. For example, perhaps the sales representative can see only selected choice list values, but the sales manager can see all the choice list values.	Roles
Default Record Type Indicate the choice list value that each role will see by default at runtime.	Roles

Using the List of Values Region

The values in a record type field are populated from FND lookup types. Select the lookup type whose values you want to display in this choice list.

You can also select a lookup type and select the Edit icon to modify the existing values.

Note: The lookup types available for selection are limited to:

1. Standard lookup types that are related to this record type field's object (by product code).
2. All custom lookup types that have been created for your implementation.

Or, create a new lookup type and add new values to it.

Additional Record Type Field Specifications

Additional specifications for this field type include the following details:

- Data type is VARCHAR2 (1500).

- A record type field is optional, and isn't required for an object.
- One record type field is allowed per object, and it will be one of the 350 fields reserved for text and check box fields, and fixed and dynamic choice lists.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Use Field Values to Control a Page Display](#)

Text Fields

Using Application Composer, you can extend an application's object model by adding new fields to both standard or custom objects. One such field type is a text field, where users in the runtime application can enter a combination of letters, numbers, or symbols.

Text Field Properties

Create a text field by specifying values for the common set of field properties, such as display label and field name. You also set properties for this field that are specific to the text field type.

The following properties are common across multiple field types:

Field Property	Field Property Region
Label	Appearance
Help Text	Appearance
Display Width	Appearance Note: Do not use. A field's display width is dynamically calculated based on the field type and resolution.
Name	Name
Description	Name
Required	Constraints
Updatable	Constraints
Searchable	Constraints
Indexed	Constraints

Field Property	Field Property Region
Fixed Value	Default Value
Expression	Default Value

The following properties are unique to only certain field types, including text fields:

Field Property	Field Property Region
<p>Display Type</p> <p>The way you want this text field to render in the runtime application:</p> <ul style="list-style-type: none"> • As a simple text box. • Allowing multiple lines where text can wrap or where the user can enter carriage returns. 	Appearance
<p>Maximum Length</p> <p>The maximum number of characters that a user can enter in the field. You can set a maximum length of 1500 characters. If the field is a multiline field, then carriage returns are permitted and count as part of the total.</p> <p>Note: The 1500-character limit applies if the characters are single byte. If the characters are multibyte, such as Japanese or Chinese, then the maximum character limit is 1500 characters divided by the number of bytes per multibyte character. For example, if characters are 2 bytes, then the name is limited to a maximum of 750 characters. If a mix of characters is used, then the maximum sum of character bytes that's supported is 1500.</p>	Constraints
<p>Minimum Length</p> <p>The minimum number of characters that a user can enter into the field.</p>	Constraints

Additional Text Field Specifications

Additional specifications for this field type include the following details:

- Data type is VARCHAR2 (1500 char).
- A object can have a total of 625 fields. Of these, 350 are reserved for text and check box fields, and fixed and dynamic choice lists.

Related Topics

- [Define Fields](#)
- [Field Types and Field Properties](#)

How do I enable tagging of service requests?

The Tag field isn't displayed automatically on some pages, such as for service requests (SRs). The Tag field is predefined, but it's not added to any of the predefined standard layouts.

Let's say you want the **Tag** field to be available to all users. Then you must add this field to your administrator-defined layout and publish it.

If tags are enabled and exposed, users can create their own tags. Tags aren't striped by business unit. So all tags are visible to all users.

Let's look at an example of how to enable tags so that the **Tag** field is displayed when creating or editing SRs:

1. Change the corresponding page layout in Application Composer by adding the **Tag** field to the Service Request Summary page.

For more information about changing page layouts, see the Extend Application Pages chapter in this guide.

2. Publish the sandbox.

Tags are now enabled and the **Tag** field is available. Anyone who has the Edit Service Request privilege can add a tag.

Create Predefined Tags

As an administrator, you have the option to create predefined tags that are visible to the users. Users can also create tags at runtime to suit their needs.

Again, let's look at a service request example. Based on common requirements, you can create tags such as performance and setup. You can then send a notification to the agents suggesting that they use these predefined tags. For example, for service requests related to performance, use the `performance` tag.

To create an administrator-defined tag:

1. In the Setup and Maintenance work area, go to the following:
 - o Offering: Service
 - o Functional Area: Productivity and Tools
 - o Task: Manage Tags

The Manage Tags page is displayed, and it shows the list of existing administrator-defined tags and user-defined tags.

2. Click **Create**.
3. In the **Create Tag** dialog box, specify a tag name in the **Tag** field.
4. Click **Save and Close**.

The new tag appears on the Manage Tags page.

Import and Export Tags

You can export the administrator-defined tags from your test environment to your production environment by using the import-export feature in Functional Setup Manager. User-defined tags can't be imported and exported. Many tags may be created for testing in the test environment, but it's not required to migrate all the tags to production.

For more information about importing and exporting setup data, see the *Oracle Applications Cloud Using Functional Setup Manager* guide.

If you export your setup data by using an Implementation Project, then the administrator-defined tags are exported with the rest of the tasks in the Implementation Project in the following cases:

- Your Implementation Project includes the **Productivity Tools** Functional Area.
- You explicitly include the **Manage Tags** task.

Delete Tags

You can delete tags that are no longer required. For example, tags with typos. When you delete a tag, it's removed from all the associated records and is then deleted.

Let's look at an example of deleting a tag for service requests:

1. In the Setup and Maintenance work area, go to the following:
 - o Offering: Service
 - o Functional Area: Productivity Tools
 - o Task: Manage Tags

The Manage Tags page is displayed, and it shows the list of existing administrator-defined tags and user-defined tags.

2. Select the row with the tag that you want to delete.
3. Click **Delete**.

A message is displayed, stating that all the references to the tag will be removed.

4. Click **Delete** to delete the tag.

Assign the Create Tag Privilege for Creating Service Request Tags

A user must have the Create Tag privilege to create a tag when they're on the SR page. The Create Tag privilege is provided to all the predefined service roles by default. But as an administrator, you can remove this privilege from any role. So you have more control over who can create tags. A user without the Create Tag privilege can't create tags, but can associate existing tags with the SR as long as they can create or edit an SR.

Note: If you're using the predefined service roles such as Customer Service Representative and Customer Service Manager, then you don't need to do anything. These roles already have the Create Tag privilege.

But if you have custom roles for your service users, then you must do one of the following:

- If you don't want your users to create their own tags while editing an SR and instead associate only the existing tags to the SR, then you don't need to make any changes.
- To let your users create their own tags when editing an SR, then you can assign the Create Tag privilege to them.

To assign the Create Tag privilege to a role:

1. Navigate to the Security Console.
2. Find the custom role equal to the predefined Service Request Troubleshooter duty role.
This duty role would typically be associated with your custom Customer Service Representative job role.
3. Edit the role and navigate to the Function Security Policies section of the wizard.
For more information, see the Edit Job or Abstract Roles topic in the Oracle Fusion Securing Fusion Sales and Fusion Service guide.
4. Click **Add Function Security Policy**.
5. Search for the Create Tag privilege.
6. Add the Create Tag privilege to the role.
7. Submit your changes.

Note: If you have a custom version of the Service Request Power User duty role, then you can follow the same steps as above to assign the Create Tag privilege to that role. This duty role would typically be associated with your custom Customer Service Manager job role.

Actions and Links

You can add actions, such as buttons and menu items, to detail pages, list pages, and so on. You can also create special fields, rendered as links, that are displayed with other fields throughout the application.

You can base an action on a script (a Groovy method that's defined on the object) or on a URL defined by a script. After you create an action, it can be exposed as a button or an option on the Actions menu. A button can perform an action or navigate the user to another page in the runtime application, or to another Web site. For example, you might want to include a button on a summary table, which users can click at runtime to create a new type of record from a selected row, such as escalating an existing "trouble ticket" to a more severe "case" that can be managed separately.

After creating a link, you can select it as a field for display at runtime. For example, you might want to provide a static link from an overview page to a corporate Web site.

Add Actions or Links

You add actions or links in two steps:

1. Define an action or link for an object.
2. Add that action or link to a page layout.

You can also manage the Actions menu by hiding or showing menu items, rearranging the action groupings or display sequence, and managing the toolbar by hiding or showing icons and buttons.

Note: Custom actions with **Save and Close** and **Save and Edit** are only available for the below objects:

- Service Request
- Business Plan
- Sales Objective
- Opportunity

Define Actions or Links

To define an action or link for an object:

1. On the main Overview page in Application composer, select a standard or custom object in the object tree.
2. Select the **Actions and Links** node.
3. In the Create Action or Link page, enter a descriptive name in the Display Label field.
4. In the Type field, select either **Action** or **Link**.
5. In the Source field, select either **Script** or **URL**.
6. In the Script region click the **New** icon to build your script.
 - If the source is a URL, you can enter a static URL enclosed in double quotation marks. Or, you can define the URL by using the expression builder, which provides access to this object's fields to assist you in constructing the URL. If this object has a parent or relationship with a source object, then optionally change the context to access another object's fields for URL definition.

Any new functions that you create will be added to the **Method Name** choice list. If functions were already created for the object, then you can select one of them from the Method Name choice list. Object functions that are created elsewhere through other flows, such as server scripts, can also be used here.

To switch the context to the object's parent or related source object, for access to the object's fields for the URL definition, check the **Select alternative context** check box.

Note: Since the script deriving the URL can execute multiple times during the page life cycle, ensure that your script is safe to run multiple times. For example, don't write custom business logic to update object records.

- If the source is a script, you can either select a predefined object function from the **Method Name** choice list, or create a new object function using the expression builder. Any new functions that you create will be added to the **Method Name** choice list.

If functions were already created for the object, then you can select one of them from the **Method Name** choice list. Object functions that are created elsewhere through other flows, such as server scripts, can also be used here.

Note: Avoid writing a process-intensive script that references external web services. Repeated executions of such scripts can cause errors and potentially degrade performance.

When creating custom actions based on a script for top-level custom objects, you can specify how you would like the action to conclude at runtime, after the script completes:

- Save the record and return to the previous page (save and close)
- Save and continue editing the record (save and continue)
- Perform the action but don't save the record (run the script only)

Note: You can also specify this same information for standard objects, although some standard objects don't support the "save and close" and "save and continue" options. If you use one of these options with an unsupported standard object, then you won't be able to select and add this action to a page layout.

Display Actions or Links on Pages

After you save actions or links, you can expose them on page layouts.

When displaying a link, you select it just as you select to display standard or custom fields. This is because, at runtime, the UI displays the URL link as if it's a field in a table.

You can add actions to two places in the UI: on the toolbar as a button and in the Actions menu for a table.

Tip: To support functions that don't need to be displayed prominently on the page, add actions as options on the Actions menu. To support key functions that are frequently executed by your users, add actions as buttons. When displaying actions as buttons, be sure to test your page at runtime (in all supported languages) to confirm that the presentation of buttons is as expected. Button display could be unexpected due to the available space on the page at runtime, the number of buttons on the page, and button width (which depends on label length). If you add more buttons than the toolbar has space, then at runtime the buttons are stacked and made available using a drop-down button.

You can display actions as either buttons or Actions menu items in a variety of locations:

- Summary table on the overview page
- Default summary on the details page
- Summary table on a details page's subtab
- Revenue table on the details page for the opportunity object

Note: If you add a custom button to a table and, at runtime, that table has no rows, then the button is automatically disabled.

You can display links for an object in a variety of locations in that object's work area. You can add a link wherever you can add a field. Possible locations include, but aren't limited to:

- As a column in the summary table on the overview page
- Default summary on the details page
- As a column in the summary table on a details page's subtab
- In the detail form under the summary table on a details page's subtab
- As a column in the summary table on a tree node page for a child object
- As a column in the revenue table on the details page for the opportunity object

Delete Unpublished Actions and Links

You can use Application Composer to delete any unpublished actions and links. Any exposure of the actions (as UI buttons or action menus) and links (as fields) in the same object's extensible pages or as a detailed subtab under another object's page are also automatically deleted.

To delete unpublished actions or links for an object:

1. On the main Overview page in Application Composer, expand the standard or custom object whose actions or links you want to delete.
2. Select the **Actions and Links** node.

Application Composer lists all the actions and links defined for the selected object.
3. Select the action or link that you want to delete and click the delete icon.
4. Click **OK** on the confirmation dialog.
5. To verify that the deleted actions and links no longer appear in the object's pages, click the Pages node and review the page layouts.

Points to Consider

When defining actions, here are some important points to consider:

- If you define a custom action and expose it on a list, ensure that you do the following:
 - a. Include a check for the active record row, and ensure that the UI supports users selecting any record as the active row before invoking the custom action.
 - b. Test the performance of the script behind the action. Actions on the list page are synchronous calls which means that long-running scripts will block the page from displaying.
- Don't create custom buttons to populate the mandatory or required fields on the UI. End users must enter the values in the mandatory fields manually.
- Avoid creating an action that, when invoked from a record, calls an external web service to update the same record currently open. Instead, consider calling the external web service asynchronously using an object workflow.

Related Topics

- [Create and Add Custom Links to Application Pages](#)

Overview of Smart Actions

Smart actions are contextual actions that are mapped to an object and client application in classic Oracle Sales and in Oracle Sales in the Redwood User Experience. Smart actions can call a REST service or object function. If you're using Sales in the Redwood User Experience, then smart actions can also navigate users to a new UI location.

Standard objects come with predefined smart actions. You can also create custom smart actions for both standard and custom objects.

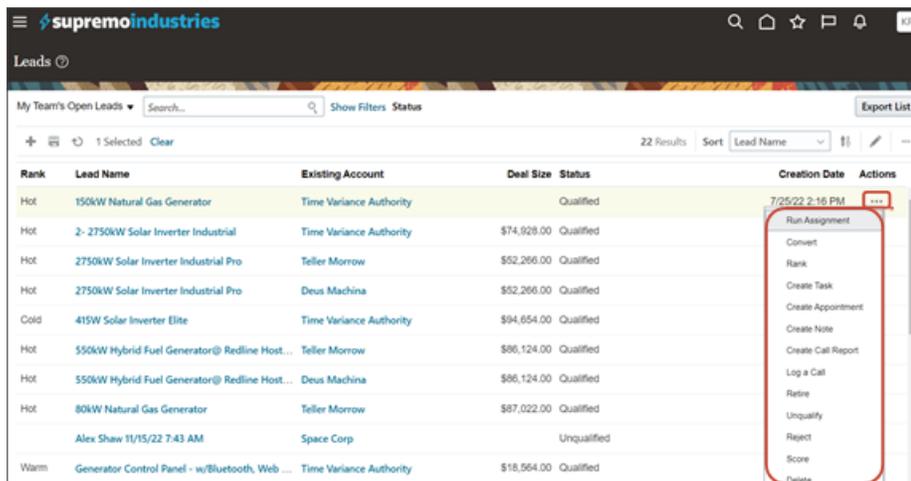
You can create:

- REST-based actions
 - Create a smart action that directly calls a service without any UI.
- Object function-based actions
 - Create a smart action that directly calls an object function without any UI.
- UI-based actions
 - Create a smart action that directly calls another UI inside the application.

Tip: When moving configurations from test to production environments, your migration set includes smart actions.

Smart Actions in Workspace

In the Workspace, a feature of classic Sales, smart actions display in the Actions column on the list page.



Smart Actions in the Redwood Action Bar

If you're using the Redwood user experience, then smart actions display in the Action Bar.

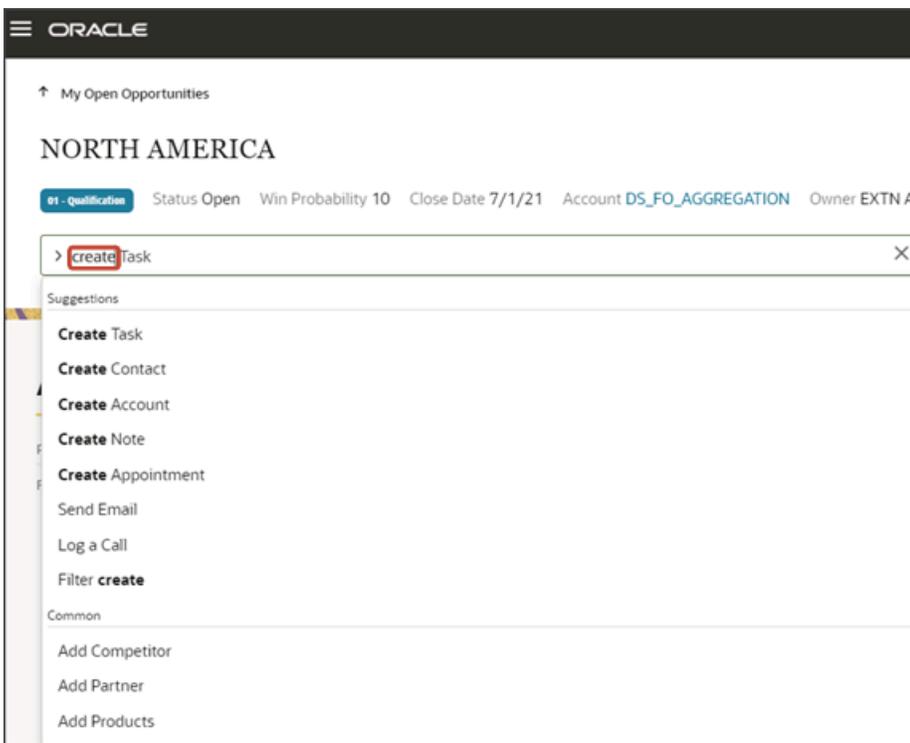
The Action Bar is a field at the top of an individual record where users can type keywords to access and update information in the record, and take actions. The list of actions that display in the Action Bar is contextual, so only actions that apply to the current page are displayed.

For example, users can enter **create**, **update**, or **add** and the Action Bar will automatically populate with a list of possible actions.

Here's a screenshot of the Action Bar:

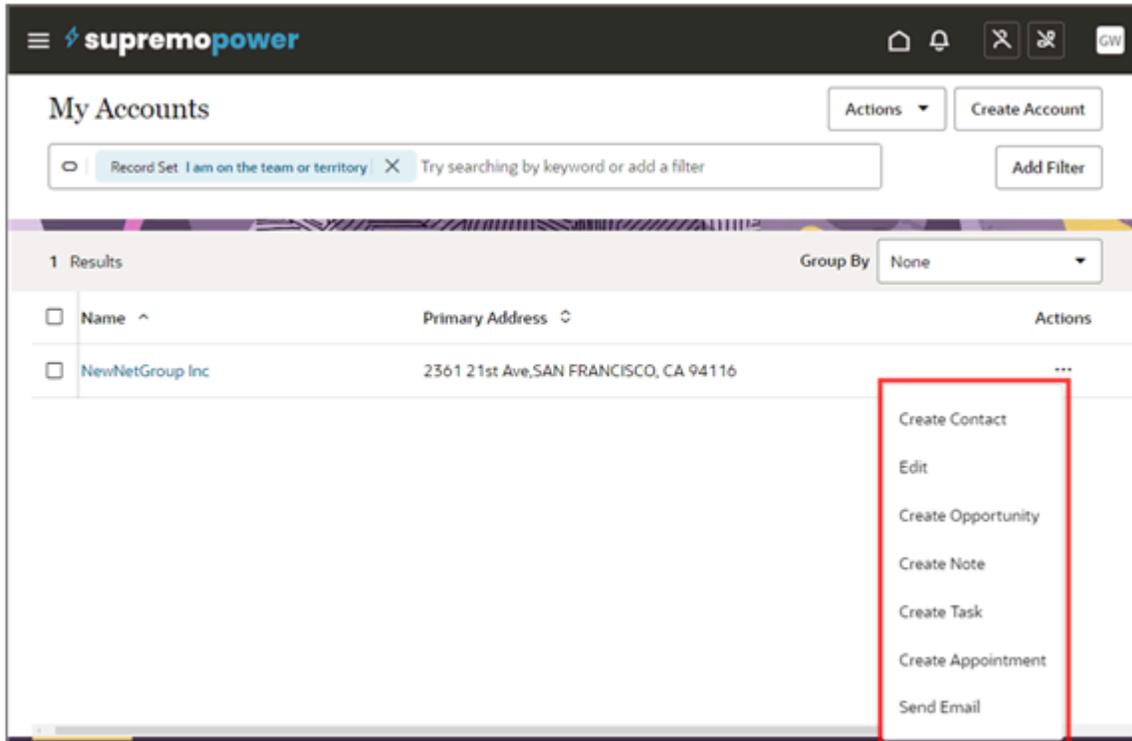


Here's a screenshot of the Action Bar with suggested actions, after typing **create**:



Smart Actions in the Redwood User Experience List Pages

On the Oracle Sales in the Redwood User Experience list pages, smart actions appear in the list of actions for individual records. The Actions list is highlighted in the following screenshot.



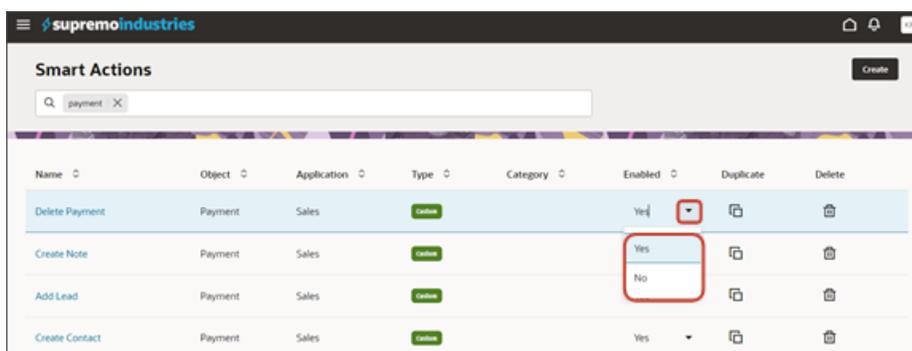
List pages don't permit users to enter smart actions in the search bar. The Create action in the search bar and the Create button aren't smart actions.

If you want to prevent a particular job role from creating a record, you must remove the create permission from their job role. You can't prevent users from creating records by restricting smart actions to specific job roles.

Enable or Disable Actions

To disable an action so that users won't see it:

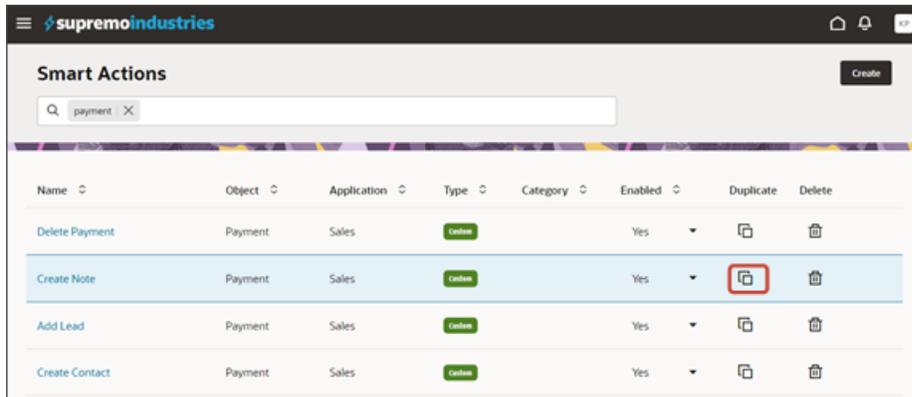
1. In Application Composer, click **Smart Actions**, available under Common Setup.
2. Find the action you want to disable and, in the Enabled column, select **No**.



Duplicate Actions

To duplicate an existing action:

1. On the Smart Actions page, navigate to the smart action that you want to duplicate and click the **Duplicate** icon.



2. On the Edit Action page, edit the name and other details, if required, and click **Save**.

Application Composer saves the action with a copy of the existing name.

Delete Actions

You can delete custom smart actions, but not system smart actions that are delivered with the application.

Additionally, you can't delete smart actions that are used by Routines or Sales Orchestration.

Related Topics

- [Create REST-Based Smart Actions](#)
- [Create Object Function-Based Smart Actions](#)
- [Create UI-Based Smart Actions](#)

Create REST-Based Smart Actions

You can create custom smart actions that directly call a service, without any UI.

To create a REST-based smart action:

1. Enter into a sandbox that's enabled for Application Composer.
2. In Application Composer, click **Smart Actions**, available under Common Setup.
3. On the Smart Actions page, click **Create**.
Application Composer displays the Create Smart Action guided process in a new browser tab.
4. On the Kind of Action page:
 - a. Click **REST-based action**.
 - b. Click **Continue**.
5. On the Basic Details page:
 - a. In the **Name** field, enter a display name for the action.

- b. In the **Object** field, select the object for which you're creating the action.
- c. In the **Action ID** field, accept the automatically generated value or enter a unique value.

This value must be unique across all smart actions.

- d. If you want REST-based smart actions to update multiple records at once, then select the **Mass Action** switch.
- e. Click **Continue**.

6. On the Availability page:

- a. In the **Application** field, select the application where the action will display. Available values include **Help Desk, Routines, Sales, Service Center, and Workspace**.
- b. In the **UI Availability** field, select the UI where the action will display in addition to the Action Bar. For example, select **List page**.

Note: For the smart action to be available in Orchestration automation steps, in the **Application** field, select **Sales** and in the **UI Availability** field, select **Sales Orchestration**.

- c. In the **Role Filter** field, optionally select the custom roles that can view the action.

If you don't select a role, then the action will be available to all roles.

- d. In the **Condition** region, optionally set conditions for the action to appear, such as only if a certain field's value is set to a particular value. For example, show a smart action only if a lead's status is **Hot**.
 - Define one or more conditions in a group.
 - You can define one or more groups of conditions.
 - Click **+ Add** to add a condition.

In the **Attribute** field, start typing an attribute name and select from the available list. The list of attributes in this field includes all REST-enabled attributes.

Enable attributes in your Adaptive Search setup to ensure this smart action condition works on list pages and in Workspace.

If you don't specify a condition, then the action will always be available based on other parameters that you specify, like role and application.

- e. If you want to restrict the smart action to specific parent objects, then enter the object names in the **Context** region, **Parent Object** field.
- f. Click **Continue**.

7. On the Action Details page, enter the REST request details:

- a. In the **Path** and **Method** fields, select the REST request details.

If the path includes `action`, then these two additional fields display to indicate if the REST API response is a URL and whether to open the URL in either a new browser tab or the same browser tab.

- **Object function returns a URL**
- **Browser Window Target**

- b. In the **Path Parameters** and **Request Body Parameters** regions, specify parameter details as needed.
- c. Click **Continue**.

8. On the Confirmation Message page:
 - a. Enter primary and secondary messages as well as continue and cancel button labels that display to users in a confirmation dialog before the action is executed.
 - b. You can also enter an optional confirmation message for display after the action is executed. Use the Add tokens region to include a token in your message.
 - c. Click **Continue**.
9. On the Review and Submit page, review the action's configuration and click **Submit** when ready.

Create Object Function-Based Smart Actions

You can create custom smart actions that directly call an object function, without any UI.

To create an object function-based smart action:

1. Enter into a sandbox that's enabled for Application Composer.
2. In Application Composer, click **Smart Actions**, available under Common Setup.
3. On the Smart Actions page, click **Create**.
Application Composer displays the Create Smart Action guided process in a new browser tab.
4. On the Kind of Action page:
 - a. Click **Object function-based action**.
 - b. Click **Continue**.
5. On the Basic Details page:
 - a. In the **Name** field, enter a display name for the action.
 - b. In the **Object** field, select the object for which you're creating the action.
 - c. In the **Action ID** field, accept the automatically generated value or enter a unique value. This value must be unique across all smart actions.
 - d. Click **Continue**.
6. On the Availability page:
 - a. In the **Application** field, select the application where the action will display. Available values include **Help Desk**, **Routines**, **Sales**, **Service Center**, and **Workspace**.
 - b. In the **UI Availability** field, select the UI where the action will display in addition to the Action Bar. For example, select **List page**.
Note: For the smart action to be available in Orchestration automation steps, in the **Application** field, select **Sales** and in the **UI Availability** field, select **Sales Orchestration**.
 - c. In the **Role Filter** field, optionally select the custom roles that can view the action. If you don't select a role, then the action will be available to all roles.
 - d. In the **Condition** region, optionally set conditions for the action to appear, such as only if a certain field's value is set to a particular value. For example, show a smart action only if a lead's status is **Hot**.
 - Define one or more conditions in a group.
 - You can define one or more groups of conditions.
 - Click **+ Add** to add a condition.

In the **Attribute** field, start typing an attribute name and select from the available list. The list of attributes in this field includes all REST-enabled attributes.

Enable attributes in your Adaptive Search setup to ensure this smart action condition works on list pages and in Workspace.

If you don't specify a condition, then the action will always be available based on other parameters that you specify, like role and application.

- e. If you want to restrict the smart action to specific parent objects, then enter the object names in the **Context** region, **Parent Object** field.
 - f. Click **Continue**.
7. On the Action Details page, enter the object function metadata:
 - a. In the **Object function action** field, choose a function available for the selected object.
 - b. Two additional fields display to indicate if the object function response is a URL and whether to open the URL in either a new browser tab or the same browser tab:
 - **Object function returns a URL**
 - **Browser Window Target**
 - c. In the **Request Body Parameters** region, specify parameter details as needed.
 - d. Click **Continue**.
 8. On the Confirmation Message page:
 - a. Enter primary and secondary messages as well as continue and cancel button labels that display to users in a confirmation dialog before the action is executed.
 - b. You can also enter an optional confirmation message for display after the action is executed. Use the Add tokens region to include a token in your message.
 - c. Click **Continue**.
 9. On the Review and Submit page, review the action's configuration and click **Submit** when ready.

Create UI-Based Smart Actions

For Oracle Sales in the Redwood User Experience, Service Center, and Help Desk, you can create custom smart actions that navigate users to a new UI location.

Note: When you extend your application using the CX Extension Generator, the standard smart actions are created for you automatically. See [Create a New Application Using the CX Extension Generator](#).

To create a UI-based smart action:

1. Enter into a sandbox that's enabled for Application Composer.
2. In Application Composer, click **Smart Actions**, available under Common Setup.
3. On the Smart Actions page, click **Create**.
Application Composer displays the Create Smart Action guided process in a new browser tab.
4. On the Kind of Action page:
 - a. Click **UI-based action**.
 - b. Click **Continue**.

5. On the Basic Details page:

- a. In the **Name** field, enter a display name for the action.
- b. In the **Object** field, select the object for which you're creating the action.
- c. In the **Action ID** field, accept the automatically generated value or enter a unique value.

This value must be unique across all smart actions.

- d. Click **Continue**.

6. On the Availability page:

- a. In the **Application** field, select the application where the action will display, such as **Sales** or **Service**.
- b. In the **UI Availability** field, select the UI where the action will display in addition to the Action Bar. For example, selecting **List page** includes the smart actions in the list of actions available from the Actions menu for each record listed on the page. For details see *Smart Actions in the Redwood User Experience List Pages*

Note: For the smart action to be available in Orchestration automation steps, in the **Application** field, select **Sales** and in the **UI Availability** field, select **Sales Orchestration**.

- c. In the **Role Filter** field, optionally select the custom roles that can view the action.

If you don't select a role, then the action will be available to all roles.

- d. In the **Condition** region, optionally set conditions for the action to appear, such as only if a certain field's value is set to a particular value. For example, show a smart action only if a lead's status is **Hot**.
 - Define one or more conditions in a group.
 - You can define one or more groups of conditions.
 - Click **+ Add** to add a condition.

In the **Attribute** field, start typing an attribute name and select from the available list. The list of attributes in this field includes all REST-enabled attributes.

Enable attributes in your Adaptive Search setup to ensure this smart action condition works on list pages and in Workspace.

If you don't specify a condition, then the action will always be available based on other parameters that you specify, like role and application.

- e. If you want to restrict the smart action to specific parent objects, then enter the object names in the **Context** region, **Parent Object** field.
- f. Click **Continue**.

7. On the Action Type page, provide action metadata for the type of action you're creating.

This step is required for smart actions created for custom objects. If the ORA-REDWOOD-SALES-UI feature is enabled, then this step is also required for some standard objects.

- a. In the **Type** field, specify the type of action, such as **Add**, **Create**, and so on.

Note: Any **Show** actions that you create are used only by Sales Orchestration flows, and aren't displayed in the Action Bar, list pages, or CX Sales Mobile.

- b. In the **Target Object** field, select the object that you're creating the action for.

For example, if you're creating an **Add Contact** smart action on the Account object, then the Type is **Add** and the Target Object is **Contact**.

- c. Use the **Field Mapping** region to map one or more values from a parent object record to a child or related object record.

For example, when users use a smart action to create a related object record, you might want the create related object record page to be prepopulated with additional values from the parent record.

- d. Click **Continue**.

8. On the Action Details page:

- a. In the Navigation Target field, select **Local**.

- b. Click **Continue**.

9. On the Confirmation Message page:

- a. In the Confirmation Message After Action Execution field, optionally enter a message to display to users.

- b. Use the Add tokens region to include a token in your message.

- c. Click **Continue**.

10. On the Review and Submit page, review the action's configuration and click **Submit** when ready.

Direct Page Links for CX Sales

Direct page links are links that point to a specific page in Oracle CX Sales. In any e-mail, report, or user interface page, you can add a link that opens an account, contact, household, opportunity, lead, activity, or top-level custom object record.

This topic covers:

- List of objects that support direct page linking to application pages.
- URL pattern to use for direct page linking.

The URL pattern is translated by the Direct Page Link servlet in the middle tier which reads the incoming request parameters, generates a new URL, and redirects the request to the target page.

- Where can you use these links?
- How user authentication provides secured access to the target page.

CX Sales Objects That Support Direct Page Linking

You can link to application pages for the following objects.

- Account
- Activity
- Analytics
- Application UI dashboard
- Asset
- Claim
- Contact
- Custom object
- Deal registration
- Fund request
- Household
- Lead
- MDF budget
- Opportunity
- Partner
- Partner Contact
- Partner Program
- Product
- Program Enrollment
- Work Order

Note: You can also link directly to the application UI dashboard and to the Analytics landing page.

URL Pattern to Use for Direct Page Links to Application Pages

The direct page link URL uses a simple pattern which points to a default tab at the top level of an object's application set of pages. Some objects support linking directly to a subtab. When linking to a page, the link opens either a specific tab that you specify in the URL or a default tab if you have previously specified one in the page layout. The default tab specified in the page layout takes precedence over any tab included in the direct page link itself.

Depending on the object you're linking to, use the following syntax to create a direct page link:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?  
fndGlobalItemId=<CARDCODE>&fndTaskItemId=<TABCODE>&fnd=%3B<TaskFlowParamName1>%253D<TaskFlowParamValue1>  
%253B<TaskFlowParamName2>%253D<TaskFlowParamValue2>%253B<TaskFlowParamName3>%253D<TaskFlowParamValue3>%253B%3B  
%3B%3Bfalse%3B256%3B%3B%3B
```

Tip: Copy the first part of the URL, `https://<hostname>:<port>/<application>/faces/FuseOverview?`, from the customer instance's actual home page. The `taskFlowParamName` and `taskFlowParamValue` pairs are optional and are typically used for record identifiers. In some cases, an additional parameter, `subTabName`, is also included in the URL. Where supported, this additional parameter allows direct links to the specified subtab.

Tip: Direct page links created in a release prior to Release 10 used a different link pattern. However, those direct page links created prior to Release 10 will continue to work in Release 10 and later.

Use the patterns specified as follows to link to the application pages for these objects:

- Account

For this object, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?  
fndGlobalItemId=ZCM_CUSTOMERCTRINFRA360_CUSTOMERS_CRM_CARD&fndTaskItemId=ZCM_CUSTOMERCTRINFRA360_CUSTOMERS_CRM_CARD  
&subTabName%253DOverview%253BAccountPartyId%253D<recordID, such as 123456>%253B%3B%3Bfalse%3B256%3B%3B%3B
```

This direct page link opens the Overview subtab on the Edit Account application page.

To link to other Edit Account subtabs, change the value for the parameter `subTabName` to one of the following values for each subtab, such as `&fnd=subTabName%253DProfile`.

- Overview
- Profile
- SalesAccountTeam
- Contacts
- Assets
- Opportunities
- Quotes
- Leads
- Relationships
- Recommendations
- Notes
- Activities
- Conversations

- Activity

For this object, use this direct page link URL pattern to open a list of all activities:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?  
fndGlobalItemId=ZMM_ACTIVITIES_CRM_CARD&fndTaskItemId=ZMM_ACTIVITIES_ACTIVITIES_CRM_CARD%253B%3B%3B%3Bfalse%3B256%3B%3B%3B
```

To link to a specific activity (appointment or task), append the record's primary key as follows: `https://<hostname>:<port>/<application>/faces/FuseOverview?`

```
fndGlobalItemId=ZMM_ACTIVITIES_CRM_CARD&fndTaskItemId=ZMM_ACTIVITIES_ACTIVITIES_CRM&fnd=
%3BFunctionCode%253DTASK%253BActivityId%253D<recordID,such as 123456>%253B%3B%3B%3Bfalse%3B256%3B%3B%3B .
```

You can also link directly to specific types of activities by changing the value for the parameter `&fndTaskItemId` to one of the following values:

- Link to a list of all tasks:

Use pattern: `&fndTaskItemId=ZMM_ACTIVITIES_TASKS_CRM`

- Link to appointments (calendar view):

Use pattern: `&fndTaskItemId=ZMM_ACTIVITIES_APPOINTMENTS_CRM`

- Analytics

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?
fndGlobalItemId=CRM_FUSE_ANALYTICS_CARD&fndTaskItemId=CRM_ANALYTICS%253D123456%253B%3B%3B
%3Bfalse%3B256%3B%3B%3B
```

- Application UI dashboard

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?
fndGlobalItemId=CRM_FUSE_DASHBOARD_CARD&fndTaskItemId=CRM_DASHBOARD%253D123456%253B%3B%3B
%3Bfalse%3B256%3B%3B%3B
```

- Assets

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?
fndGlobalItemId=itemNode_Sales_Assets&fndTaskItemId=ZCM_CUSTOMERCTRINFRA360_ASSETS_CRM&fnd=
%3BAssetId%253D<recordID,such as 123456>%253B%3B%3B%3Bfalse%3B256%3BCRM%3BASSET%3BAsset%2BName
```

- Claim

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?fnd=%3BClaimId
%253D300100089921747%253BsubTabName%253DSUMMARY%3B%3B%3Bfalse%3B256%3B%3B
%3B&fndGlobalItemId=itemNode_partner_management_mdf&fndTaskItemId=MKT_MDF_CLAIMS_CRM&_afrLoop=1700807529
```

- Contact

For this object, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?
fndGlobalItemId=HZ_FOUNDATIONPARTIES_CONTACTS_CRM_CARD&fndTaskItemId=HZ_FOUNDATIONPARTIES_CONTACTS_CRM&f
```

```
%3BsubTabName%253DOverview%253BContactPartyId%253D<recordID, such as 123456>%253B%3B%3Bfalse%3B256%3B%3B%3B
```

This direct page link opens the Overview subtab on the Edit Contact application page.

To link to other Edit Contact subtabs, change the value for the parameter `subTabName` to one of the following values for each subtab, such as `&fnd=%3BsubTabName%253DProfile`.

- o Overview
- o Profile
- o Team
- o Opportunities
- o Leads
- o Assets
- o Relationships
- o Recommendations
- o Notes
- o Activities
- o Conversations
- Custom object
 - o To create a direct link to the default summary tab for a custom object, use this direct page link URL pattern:


```
https://<hostname>:<port>/<application>/faces/FuseOverview?
fndTaskItemId=CRM_CUSTOM_TAB_<XXXX>&fndGlobalItemId=CRM_CUSTOM_CARD_<XXXX>&fnd=
%3BsubTabName%253DSUMMARY%253BObjectId%253D<YYYY>%253B%3B%3Bfalse%3B256%3B%3B%3B
```

 - Copy the first part of the URL, `https://<hostname>:<port>/<application>/faces/FuseOverview?`, from the customer instance's actual home page.
 - Replace `xxxx` with the custom object's API name using all upper case letters including `_C`. For example, `TROUBLETICKET_C`. Obtain the API name from the object's overview page (click the object's node in the Custom Objects tree in Application Composer).
 - Replace `yyyy` with the custom object's primary key in the database.
 - o To create a direct link to a different subtab, replace `SUMMARY` with the subtab's Component ID. This is an automatically generated ID which you can find by viewing the details page layout where the custom subtab exists.
- Deal registration

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?fnd=%3BDealId
%253D300100072550501%253BsubTabName%253DSUMMARY%3B%3B%3Bfalse%3B256%3B%3B
%3B&fndGlobalItemId=MKL_DEALREGS_CARD&fndTaskItemId=DEALREGS
```

- Fund request

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?fnd=%3BFundRequestId%253D300100089921736%253BsubTabName%253DSUMMARY%3B%3B%3Bfalse%3B256%3B%3B%3B&fndGlobalItemId=itemNode_partner_management_mdf&fndTaskItemId=MKT_MDF_FUNDS_CRM&_afrLoop=16917244458
```

- Household

For this object, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?fndGlobalItemId=ZCM_CUSTOMERCTRINFRA360_GROUPS_CRM_CARD&fndTaskItemId=ZCM_CUSTOMERCTRINFRA360_GROUPS_CRM%3BsubTabName%253DOverview%253BHouseholdPartyId%253D<recordID, such as 123456>%253B%3B%3B%3Bfalse%3B256%3B%3B%3B
```

This direct page link opens the Overview subtab on the Edit Household application page.

To link to other Edit Household subtabs, change the value for the parameter `subTabName` to one of the following values for each subtab, such as `&fnd=%3BsubTabName%253DProfile`.

- Overview
- Profile
- SalesAccountTeam
- Contacts
- Opportunities
- Assets
- Leads
- Relationships
- Notes
- Activities
- Conversations

- Lead

For this object, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?  
fndGlobalItemId=MKL_LEADS_CARD&fndTaskItemId=MKL_LEADS&fnd=%3BsubTabName%253DSUMMARY%253BLeadId  
%253D<recordID,such as recordID, such as 123456>%253B%3B%3B%3Bfalse%3B256%3B%3B%3B
```

This direct page link opens the Summary subtab on the Edit Lead application page.

To link to other Edit Lead subtabs, change the value for the parameter `subTabName` to one of the following values for each subtab, such as `&fnd=%3BsubTabName%253DNOTES`.

- SUMMARY
- CONTACTS
- QUALIFICATIONS
- SALESTEAM
- ACTIVITIES
- RESPONSES
- NOTES
- OPPORTUNITIES
- CONVERSATIONS

You can also link directly to Analytics tabs off the Opportunities landing page, by replacing the `fndTaskItemId` value with these values:

- MKL_LEADS_ANALYTICS1_CRM
- MKL_LEADS_ANALYTICS2_CRM
- MKL_LEADS_ANALYTICS3_CRM

- MDF budget

For this object, use this direct page link URL pattern:

```
https://<hostname>/sales/faces/CrmFusionHome?  
cardToOpen=itemNode_partner_management_mdf&tabToOpen=MKT_MDF_BUDGETS_CRM&TF_subTabName=SUMMARY&TF_BudgetId=xxxxx
```

Use the below script to add an ID to the deep link:

```
https://<hostname>/sales/faces/CrmFusionHome?  
cardToOpen=itemNode_partner_management_mdf&tabToOpen=MKT_MDF_BUDGETS_CRM&TF_subTabName=SUMMARY&TF_BudgetId=  
${API_ID_FIELD}"
```

- Replace `{API_ID_FIELD}` with the MDF budget ID.

- Opportunity

For this object, use this direct page link URL pattern:

```
https://<hostname>/crmUI/faces/FuseOverview?  
fndTaskItemNodeId=MOO_OPPTYMGMTOPPORTUNITIES_CRM&fndGlobalItemNodeId=MOO_OPPTYMGMTOPPORTUNITIES_CRM_CARD&fnd=  
%3BsubTabName%253DSummary%253BSkipToEditOpptyId%253D300100186091209%253B%3B%3Bfalse%3B256%3B%3B%3B
```

This direct page link opens the Summary subtab on the Edit Opportunity application page.

To link to other Edit Opportunity subtabs, change the value for the parameter `subTabName` to one of the following values for each subtab, such as `&fnd=%3BsubTabName%253DNotes`.

- Summary
- Quotes
- Contact
- OpptyTeam
- OpptyPartner
- Activities
- Notes
- Assessments
- Leads
- Conversations

You can also link directly to Analytics tabs off the Opportunities landing page, by replacing the `fndTaskItemNodeId` value with these values:

- `MKL_LEADS_ANALYTICS1_CRM`
- `MKL_LEADS_ANALYTICS2_CRM`
- `MKL_LEADS_ANALYTICS3_CRM`

- Partner

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?  
ZPM_PARTNERS_CARD&findTaskItemNodeId=ZPM_PARTNERS&find=%3BpartyId%253D10000151014782%253B%3B%3Bfalse  
%3B256%3B%3B%3B
```

- Partner Contact

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?  
fndGlobalItemNodeId=ZPM_PARTNERS_CARD&fndTaskItemNodeID=ZPM_PARTNERS&fnd=%3B%3BpartyId  
%253D[$PartnerPartyId$]%253B%253BcontactPartyId%253D[$ContactPartyId$]%253B;;;false;256;;;
```

- Replace `[$PartnerPartyId$]` and `[$ContactPartyId$]` with the partner's party ID number.

- Partner Program

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?
fnd=;programIdFromRecentItems=300100017959761&subTabName=SUMMARY&fndGlobalItemId=ZPM_PROGRAMS_CARD&fndTaskIt
```

- Product

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?fnd=%3BskipToEditItemNumber%253D<PUT
ITEM NUMBER HERE LIKE 12345>%253BskipToEditInventoryItemId%253D<PUT ITEM INVENTORY ID HERE>
%253B%3B%3Bfalse%3B256%3BCRM%3BPRODUCT%3BProduct&pageParams=nullnavigateFromFavRI%3Dtrue
%3B&nullnavigateFromFavRI=true&fndGlobalItemId=QSC_PRODUCT_CRM_CARD&fndTaskItemId=QSC_PRODUCT_CRM_CARD
```

- Program Enrollment

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?
fnd=;enrollmentIdFromRecentItems=300100185785292&subTabName=SUMMARY&fndGlobalItemId=ZPM_ENROLLMENTS_CARD&fnd
```

- Work Order

For this page, use this direct page link URL pattern:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?fnd=%252FWEB-INF%252Foracle%252Fapps
%252FcrmCommon%252Fcore%252FpublicUi%252Fflow%252FNestedDynamicTab%252FNestedDynamicTabContainerTF.xml
%2523NestedDynamicTabContainerTF%3BobjectId%253D300100543227013%253BtabKey%253DWorkOrder
%252FFieldSvcFuseWorkOrderDrillDownTF%253BparamMapString%253DwoId%253D300100543227013%253BwoNumber
%253D0000015017%253BdisplayMode%253DTopLevelTab%253B%3BwoId%253D300100543227013%3B0000015017%3Bfalse
%3B256%3BCRM%3BWorkOrder%3B&fndGlobalItemId=itemNode_Service_WorkOrders1
```

- Replace `woId` with the work order ID and `woNumber` with the work order number.

Use the following URL pattern with parameters:

```
https://<hostname>:<port>/<application>/faces/FuseOverview?fnd=%252FWEB-INF%252Foracle%252Fapps
%252FcrmCommon%252Fcore%252FpublicUi%252Fflow%252FNestedDynamicTab%252FNestedDynamicTabContainerTF.xml
%2523NestedDynamicTabContainerTF%3BobjectId%253D@{1}%253BtabKey%253DWorkOrder
%252FFieldSvcFuseWorkOrderDrillDownTF%253BparamMapString%253DwoId%253D@{1}%253BwoNumber
%253D@{2}%253BdisplayMode%253DTopLevelTab%253B%3BwoId%253D@{1}%3B@{2}%3Bfalse%3B256%3BCRM%3BWorkOrder
%3B&fndGlobalItemId=itemNode_Service_WorkOrders1
```

- Replace `woId` with the column variable for the work order ID and replace `wo_number` with the column variable for the work order number.

Assigning These Links

First, you should know which page you want to link to, and where you want that link to appear. You can add direct page links to:

- Reports

Use BI Composer or Analytics Answers to add direct page links to your reports.

- User interface pages
Create a direct page link using an object's Actions and Links node, then add the link to the object's user interface pages.
- External third-party applications
Create a direct page link to link directly to a page in your application.

User Authentication for Secured Access

The direct page link servlet requires authentication. If you haven't been previously authenticated, you must sign in to gain access to the target page. After signing in, you're redirected to the target page. If you have already been authenticated at the time of clicking the direct page link, the target page is immediately displayed (without asking you to sign in).

Related Topics

- [Actions and Links](#)

Deep Links for Service Requests

Deep links are links that point to another page. You can link to a page where users can create a new service request, or edit an existing service request. Read this topic to learn how to construct these links.

Once you have a link, you can then send it to someone else, or you can manually add the link to any page, such as to a report or user interface page.

This topic covers:

- Deep linking methods
- Get Link method
- Manual URL method
- Where can you use these links?
- How user authentication provides secured access to the target page.

Deep Linking Methods

To create deep links to service request pages, you can use one of two methods.

- Get Link method
Use this method to easily "get a link" to the details page (edit page) of any service request.

- Manual URL method

Use this method to link to any service request page. Link to where users can create a new CRM or HR Help Desk service request, or link to the details page of an existing service request.

This method is useful because you have the flexibility to specify parameters in the URL itself, so that you can link to exactly the page you want. For example, link to a page where the user can create a service request for a particular account and product group.

Get Link Method

The Get Link method is the simplest method to obtain a deep link, but it works only for service request details pages (edit pages). To obtain an automatically generated deep link:

1. Navigate to the details page (edit page) of the service request that you want to link to.
2. Click **Actions > Get Link**.
3. Copy the link by clicking Control + C.
4. Share that link.

Manual URL Method

You must use the manual URL method to link to a service request create page, or to link to a specific subtab other than the summary tab on a details page. To link to a details page with the summary subtab in focus, you can use this method but the Get Link method is simpler.

To use the manual URL method, choose from four URL patterns to build your deep link. In each pattern, you can specify a variety of parameter values to satisfy the particular needs of your "create service request" business case. Depending on the link that you build, your users can navigate to the create page for the service request, or to a specific record.

To link to a page where your users can create new CRM service requests, use the following syntax:

- `https://<hostname>:<port>/fscmUI/faces/deeplink?objType=SVC_SERVICE_REQUEST&action=CREATE_IN_TAB`

To link to a page where your users can create new HCM service requests, use the following syntax:

- `https://<hostname>:<port>/fscmUI/faces/deeplink?objType=SVC_SERVICE_REQUEST_HCM&action=CREATE_IN_TAB`

To link to an existing CRM service request (in this example, service request SR0000033095), use the following syntax:

- `https://<hostname>:<port>/fscmUI/faces/deeplink?objType=SVC_SERVICE_REQUEST&objKey=srNumber%3DSR0000033095&action=EDIT_IN_TAB`

To link to an existing HCM service request (in this example, service request SR0000033095), use the following syntax:

- `https://<hostname>:<port>/fscmUI/faces/deeplink?objType=SVC_SERVICE_REQUEST_HCM&objKey=srNumber%3DSR0000033095&action=EDIT_IN_TAB`

You can link to specific subtabs (in this example, the Messages subtab) using this syntax:

- `https://<hostname>:<port>/fscmUI/faces/deeplink?objType=SVC_SERVICE_REQUEST&objKey=srNumber%3DSR0000033095%3BselectedSubTabName%3DMessages&action=EDIT_IN_TAB`
- `https://<hostname>:<port>/fscmUI/faces/deeplink?objType=SVC_SERVICE_REQUEST_HCM&objKey=srNumber%3DSR0000033095%3BselectedSubTabName%3DMessages&action=EDIT_IN_TAB`

Tip: Copy the first part of the URL, `https://<hostname>:<port>/`, from your instance's actual home page.

Note: The deep links don't work if they contain the # character. Also, special characters such as @, ^, and * can be used only if preceded with the backslash character \. For example, you must use * instead of * in your regular expression. For more information about using special characters in regular expressions, see the open source documentation for regular expressions. After you modify the regular expression, remember to use a validator to ensure that it's still working.

This table lists of the components of the service request deep link pattern.

Parameter	Value	Description
objType	SVC_SERVICE_REQUEST SVC_SERVICE_REQUEST_HCM	Use SVC_SERVICE_REQUEST to link to CRM service requests. Use SVC_SERVICE_REQUEST_HCM to link to HCM service requests.
objKey	Here's a list of the available fields (attributes) for the objKey parameter which you might typically want to use. When linking to the create page, you can pass any and all of the fields listed below. Additionally, you can use all fields, including custom fields, that are exposed for the service request object. If using custom fields, be sure to use the custom field's API name, not display name, in the deep link URL. When linking to the edit page, pass only the srNumber parameter. <ul style="list-style-type: none"> • srNumber • srID • accountid (or accountPartyId) • primarycontactid (or primaryContactId) • productgroupid (or productGroupId) • productid (or inventoryItemId) • partneraccountid (or PartnerAccountPartyId) • assetid (or assetId) 	The objKey parameter is optional and is typically used for record identifiers. When linking to the create page, you can add multiple fields for the objKey parameter to prepopulate values onto the create page. For example, in a single URL, you could specify both a product ID as well as a primary contact ID to prepopulate onto the create service request page. The objKey takes key value pairs. Specify both the field and the field's value. When linking to the create page, you can specify more than one pair for the objKey parameter by using the & symbol. For example: objKey=accountPartyId%3D456&productGroupId%3D789
selectedSubTabName	<ul style="list-style-type: none"> • To link to the Messages subtab, use this value: Messages • To link to the Interaction History subtab, use this value: Interactions • To link to the Contacts subtab, use this value: Contacts 	The selectedSubTabName parameter is optional. Use this parameter to link directly to a subtab in an existing record.

Parameter	Value	Description
	<ul style="list-style-type: none"> To link to the Team subtab, use this value: Team To link to the Activities subtab, use this value: Activities To link to the Linked Articles subtab, use this value: LinkedArticles To link to the Milestone Details subtab, use this value: Milestones To link to the Part Details subtab, use this value: Parts To link to the Work Orders subtab, use this value: WorkOrders To link to the Audit History subtab, use this value: Audits To link to the Social subtab, use this value: Conversations To link to the Action Plan subtab, use this value: ActionPlan 	
action	CREATE_IN_TAB EDIT_IN_TAB	Open the Create Service Request page in dynamic tab mode. Edit an existing service request in dynamic tab mode.

Assigning These Links

You can add deep links to:

- Reports

Use BI Composer or Analytics Answers to add deep links to your reports.

- User interface pages

Navigate to the service request object's Actions and Links node in Application Composer, and create an action or link using the deep link that you want. Then, use Application Composer to add the action or link to the object's user interface pages.

Note: On the Approvals page, deep links are only supported for top-level custom objects and the following standard objects:

- Account
- Contact
- Household
- Lead
- Opportunity
- Partner
- Activity
- Business Plan
- Service request

- External third-party applications

Create a deep link to link directly to a service request page.

User Authentication for Secured Access

The deep link servlet requires authentication. If you haven't been previously authenticated, you must sign in to gain access to the target page. After signing in, you're redirected to the target page. If you have already been authenticated at the time of clicking the deep link, the target page is immediately displayed (without asking you to sign in).

Related Topics

- [Actions and Links](#)

How to View Application Composer Changes

Use the Configuration report to view a detailed list of Application Composer changes made by administrators in your environment. Access the report under Metadata Manager in the Common Setup pane in Application Composer.

You can generate the report as many times as needed, each time overwriting the previously generated version of the report.

Configuration Report

The Configuration report lists changes in your environment that were created in Application Composer, as described in the following list. You can download the report in HTML, in Microsoft Excel (.xls) format, or in Microsoft Excel Worksheet (.xlsx) format.

The report includes:

Report Item	Description
Configuration Summary	Provides a summary of changes such as the number of objects and fields, relationships created, and object validations.
Global functions	Includes a listing of the global functions created.
Relationships	Includes a listing of all the relationships created.
Object summary	<p>Select All modified objects to include all the changes that have been made to any objects.</p> <p>Select Selected objects to include all the changes for only a subset of objects.</p> <p>Tip: To include layout details, select up to five objects only.</p>
Fields	Includes custom fields and standard fields that have been modified for the objects that you selected.
Server scripts	Includes object validations, object functions, object triggers, and global functions for the objects that you selected.
Object workflows	Includes object workflows for the objects that you selected.
Page layout summary	<p>Includes a listing of dynamic page layout names and layout conditions for the objects that you selected.</p> <p>Select Layout details to include a list of all the actions, buttons, subtabs, and field names exposed in each subtab of a dynamic page layout.</p>
Custom object details	<p>Includes details of custom objects and fields that you have created.</p> <p>To view the details of a custom object, simply click its name in the report.</p> <p>You can retrieve details such as the Table Name, which indicates what table the custom object or field belongs to. You can also identify the fields or objects in that table using the Column Name attribute.</p>

To submit the report:

1. Confirm that you're not in a sandbox session.
2. In Application Composer, select **Metadata Manager** in the Common Setup pane.
3. Click **Generate**.
4. In the Generate Configuration Report dialog, indicate the items that you want to include in the Configuration report.
5. Click **Generate**.
6. Download the report by selecting either the HTML or Excel format.

The report doesn't include the following:

- Value map with the Parent Choice List set in the custom field "Choice List<Fixed>"
- Data Filter (Advanced Filter) set in the custom field "Choice List<Dynamic>"
- Security of custom objects
- Actions and links of custom objects
- Web services
- Copy map
- Email templates

How to View a Diagnostic Report of Your Application Composer Changes

The Configuration Analysis report provides a diagnostic analysis of all your custom application changes that were made in Application Composer. The report lists any non-recommended Groovy-related patterns that can impact the runtime scalability of your application. It also lists high-level corrective actions for the same.

You can download the report in an HTML format for a maximum of 5 objects.

To generate the report:

1. Make sure that you're in an active sandbox session.
2. In Application Composer, select **Metadata Manager** in the Common Setup pane.
3. In the Configuration Analyzer section, click **Generate Configuration Analysis Report**.
4. On the Generate Configuration Analysis Report dialog, select the objects for which you want to generate the diagnostics report.
5. Click **Generate Report**.

The following is a sample report:

Configuration Analysis Report

Created By : MHoopes
Creation Time : 2021-02-15 07:35:00
Sandbox Name : My_TEST_Sandbox

Following are some of the Groovy related Patterns which can impact the runtime scalability of the Product. Please review the same and also take the suggested corrective actions.
For more details you can refer: [Performance Best Practices for Extending Oracle CX Sales and B2B Service](#)
To find documentation and other learning resources, such as guides, whitepapers, and videos, visit the [Help Center for Oracle CX Sales and B2B Service](#).

- Use Case: Limit the Number of Triggers per Object

Use-Case Description : You might need to define object triggers to extend the standard processing logic, such as record creation, updates, and deletions. For best performance, you want to minimize the number of triggers.
Number Of Issues : 2
Corrective Action : When using triggers, follow these guidelines:

- Do not exceed 10 triggers per object.
- Minimize the number of triggers on an object by combining logically related inside a single trigger (A single trigger produces better performance than using multiple triggers)

Object Name	Number of Object Triggers	Triggers
Activity	11	PartnerRatUpdate, PopulateLastModifiedDate, Set_Theatre_CountryGroup, UpdatePartnerSpecializations, SetRollupFields, SetSalesMethodid, SetForecastCategory, SoldToValidate, OtherFieldsUpdate, RevRecDateUpdate, ChangeSoldToShipTo
Training_c	12	RemoveEvaluationLogOnExpiry, UpdateReporFields, SetContactNameContactEmail, cascadeInheritedSpecialization, UpdateReporField, FieldUpdateOnPartnerProfileOnStatusChange, UpdateGPPRFig, PartnerSpecializationOnExpiry, Populate_LinkidFields, SPPTrig, RegSPP, Acc

+ Use Case: Object Function Name having Special Characters
+ Use Case: Invoking Internal/OUTB Web Services from Triggers and Object Functions
+ Use Case: Performing Validations from Triggers instead of using Validation Rules
+ Use Case: Calling newView() function inside loops
+ Use Case: Calling Count() aggregate function instead of getEstimatedRowCount()
+ Use Case: Restricting number of Fields per Type per Object
+ Use Case: Complex Formula Fields

The following table lists the Groovy-related patterns that can impact the scalability of your application.

Pattern	Details
More than 10 object triggers per object	For better performance, minimize the number of triggers to 10 per object. The report lists the objects with more than 10 object triggers, along with the count and corresponding object trigger names.
Object functions with special characters in their names	Avoid object functions with special characters (like " , ' , (,) , * , / , \ , # , @ , + , or !) in their names. Some characters are already blocked by the application, but some aren't and may be allowed during object function creation or while renaming existing object function names. The report lists any such object functions with special characters.
Web services registered in Application Composer and invoked from triggers and object functions	The Web Services UI in Application Composer is used to support external web services for integration purposes. Don't register and use internally available web services. The report lists any web services that are internally deployed on the same host.
Validation exceptions thrown from triggers instead of directly using validation rules	Triggers are generally used for standard processing logic, such as record creation, updates, and deletions, or to assign programmatic default values to fields. Don't use triggers to perform validations. Instead, use field-level or object-level validation rules. The report lists all the field-level and object-level triggers that throw validation exceptions.
Complex formula fields	Use formula fields for simple computation logic only, since they're evaluated whenever referenced. If you use complex logic, it may have a performance impact due to additional evaluation overheads. For example, using executeQuery() in formula fields impacts performance and using certain APIs like setAttribute() can cause record lock issues. The report lists all the formula fields where you've used executeQuery() or setAttribute().

Pattern	Details
Number of field types per object	<p>The performance guidelines recommend that you limit the number of long text attributes for an object to 4 to optimize performance. It also recommends that you limit the number of dynamic choice list fields to 15.</p> <p>The report lists any object where the number of long text and dynamic choice list fields exceed their recommended limit of 4 and 15, respectively.</p>
newView() API inside a loop	<p>While implementing business logic, make sure that you keep the number of queries minimal. Avoid invoking the newView() function inside loops as this could cause extra queries.</p> <p>The report lists any occurrences where you've used the newView() API inside any loop.</p>
Long text fields on landing pages	<p>Restrict long text fields to Create and Detail pages, as appropriate.</p> <p>Using long text fields can impact performance. So, avoid adding long text fields to landing pages. Querying a long text field for multiple records on a summary page can impact performance.</p>
Number of custom subtabs per details page layout	<p>Limit the number of custom subtabs per details page layout for optimal performance. Don't exceed 10 custom subtabs per layout.</p> <p>Custom sub tabs and custom layouts are essential features that make a page dynamic. You create a custom layout when you want to display a different set of fields/tabs based on roles, record type fields, or conditions (implemented as Groovy expressions). However, you want to be sure to use custom sub tabs and custom layouts only when they're used to support a business use case. A custom layout puts a lot of information in metadata, so many layouts can degrade the performance of the runtime flow because the MDS loading time is higher.</p>
Actions and buttons with the setAttribute() exposed on pages	<p>Avoid using the setAttribute() API inside a Groovy function used for actions, buttons and links. Groovy expressions written for actions, buttons and links can execute multiple times during the page life cycle. So, make sure the Groovy logic is safe to run multiple times. Don't write any custom business logic to update object records, because repeated execution of such scripts can lead to errors.</p>
Number of object workflows per object	<p>Limit the number of active workflows per object to fewer than 10 and use a light-weight expression for the Groovy evaluation.</p> <p>Even though the execution of an object workflow is asynchronous, during the triggering action (Insert/Update), it evaluates conditions and publishes the events. As a result, complex Groovy logic or a large number of object workflows can impact the runtime performance of the application. Hence, it's important to restrict the number of active object workflows per object.</p>
Object workflows with a field update action	<p>Avoid field update actions for optimal performance.</p> <p>You can instead incorporate the action inside a trigger and handle it as a synchronous action.</p> <p>Use field update object workflows only if you need an execution schedule or if the business use case demands an asynchronous execution of the field update.</p>
Validation rules with the setAttribute() API	<p>For optimal performance, don't implement any business logic other than validations inside an object or a field validation rule Groovy script.</p> <p>You should only use validation scripts for validation. For example, you shouldn't use setAttribute() values in a validation script.</p>
Number of layouts per object	<p>For optimal performance, follow these guidelines when working with custom layouts:</p> <ul style="list-style-type: none"> Minimize the number of custom layouts to 20 or less. Avoid creating layouts frequently and making them inactive later. Marking a layout as "Inactive" only hides it from the UI.

Pattern	Details
	<ul style="list-style-type: none"> Avoid the usage of Groovy expressions to differentiate layouts. Always, consider other options such as Roles and Record Type fields.
<p>Web service invocations from Before Insert and Before Update triggers</p>	<p>Try to implement business logic without relying on web services, if possible.</p> <p>Consider whether the same logic can be executed in an asynchronous way. If you can execute the Groovy logic asynchronously, then the parent transaction (object record creation) will not wait for the completion of this Groovy. Instead, the Groovy logic is executed in the background, without impacting the UI performance.</p> <p>However, if your business requirement absolutely requires it, note the following:</p> <ul style="list-style-type: none"> Ensure that the web service call is executed at the right time, ideally as late as possible in a commit life cycle to ensure the validations in the current transactions are passed. Otherwise, it may lead to data inconsistency, because the web service call will commit automatically. Also, consider the After Commit, After Delete, and After Change Posted to DB triggers for logic involving Web Services. Avoid using web services to manipulate a business object which is directly accessible from Groovy. A before update trigger to call an external web service to sync data is a time-consuming process. The entire commit has to wait for the web service response, impacting the total runtime performance. Instead of doing this with triggers, you can write an object workflow that calls a Groovy expression on the update of a record and syncs to the external system. This asynchronous method of syncing data with the external system reduces the UI overhead.
<p>Web service invocations inside a loop</p>	<p>Avoid the use of web service calls inside a loop. Consider if it's absolutely necessary and if it can be done asynchronously as an ESS job or as a straightforward Groovy update.</p>
<p>Object workflows calling <code>sendNotification()</code></p>	<p>Avoid using the <code>adf.util.sendNotification()</code> API from the context of an object workflow.</p>
<p>ExecuteQuery API inside a loop</p>	<p>Avoid using <code>executeQuery()</code> inside loops, wherever possible and consider if it's absolutely necessary. If yes, it should be done as per Groovy Scripting guidelines, by using bind variables to achieve better performance.</p>
<p>ExecuteQuery before <code>getEstimatedRowCount</code></p>	<p>Avoid <code>executeQuery()</code> if you're using the <code>getEstimatedRowCount()</code> API.</p> <p>If the requirement is to get the row Count, simply use the <code>getEstimatedRowCount()</code> API. It will return the row count from the database. You don't need a separate <code>executeQuery()</code> execution.</p>
<p>ExecuteQuery called from dynamic layout advanced expression</p>	<p>Avoid the usage of complex Groovy expressions with <code>executeQuery()</code> in dynamic layout advanced expressions.</p> <p>Always consider other options, such as Roles and Record Type fields.</p>
<p>ExecuteQuery called without any selected attributes</p>	<p>When performing a business object query, it's important to indicate which fields your code will access from the results. This includes fields your logic plans to update as well.</p> <p>By doing this proactively, your application gains two advantages:</p> <ul style="list-style-type: none"> You retrieve only the data you need from the database. You avoid an additional system-initiated query to "fault-in" missing data on first reference. <p>This can be achieved using the <code>selectAttributesBeforeQuery</code> API. Ensuring this will improve the performance of the Groovy scripts.</p> <p>For more details, see "Explicitly Select Only the Attributes You Need" of the Groovy Scripting guide.</p>

Pattern	Details
Number of searchable fields per object	<p>Keep the number of searchable fields restricted to 100 or less.</p> <p>You can reduce the performance impact on a list page by reducing the number of searchable fields. The recommended practice is to determine which fields your user will most often need to perform searches for the object. Then, when you create a new custom field, clear the Searchable option, unless it's a necessary search field. This will reduce the performance impact of needing to populate the searchable attributes list whenever Advanced Search is used.</p>
Field updatable expressions calling the <code>setAttribute</code> API	<p>Follow the below guidelines when using field updatable expressions:</p> <ul style="list-style-type: none"> • Don't set any object attribute values inside an updatable Groovy expression. • Avoid using object functions in an updatable Groovy expression. • If the use case requirement is that an attribute's required or updatable property changes at runtime, then consider using simple expressions in the condition. • If the use case requirement is to make an attribute read only in a single page, you can use the layout level UI properties. • If the updatable property for an attribute can have single value on different functional flows, and this value is True or False, then you shouldn't use expressions. Instead, use the absolute values for the Required or Updatable property.
Global functions in updatable and required field level expressions	<p>Follow the below guidelines when using updatable or required field expressions:</p> <ul style="list-style-type: none"> • Avoid using global function invocations as much as possible in required or updatable Groovy expressions • If the use case requirement is that an attribute's required or updatable property changes at runtime, then consider using simple expressions in the condition.

Import and Export Custom Objects

Before you can import and export data for custom objects created with Application Composer, you must first generate the object artifacts required for Import and Export Management. To generate the required artifacts, you must first publish your active sandbox.

Enable Import and Export for Custom Objects

After completing your object model extensions, publish your sandbox. Sandbox publication registers your extensions and makes them available to Import and Export Management, so that you can import and export custom object data.

After you publish your sandbox, you can then schedule your import and export processes. For instructions and examples, see the chapter about importing and exporting custom objects in the import documentation for your cloud service.

Error Handling

After publishing a sandbox, always review the Import and Export page to confirm that the artifact generation successfully completed.

If errors occur, review the error details and diagnostics, update the relevant custom fields and objects, and then republish your sandbox to regenerate the artifacts.

If the status is unsuccessful after publishing the sandbox, then click the **Generate** button on the Import and Export page to try again:

1. Confirm that you're not in a sandbox.
2. On the main Overview page in Application Composer, select the **Import and Export** link in the Common Setup pane, or in the local area of the main Overview page.
3. On the Import and Export page, click the **Generate** button.

Related Topics

- [What are the available data import options?](#)
- [How do I enable import of custom objects?](#)
- [Overview of Bulk Data Export](#)

FAQs for Using Application Composer

What job role must I have to create my own objects in Application Composer?

Users with any one of the following three job roles can create custom objects and use all other Application Composer functions:

- Customer Relationship Management Application Administrator.
- Application Implementation Consultant.
- Master Data Management Application Administrator.

In Oracle CX Sales, provision the user with the Customer Relationship Management Application Administrator job role (for performing the configurations) and the Custom Objects Administration job role and Sales Administrator job role (for testing the configurations).

What's the difference between fixed choice lists and dynamic choice lists?

A fixed choice list and a dynamic choice list are similar in that the ultimate goal of both types of choice lists is to generate a field with a list of values.

For a fixed choice list, the field's specific list values are populated from a lookup type that you select when you define the field. The list displays in a single column and doesn't change.

A dynamic choice list, meanwhile, is populated from an existing object's actual data, which you can add filters to. Based on how you define the field, the list is dynamically populated at runtime and its values can change depending on the

user's context. In addition, you can add more columns to the dynamic choice list field to assist your users in making a selection at runtime.

What's the difference between Page Composer and Application Composer?

Page Composer is a web-based tool you can use to modify user interface (UI) pages and components for all products designated for use with Page Composer.

Page Composer uses two different modes of Design View. The first mode, Design View: Standard mode, is selected by default in all the pages when opening a page with Page Composer with the Design button selected. The second mode, Design View: Direct Selection mode, is activated when you click the Select tab for the UI page you want to modify. Direct Selection mode is available when you modify pages, but not when you personalize a dashboard page. With the Design View: Direct Selection mode, you can select and edit UI elements such as form fields and table columns. In Direct Selection mode, the UI components that you can select become apparent when you move your cursor over them. The UI components that you can select are highlighted and can be edited.

The following table describes how you can use each mode of Page Composer to modify dashboard pages and other select pages (such as the Partner Public Profile page, Partner Landing page, Partner Registration, Customer Snapshot, and Customer Overview - Analysis tab), and modify transactional pages (all other non-dashboard pages).

Use Cases	Design View - Standard mode	Design View - Direct Selection mode	Page Type
Add content (Business Intelligence reports, portlets such as Calendar)	Yes	No	Dashboard and other select pages
Delete region	Yes	No	Dashboard and other select pages
Move region	Yes	No	Dashboard and other select pages
Change page layout (for example, change a two column layout to three column layout)	Yes	No	Dashboard and other select pages
Default region state (open or close)	Yes	No	Transactional pages (all non-dashboard pages)
Manage save queries (create and edit)	Yes	No	Transactional pages (all non-dashboard pages)
Hide or show field	No	Yes	Transactional pages (all non-dashboard pages)
Change field label	No	Yes	Transactional pages (all non-dashboard pages)

Use Cases	Design View - Standard mode	Design View - Direct Selection mode	Page Type
Make field required or not	No	Yes	Transactional pages (all non-dashboard pages)
Make field read-only or updatable	No	Yes	Transactional pages (all non-dashboard pages)
Reorder fields in a Form	No	Yes	Transactional pages (all non-dashboard pages)
Reorder table columns	Yes	Yes	Transactional pages (all non-dashboard pages)
Hide or show table columns	Yes	Yes	Transactional pages (all non-dashboard pages)
Set table column width with the mouse	Yes	No	Transactional pages (all non-dashboard pages)
Set table column width and minimum width in percent or pixels	No	Yes	Transactional pages (all non-dashboard pages)
Sort column or not	No	Yes	Transactional pages (all non-dashboard pages)

Application Composer also lets you make UI changes at run time. However, the types of UI changes that you can make using Application Composer are quite different. Specifically, your primary focus when using Application Composer is to make actual object model changes. For example, you can create a new business object and related fields, and then create new application pages where that object and its fields are exposed to users.

The following table describes some of the primary differences between Page Composer and Application Composer. For example, using Application Composer, you can't access the Resource Catalog to add new content to a page.

Task	Available in Page Composer (site, job role, external or internal level)?	Available in Application Composer (site level only)?
Make object model extensions and expose your changes by creating or modifying work area pages	No	Yes
Reorder subtabs	No	Yes
Modify dashboard pages	Yes	No
Add content from the Resource Catalog	Yes	No

Task	Available in Page Composer (site, job role, external or internal level)?	Available in Application Composer (site level only)?
Simple field changes (show, hide, make read only, make required)	Yes (WYSIWYG - what you see is what you get)	Yes (non-WYSIWYG)
View results of changes immediately	Yes, in the Page Composer design interface	Yes, in the application that you're making changes to

Related Topics

- [About Application Composer](#)

What record sets is an owner permitted to search?

As an owner you can use the Record Set filter to search for different record sets that list all of the records that you and your subordinates own.

The following table lists and describes the record sets that you are permitted to search as an owner.

Record Set Name	Description
Records I own	Records you own. You become the record owner when you create it or if ownership is assigned to you.
My subordinates own	Records owned by you and your subordinates.
All records I can see	Records that you can view based on your position in the owner management chain and security permissions.

How can I hide record set lookup codes of custom objects?

To disable a custom record set lookup code of a custom object, do the following:

1. In the Setup and Maintenance work area, go to the following:
 - o Offering: Sales
 - o Functional Area: Sales Foundation
 - o Task: Manage Standard Lookups

The Manage Standard Lookups page opens.

2. In the **Lookup Type** field, enter **ORA_ZCX_SRCH_RECORD_SET**, which is the custom objects search record set filter.

3. In the Lookup Codes section, go to the lookup code that you want to disable and clear the check box in the **Enabled** column.

How can I view a custom object's table and column details?

Use the Configuration Report to view the details of a custom object. You can retrieve details such as the Table Name, which indicates what table the custom object or field belongs to.

You can also identify the fields or objects in that table using the Column Name attribute. Simply click the name of the custom object in the report.

For information on how to retrieve the configuration report, see the "How to View Application Composer Changes" topic in this chapter.

4 Extend Application Pages

Overview of Application Pages

Use Application Composer to modify application pages. You can make changes that some, or all, of your end users can see, depending on the conditions you set. You can create new pages for a custom object and modify existing pages for standard objects.

Custom Objects

You can create a set of application pages for all custom objects. Read this chapter to learn how.

Standard Objects

In addition, you can modify the application pages of many standard objects.

Objects whose application pages are extensible have a tab called Application Pages when the Pages node is selected.

1. In Application Composer, navigate to the object that you're interested in.
2. Click the Pages node.
3. Click the Application Pages tab.

You can duplicate and then modify the page layouts listed on this tab.

Examples of Page Modifications

Some of the page modifications that you can make using Application Composer include:

- Hide or show custom fields

Tip: When creating custom fields, set a maximum width of 15-20 characters for optimum display on application pages.

- Hide or show standard fields
- Extend form regions
- Extend table regions
- Reorder fields (tables and pages)
- Change field labels
- Add subtabs

You can also set conditions on your new page layouts. For example, maybe only certain roles can see the fields that you add to a page.

Related Topics

- [Modify Application Pages Using Application Composer](#)
- [Overview of Dynamic Page Layouts](#)

Modify Application Pages Using Application Composer

To modify application pages using Application Composer:

1. On the main Overview page in Application Composer, select an object and expand its nodes.
2. Click the **Pages** node and select the **Application Pages** tab.
3. Duplicate and modify page layouts for that object, as needed.

Application Pages for Standard Objects

Standard objects are delivered with standard layouts. To modify a layout, duplicate the standard layout for the following types of application pages:

- Landing page (list page)
- Creation page
- Details page (edit page)
- Search and select dialog box

For more information on how to work with these pages, see [Page Layouts for Standard Objects](#).

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Field Groups](#)
- [Page Layouts for Standard Objects](#)
- [Define Objects](#)

Application Pages for Custom Objects

Application pages are available for all custom objects. You can create a set of application pages with a click of a button, after creating a custom object.

When you do so, Application Composer automatically creates default custom layouts for the following:

- Landing page
- Creation page
- Details page

- Search and select dialog box

Working with these page layouts is exactly the same as working with the page layouts for standard objects.

In the following sections, you will learn how to:

- Create application pages for custom objects
- Select a display icon for application pages
- View custom object application pages at runtime

Create Application Pages for Custom Objects

1. Make sure you're working in an active sandbox.
2. In Application Composer's object tree, expand the object that you want to create page layouts for.
3. Click the **Pages** node.
4. On the **Application Pages** tab, click the **Create Default Pages** button.

Select the Display Icon for Application Pages

Application Composer creates custom objects with a default icon, but you can change it. Your end users can view this icon in many locations, including the springboard strip of application pages.

To select the icon:

1. Click the object's node in the object tree to view its Overview page.
2. In the Display Icon region of the Overview page, select an icon.

See Define Objects for more information.

View Custom Object Application Pages at Runtime

After configuring the application pages for your custom object, test the runtime pages by clicking the Home icon. Depending on which application you created your custom object in, you might have to wait a few minutes for the new display icon to appear on the Home page.

Tip: Use the Favorites and Recent Items menu to switch between runtime pages and Application Composer design time pages.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Field Groups](#)
- [Page Layouts for Standard Objects](#)
- [Define Objects](#)

Search and Select Dialog Boxes

About Search and Select Dialog Boxes

A search and select dialog box is a quick way for your end users to look for records at runtime. For example, your users may need to assign solutions to service requests, but what if your company's knowledge base has over 1,000 solutions?

Scrolling through 1,000 records to find the right solution is impossible. Instead, users can leverage a search and select dialog box to enter search criteria (for example, all solutions relating to a particular product) to easily find what they're looking for. And, if your users don't find what they need in the search results table, then they can also create a new record by clicking the **Create** button.

You can configure search and select dialog boxes for both custom objects, and most standard objects.

A search and select dialog box includes two regions that you can configure:

- A search region where your users can enter search criteria
- A search results table where your users can review the search results and select a record

Note: You can add dynamic choice list fields to the search results table, but for optimal performance, users can't sort on those columns at runtime. Also, fixed choice lists added to the Search and Select dialog box may appear wider than other fields.

Your users can launch search and select dialog boxes from two areas:

- From a dynamic choice list field, commonly referred to as a list of values.
- From a subtab.

Related Topics

- [Application Pages for Custom Objects](#)

Configure a Search and Select Dialog Box for Custom Objects

Application Composer automatically creates a search and select dialog box when you create a set of application pages for a custom object. You can then configure and reuse that dialog box wherever you want to associate the custom object with another object.

To configure a search and select dialog box for a custom object:

1. In Application Composer, navigate to the Application Pages tab for a custom object.
2. Click **Create Default Pages** if you haven't yet created the set of application pages for a custom object. See [Create Application Pages for Custom Objects](#).
3. On the Application Pages tab for the object, go to Reusable Region and edit the **Default custom layout**. You can also duplicate an existing custom layout.
4. On the Edit Search and Select Dialog Box Layout page, configure both the search and table regions of the search and select dialog box.

For example, select which fields to show and hide. You can also rearrange the fields in the search region by moving the fields up and down inside the Selected Fields pane.

Note: If possible, avoid adding a parent/child pair of dependent fixed choice list fields to the search region, because the two fixed choice list fields operate independently at runtime.

5. For each field that appears in the search region, you can set these additional search options:
 - o Set a default search operator, such as Equals or Contains.
 - o Indicate if a search field is required.
 - o Select the **At Least One Is Required** check box for two or more search fields to indicate that at least one of those fields is required.

After you configure the look of the search and select dialog box, you can optionally set default values that appear in the search region to users at runtime. You set default values in the creation page layout or details page layout of the object where the dialog box appears. See [Prepopulate a Search and Select Dialog Box with Default Values](#) for detailed instructions.

Related Topics

- [Application Pages for Custom Objects](#)

Examples of How You Can Use Custom Object Search and Select Dialog Boxes

Use a search and select dialog box that you configured for a custom object to let your users associate the custom object with any other object. Here are some examples.

Example 1

Maybe your users want to assign a solution to a service request. In which case, they will assign a solution to a service request using a dynamic choice list field.

1. Create a custom object for the Solution object.
 - a. Create the set of default application pages for the object.
 - b. Configure a search and select dialog box for the Solution object.
2. Create a custom object for the Service Request object.
 - a. Create a dynamic choice list field, Solution, that's populated with records from the Solution object.

This field will automatically use the search and select dialog box that you configured in step 1.a.

When you create the dynamic choice list field, Application Composer creates a one-to-many relationship between the Solution and Service Request objects. In other words, one solution can be associated with many service requests, but a service request can have only one solution.
 - b. Add the new Solution field to the Create Service Request and Edit Service Request pages.

At runtime, your users will use the Solution's search and select dialog box, available from the Solution dynamic choice list field, to search for and select a solution to assign to a service request record.

Tip: Optionally, if you want users to view all service requests associated with a solution, you could add a Service Requests related object subtab to the Edit Solution page. However, to do this, you must create a search and select dialog box for the Service Request object.

Example 2

In this next example, you assign solution records to the company employee who authored the solutions. To do this, you would use the same search and select dialog box that you configured for the Solution object on a subtab.

1. Create a one-to-many relationship between the Resource and Solution objects, using the Relationships page, available under the Common Setup pane in Application Composer.

In this case, one resource can author multiple solutions, but a solution can have only one author.

2. On the Edit Resource page, create a related object subtab called Solutions that's populated with records from the Solution object.

At runtime, your users can search for and select one or more solutions to assign to a resource. They can also create a new solution record to assign to the resource, right from the search and select: Solutions dialog box.

Related Topics

- [Application Pages for Custom Objects](#)

Configure a Search and Select Dialog Box for Standard Objects

You can configure the search and select dialog boxes that are automatically provided for most standard objects.

Standard objects that have extensible search and select dialog boxes include:

- Account
- Asset
- Campaign
- Contact
- Household
- Opportunity
- Partner
- Product
- Product Group
- Resource
- Sales Lead
- Service request

Note: Changes made using Page Composer in previous releases to configure the non-extensible search and select dialog boxes for the above objects aren't migrated to these extensible search and select dialog boxes. Repeat the same configuration using Application Composer to replicate the same look and feel in the extensible search and select dialog boxes.

To configure a search and select dialog box for a standard object:

1. In Application Composer, select a standard object.
2. Expand the object's tree structure and click the Pages node.
3. On the Application Pages tab, go to the Reusable Regions region, and edit the **Default custom layout**.

If the default custom layout doesn't yet exist, then duplicate the standard layout. You can also duplicate any existing custom layout.

Tip: You can configure multiple custom layouts for a search and select dialog box of a standard object. For example, you can display one version of the Contact search and select dialog box on the Edit Account page and a different version of the same Contact search and select dialog box on the Edit Opportunity page.

4. On the Edit Search and Select Dialog Box Layout page, configure both the search and table regions of the search and select dialog box.

For example, select which fields to show and hide. You can also rearrange the fields in the search region by moving the fields up and down inside the Selected Fields pane.

Note: Some fields that appear in the search region are grayed out. You can't remove these fields, but you can rearrange their order. If possible, avoid adding a parent/child pair of dependent fixed choice list fields to the search region, because the two fixed choice list fields operate independently at runtime.

5. For each field that appears in the search region, you can set these additional search options:
 - o Set a default search operator, such as Equals or Contains.
 - o Indicate if a search field is required.
 - o Select the **At Least One Is Required** check box for two or more search fields to indicate that at least one of those fields is required.

After you configure the look of the search and select dialog box, you can optionally set default values that appear in the search region to users at runtime. You set default values in the creation page layout or details page layout of the object where the dialog box appears. See [Prepopulate a Search and Select Dialog Box with Default Values](#) for detailed instructions.

Related Topics

- [Application Pages for Custom Objects](#)

Prepopulate a Search and Select Dialog Box with Default Values

Configure a search and select dialog box to display default values in the search region where users enter search criteria at runtime. You can also set default operators, such as Starts with, End with, and Equals.

Prepopulating a search and select dialog box per page layout replaces the saved search capability that was available in previous releases.

To set default search values in search and select dialog boxes:

1. In Application Composer, select the object that displays the dynamic choice list field, whose search and select dialog box you're configuring.

For example, the Primary Contact dynamic choice list field appears on the Edit Account page. To set default values for that field's Contact search and select dialog box, go to the Account object.

2. Expand the object's tree structure and click the Pages node.
3. On the Application Pages tab, go to either the creation page layout or details page layout for the object, and edit the region where the dynamic choice list field appears.
4. All dynamic choice list fields that appear in this region are displayed in a table. Find the search and select dialog box that you want to set default values for, and click the Edit icon under **Default Search Value**.
5. In the Layout field, select the search and select dialog box layout that you want to set default values for.

Tip: You can configure multiple custom layouts for a search and select dialog box for a standard object. For example, you can display one version of the Contact search and select dialog box on the Edit Account page, and a different version of the same Contact search and select dialog box on the Edit Opportunity page.

6. For each field that appears in the search region, you can set these search options:

- o Set a default search operator, such as Equals or Contains.
- o Indicate the value type of the value that you're entering as a default:
 - Expression
 - Literal value
 - Object field

The value of an object field comes from the source object. In our example of the Contact search and select dialog box that is launched from the Edit Account field, the object field you select here is from the Account object record.

- o Enter the default search value.

Note: The search and select dialog box prepopulates with default values only if there are no user-entered values. The dialog box overrides any prepopulated values with the search criteria that your users enter.

- o If you don't want the default search value to be changed at runtime, click the **Read Only** check box.

If you would like different default search values to display for the Contact search and select dialog box on the Edit Opportunity page, then go to the Opportunity object > Pages node and follow the same procedure again.

Note: To make the default search values display in the Opportunity Revenue item, you must prepopulate the search and select dialog box in the Opportunity Line Item table and on the Opportunity Detail page.

Related Topics

- [Application Pages for Custom Objects](#)

Dynamic Page Layouts

Overview of Dynamic Page Layouts

Present the same page to multiple users differently using page layouts. Set conditions to page layouts for controlled visibility. For example, a sales executive can see privileged fields on an opportunity record, but not other sales team members.

Similarly, an open opportunity may have fields that are in progress, which won't appear in a closed opportunity.

Where Can You Use Dynamic Page Layouts?

You can create page layouts for both standard and custom objects. Create page layouts for these types of application pages:

- Landing page (list page)
- Creation page
- Details page (edit page)
- Search and select dialog

Control Visibility of Page Layouts

When you create a page layout, you set one or more conditions to control when that layout is displayed.

The conditions you can attach to a layout include:

- Type of record
Not supported for landing page (list page) custom layouts.
- Role
- Advanced expression
Not supported for landing page (list page) custom layouts.

The record type and role conditions are convenient, declarative ways of attaching conditions to a layout. Use an expression to control the display of a layout, only if the record type and role conditions don't meet your needs.

Examples of layout conditions include:

- Type of record
 - Only display the Closed Reason field on an opportunity, when the opportunity is closed.
 - Display different page layouts depending on the product category. For example, display different fields if the product category is a physical item, or if it's a service pack.
 - Display different page layouts depending on type of activity, such as a telephone call, task, or appointment.
- Role

- A sales manager might see fields related to approving an opportunity, whereas the sales representative wouldn't see those fields.
- Some opportunity fields might apply only to field sales representatives, some fields to inside sales representatives, and some fields to follow-up sales representatives.
- Expression
 - Don't allow users to add a revenue item or a product to an opportunity, after a quote has been generated and approved, or while an opportunity is in approval.
 - Control the display of page layouts based on the user's location, language, or device.
 - Write an expression to combine multiple conditions.

Related Topics

- [Page Layouts for Standard Objects](#)
- [Field Groups](#)
- [Use Field Values to Control a Page Display](#)
- [Control a Page Display Based on a User's Role](#)
- [Use Advanced Expressions to Control a Page Display](#)

Field Groups

Use field groups to organize your pages and make them look more readable. A field group lets you group fields into collapsible regions, each with its own header that you can modify.

Create field groups as part of either a creation page layout or a details page layout. For detailed instructions on how to create field groups, see [Add Field Groups to a Page Layout](#).

Which Fields Can You Group?

The fields that you can select for a field group are attributes of the top-level object that you're creating the page layout for, such as the opportunity object.

Why Use Field Groups?

Field groups organize your page layouts. Here are some reasons why you should use them:

- Group related fields so they always appear together on a page.
Perhaps you want a group of fields, such as Home Ownership and Purchase Date of Home, to always appear together. Group them.
- Group secondary fields in a region that your end users can optionally expand, if they need to.
Maybe some fields on a page are useful, but not critical for your end users. Define the field group to be collapsed by default at runtime.
- Manage page layouts with fewer clicks.
Once you combine fields into a field group, you can easily move that group up or down the page layout, with a single click.

Multiple field groups always appear together at runtime within a larger field group "container". When designing a page layout, you can move a field group up or down, but only within this larger container.

In most cases, field groups appear at runtime as regions right below the page's top summary region.

Field Group Validation

Application Composer validates the contents of field groups. You can't add the same field to different groups. This validation applies only across the field groups created for one page type (creation page or details page).

Tip: Although you can't add the same field to multiple field groups, you can easily move a field between groups. This makes it easy to manage fields within groups, if you later change your mind about field placement.

Performance Considerations: Keep Field Groups Collapsed

Field groups are displayed inside a panel group that users can expand or collapse, but are initially in expanded mode by default. However, having 3 or 4 field groups on a page reduces page performance because, as part of drilling down on a record, multiple UI events must be fired to expand the field groups.

For optimal performance, always keep field groups on a page as collapsed by default. Users can then expand field groups as needed.

Performance Considerations: Plan Your Layouts in Advance

You can move fields between field groups, but when customizing page layouts, avoid frequent shuttling of fields between regions and field groups.

As a best practice, plan layouts before making field changes:

1. Decide which fields to include on each page, as well as their relative position (sequence) on the layout. Then add the fields in order.
2. Avoid repeatedly selecting and deselecting the same fields on the same pages.

Each time you select or deselect a field and save the layout, a corresponding customization instruction is saved and remains in metadata. Frequent selecting and deselecting of fields could lead to a large number of instructions that must be resolved at runtime.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Page Layouts for Standard Objects](#)
- [Use Field Values to Control a Page Display](#)
- [Control a Page Display Based on a User's Role](#)
- [Use Advanced Expressions to Control a Page Display](#)

Page Layouts for Standard Objects

All supported objects are delivered with a standard layout for their application user interface pages, called standard layouts. Standard layouts are the pristine model layouts that you can't edit.

However, you can:

- Duplicate standard layouts to create custom layouts, and then make your edits.

The first custom layout for a page type is automatically named the default custom layout, but you can change the name.

- Edit custom layouts.
- Inactivate or deprecate custom layouts.

You can't delete page layouts, but you can inactivate layouts by selecting an active layout on the Application Pages tab, and then clicking the **Inactivate** button. Filter the page layouts using the **Layout Status** drop-down.

Note: The **Inactivate** button appears only when you select an active page layout. And, the **Activate** button appears only when you select an inactivated or deprecated layout.

You can't inactivate the standard layouts that are automatically delivered for an object.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Field Groups](#)
- [Application Pages for Custom Objects](#)
- [About Search and Select Dialog Boxes](#)

Page Layouts for Custom Objects

Working with page layouts for custom objects is exactly the same as working with page layouts for standard objects.

The only difference is that after you create a custom object, you must manually create its set of custom layouts before you can start to work with them. Custom objects don't have a set of standard layouts.

Create a set of application page layouts for a custom object with a click of a button. Doing so lets Application Composer automatically create the following:

- Page layouts for the object's set of user interface pages, such as the creation and details pages.

To learn how to create a set of application pages for a custom object, see [Create Application Pages for Custom Objects](#).

- A Search and Select dialog, which you can configure.

To learn how to configure the Search and Select dialog for custom objects, see [Configure a Search and Select Dialog Box for Custom Objects](#).

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Field Groups](#)
- [Application Pages for Custom Objects](#)
- [About Search and Select Dialog Boxes](#)

Standard Layouts vs. Custom Layouts

Standard layouts are the pristine model layouts that you can't edit. Custom layouts are copies of standard layouts that you make, which you can edit.

Standard layouts exist to make your upgrades seamless. When you upgrade to a new release, Oracle upgrades only the standard layouts for each object. Your custom layouts aren't touched. This makes it easy for your users to continue working immediately after an upgrade. In the meantime, you can take your time to review the changes that happened to standard layouts as part of an upgrade, and manually incorporate those changes as and when needed.

After an upgrade, you can easily review the newly upgraded standard layouts by deactivating all existing custom layouts for a page type, such as the creation page. Then, sign in as a user to view the standard layout at runtime. Observe the changes for the creation page and, if desired, go back to Application Composer to incorporate those changes into your creation page custom layouts and reactivate them.

If an object has one or more custom layouts for some page types, but not for others, then Oracle considers the whole set of pages for that object to be modified. As part of the upgrade, Oracle:

1. Doesn't touch the custom layouts that already exist, as usual.
2. Creates custom layouts for those pages that don't have any custom layouts. These new custom layouts preserve what users experienced before the upgrade.
3. Upgrades all standard layouts, as usual.

Since standard layouts are model layouts that you can't edit, this means that you can't make changes to the page using Page Composer if the standard layout displays at runtime. However, customers can still personalize the page.

Tip: If your page layout isn't displaying as expected, then try recreating the layout.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Field Groups](#)
- [Application Pages for Custom Objects](#)
- [About Search and Select Dialog Boxes](#)

Edit Page Layouts

Edit a custom layout to do the following:

- Add fields, actions, links, buttons, and subtabs.
If you add the Attachments field, then at runtime, users can add attachments to the record.
- Hide and show, reorder, and relabel regions, including subtabs.

To edit a custom layout:

1. In Application Composer, select the object that you want to create custom layouts for.
2. Expand the object in the object tree, and select the Pages node.

3. Select the Application Pages tab.
4. Find the type of page that you want to modify, such as a creation page layout or a details page layout. Click the **Duplicate Layout** icon to duplicate and edit an existing layout.
5. Design the page layout. Depending on how the page is designed, you can add fields, actions, links, and buttons.

You can also hide and show, reorder, and relabel regions.

If the page layout is for a details page, then you can also add and reorder subtabs, and add the Attachments field.

Tip: Adding the Attachments field lets users add attachments to the record at runtime.

At runtime, while editing an object, you can mark the record as your favorite using the Add to favorites icon. See "User Favorites" in the Using Sales guide for more information.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Field Groups](#)
- [Application Pages for Custom Objects](#)
- [About Search and Select Dialog Boxes](#)

Add Field Groups to a Page Layout

To create a field group:

1. On a custom layout page, click the New icon in the Field Groups region.
2. On the Create Field Group: Configure Field Group Details page:

- a. Enter the name of the field group.

At runtime, the name is displayed as the name of this collapsible region.

- b. Indicate if the region is automatically expanded, or collapsed by default.

Note: For optimal performance, always keep field groups on a page as collapsed by default. Users can then expand field groups as needed.

- c. You can also set the position of the field group in relation to other field groups, if other field groups already exist.
- d. Click **Next**.

3. Add fields to the group.

Tip: You can multi-select and double click fields in the Available Fields list to move them to the Selected Fields list.

4. If additional custom layouts exist, then you can click Next to add this field group to other custom layouts.

Note: You can't add the same field to different field groups. But, this validation applies only across the field groups created for one page type (creation page or details page). If you add a field group to multiple custom layouts, then it's possible that a field in the field group could already be present on the other custom layouts. After adding a field group to other custom layouts, review those pages to ensure that such duplicates are removed.

5. Click **Save**.

Your new field group now appears in the Field Groups region.

6. After your field groups are defined, you can optionally hide them, or reorder them within the larger field groups "container" in the custom layout.
7. When you finish making changes to your custom layout, click **Done**.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Field Groups](#)
- [Application Pages for Custom Objects](#)
- [About Search and Select Dialog Boxes](#)

Assign Conditions to Page Layouts

You can assign conditions to a custom layout that control when the layout is displayed. You can't assign conditions to a standard layout.

Note: The layouts you create are displayed in a table, and the order of layouts in each table is significant. At runtime, Application Composer evaluates the condition or conditions specified in each layout, starting with the first layout listed in the table. The first layout that matches all Type, Role, and Expression conditions is selected for display at runtime. The standard layout is always the last layout in the table, and it can't be deleted or inactivated.

Assign one or more of these conditions to a custom layout.

- Type
 - a. Select the custom layout you want to add conditions to.
 - b. Select a record type field value, if a record type field has been created for the object. At runtime, if the value is selected, then this custom layout displays.

The default value for this condition is ANY, so if you do not specify a Type condition for a layout, then Application Composer views this condition as satisfied when evaluating a layout for display at runtime.

You can't assign this condition to landing page (list page) custom layouts.

- Role
 - a. Select the custom layout you want to add conditions to.
 - b. Select the role; this is the audience that can view this page layout. For example, perhaps only the sales representative can see this custom layout at runtime. Custom roles, which are copies of the predefined

roles that Oracle provides for all customers, are displayed by default for you to select. However, you can optionally choose to display predefined roles, as well.

The default value for this condition is ANY, so if you do not specify a Role condition for a layout, then Application Composer views this condition as satisfied when evaluating a layout for display at runtime.

- Advanced Expression
 - a. Select the custom layout you want to add conditions to.
 - b. Click the Calculator icon.
 - c. Enter a Groovy expression that controls when this custom layout is displayed.

You can't assign this condition to landing page (list page) custom layouts.

Tip: In general, it's best to keep a single custom layout, for each page type, condition-free. This way, if no conditions apply at runtime, then at least your users see a generic custom layout. Otherwise, if no conditions apply at runtime, then the standard layout displays.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Field Groups](#)
- [Application Pages for Custom Objects](#)
- [About Search and Select Dialog Boxes](#)

Configure Page Titles for Details Pages

Do your end users identify an opportunity by its number?

In that case, instead of having them navigate to the opportunity details each time to determine the opportunity number, configure the custom details page layout to display the opportunity number in the page title itself. Thus, giving your users quick and easy access to more relevant information about the record.

You can replace the default page title of a standard or custom object's custom details page with a field's value and its label, or just the field's value. You can also reset the title to its default value.

To modify the default title of an object's custom details page:

1. In Application Composer, expand the object and click the **Pages** node.
2. On the **Application Pages** tab, navigate to the **Details Page Layouts** section, and click the custom page layout whose title you want to change.
3. On the selected custom layout page, click the edit icon next to the **Page Title** field.

The **Edit Page Title** dialog opens.

4. From the **Add Field** drop-down list, select the field whose value you want to display in the page title.
5. To display the field's label as well, select **Show Field Label**.
6. Click **OK**.

The page title changes to the selected field's value and label (if opted for) at runtime.

7. To reset the title to the default title that was shipped with the product, click the **Reset to Default Page Title** icon.

Set the Default Subtab for Details Pages

Set any subtab as the default tab on a custom (company-defined) details page of standard and custom objects.

This default subtab is the tab that your users first see when navigating to a record's details page from the landing page, workspace, or from another object. You can also reorder and remove any subtabs on the details page. However, you can't remove a subtab that has been configured as the default subtab for a page layout.

For example, a sales representative who's currently assigned to work on the contacts of an account on a daily basis can have the Contacts tab in focus and reordered to be the top-most tab when navigating to the account details.

Set the Default Subtab

To set a subtab as the default tab of an object's custom details page:

1. In Application Composer, expand the object whose custom page layout you want to edit, and then click the **Pages** node.
2. On the **Application Pages** tab, navigate to the **Details Page Layouts** section, and click the custom page layout whose default subtab you want to configure.
3. On the selected custom layout page, click the edit icon next to the **Default Subtab** field.

The **Edit Default Subtab** dialog opens.

4. From the **Default Subtab** drop-down list, select a subtab.
5. Click **OK**.

The default subtab changes to the selected subtab.

6. To reorder subtabs and to choose which subtabs you want to display on the object's details page using the Configure Subtabs dialog, click the **Hide, Show, or Reorder Subtabs** icon. For more information, see Create and Reorder Subtabs in Application Details Pages.

Note: The subtab that's selected as the default tab can't be removed from the list of Selected Subtabs while configuring the subtabs on the Configure Subtabs dialog. You must deselect the subtab as the default tab before removing it from the Selected Subtabs list.

Default Subtabs and Direct Page Links

Direct page links are links that point to a specific record. You can add such a link to any e-mail, report, or user interface page. Some objects support linking directly to a subtab. If you configure a default subtab, however, then note that the default subtab always takes precedence over any subtab included in a direct page link. This means that when a user clicks a direct page link, any subtab included in that link is ignored in favor of the default subtab configured for the page.

Use Field Values to Control a Page Display

Present a page differently to different users with page layouts, depending on the conditions you define.

One condition that you can set for a layout is based on the type of record, which your end users indicate by selecting a value from a field at runtime. For example, an open opportunity could display certain fields that a closed opportunity won't.

Control the display of page layouts this way with a special kind of choice list field, called a record type field. A record type field is a choice list field with a list of values that you specify.

Note: You can create only one record type field per object.

Create a Record Type Field

1. In Application Composer, navigate to the object that you want to create page layouts for.
2. Expand the object and click the Fields node.
3. On the Custom tab of the Fields page, click **Create a custom field**.
4. Select **Record Type** and click **OK**.
5. Enter basic field attributes, such as the field display name and whether or not the field is required and updatable.
6. Configure the list of values to display in the choice list. You can either select a predefined lookup type, or create a new one.
7. Configure which roles have what access to particular choice list values to restrict the list of values displayed at runtime by role.
Custom roles, which are copies of the predefined roles that Oracle provides for all customers, are displayed by default. However, you can optionally choose to display predefined roles also.

Next, add the field to the desired application page layout, where you want the field to appear. This step is described in the next section.

Create Page Layouts per Record Type

1. In Application Composer, expand the object and select the Pages node.
2. On the Application Pages tab, find the page layout that you want to create for a record type.
For example, to create a page layout for an open opportunity, in the Details Page Layouts region, select the standard layout and click the Duplicate Layout icon, and then make your changes in the duplicate custom layout.

Note: You can't add a record type condition to landing page (list page) custom layouts.

The first custom layout that you create from a standard layout is called the default custom layout, but you can optionally change the name. Oracle recommends that you don't add conditions to the default custom layout. You can add record type conditions, and all other conditions, to subsequent custom layouts that you create.

3. Add the record type field to the selected layout.
4. After editing and saving the layout, select a choice list value under the Type column.
At runtime, if an end user selects this value, then this layout appears.

CAUTION: Remember that during the creation of the record type field, you can also restrict the list of values by role. If you assign a Role condition to the layout as well, then confirm that both Role conditions are complementary.

Assign a Choice List Value to Control a Page Layout Display

1. Create a record type field for an object.

2. Add the field to the desired application page where you want the field to appear, such as the object's creation page or details page (edit page).
3. Assign each choice list value to a layout.

Landing page (list page) custom layouts don't support record type conditions.

At runtime, when an end user selects a value from the field, the page display changes to match the application page layout that you associated with the field value.

Examples of Page Layouts per Record Type

Examples of page layouts that you might want to create for a record type field and its values are:

- Only display the Closed Reason field on an opportunity, when the opportunity is closed.
- Display different page layouts depending on the product category. For example, display different fields if the product category is a physical item, or if it's a service pack.
- Display different page layouts depending on type of activity, such as a telephone call, task, or appointment.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Control a Page Display Based on a User's Role](#)
- [Use Advanced Expressions to Control a Page Display](#)
- [Record Type Fields](#)

Control a Page Display Based on a User's Role

Want to display the same page differently to different users depending on their roles? Then, assign a role to one or more page layouts and control which layout appears at runtime to which user.

For example, you may want to show fields related to approving an opportunity to a sales manager, but not to a sales representative.

To create a page layout for a role:

1. In Application Composer, expand the object and select the Pages node.
2. On the Application Pages tab, find the type of page layout that you want to create for a role.

For example, to create a page layout for the sales manager, in the Creation Page Layouts region, select the standard layout and click the **Duplicate Layout** icon. Then, make your changes to the duplicate custom layout.

Note: The first custom layout that you create from a standard layout is called the default custom layout, but you can change the name. Oracle recommends that you don't add conditions to the default custom layout. You can add role conditions, and all other conditions, to the next custom layouts that you create.

3. After editing and saving the custom layout, select a role under the Role column.

Custom roles, which are copies of the predefined roles that Oracle provides for all customers, are displayed by default. But, you can also choose to display predefined roles.

Ideally, your roles should be stable before you assign roles to page layouts. If you make changes to roles after you create role-specific page layouts, then you will have to come back to these page layouts and update the assigned roles.

At runtime, the page layout appears only to a user with the specified role.

CAUTION: If you assign a Type condition to the layout in addition to a Role condition, then confirm that the Role condition is complementary with any role assignments made at the record type field level.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Use Field Values to Control a Page Display](#)
- [Use Advanced Expressions to Control a Page Display](#)

Use Advanced Expressions to Control a Page Display

Use Application Composer's expression builder and write Groovy scripts to set conditions for page layouts. For example, an expression on a page layout can prevent users from updating a particular field based on the value of an opportunity.

Note: You can't add an expression condition to landing page (list page) custom layouts.

Tip: You can also control the display of pages using choice list values (using a record type field) and roles. The record type and role conditions are convenient, declarative ways of attaching conditions to a layout. Thus, supply an expression to control the display of a layout, only if the record type and role conditions don't meet your needs.

To write an expression for a page layout:

1. In Application Composer, expand the object in the object tree, and select the Pages node.
2. On the Application Pages tab, find the type of page layout that you want to create.

For example, to write an expression that controls the display of a creation page, in the Creation Page Layouts region, select the standard layout and click the **Duplicate Layout** icon. Then, make changes to the duplicate custom layout.

3. **Note:** The first custom layout that you create from a standard layout is called the default custom layout, but you can change the name. Oracle recommends that you don't add conditions to the default custom layout. You can add an expression, and all other conditions, to the next custom layouts that you create.
4. After editing and saving the layout, under the Advanced Expression column, click the calculator icon to open the expression builder.

5. Write an expression that describes the conditions required for this layout to appear at runtime.
- o If your script references one or more fields, then select those fields in the Depends On choice list, too. If those field values change at runtime, then the expression is reevaluated and the page layout is refreshed if the new condition is met.

For example, let's say you write this script:

```
PartyName == 'abc' || NoteTypeCode == 'GENERAL'
```

Your script references two fields. Thus, you must select those fields, Author and Type, in the Depends On choice list, which appears at the top of the expression builder.

Note that long text fields don't work like other fields. If your script references a long text field, such as Note Text, then you must use the toString() operator in your script. For example:

```
If (NoteText?.toString() == 'abc')
```

Remember to select the Note Text field in the Depends On choice list, too.

- o Your expression should return either a True or False value. At runtime, Application Composer interprets a True value to mean that the condition was met.

Let's look at the following examples:

- Let's say you want to display a specific page layout if the Win Probability for an opportunity is 95. In this case, your script could be:

```
if (WinProb==95)
{
return true;
}
else
{
return false
}
```

- In another example, let's say you want to show or hide some fields from the layout based on the category ID value. In this case, your script could be:

```
if ([300000018675471,300000018675472].contains(CategoryId)){
return false;
}
else
{
return true
}
```

Examples of Page Layouts that Appear Based on Expressions

- Don't allow users to add a revenue item or a product to an opportunity, after a quote has been generated and approved, or while an opportunity is in approval.
- Control the display of page layouts based on the user's location, language, or device.
- Write an expression to combine multiple conditions.

Related Topics

- [Overview of Dynamic Page Layouts](#)
- [Use Field Values to Control a Page Display](#)
- [Control a Page Display Based on a User's Role](#)

Configure the Summary Table on a Landing Page

Configure an object's landing page to define the summary table's column display. For example, set which columns appear by default.

After you configure the landing page, you can assign a role condition to the page layout to display a different set of columns on the page, per role. For example, you may want your salespeople to see different landing page information than your partner managers.

At runtime, your end users can create their own saved searches on the landing page summary table, choosing from the set of columns that you selected.

Note: For Oracle CX Sales and Fusion Service users: If you're using Workspace instead of landing pages, note that the Workspace UI isn't extensible in Application Composer. But, even though you can't use Application Composer to configure the Workspace UI, you can configure Workspace to search for custom objects and fields created in Application Composer. See the related Workspace configuration documentation for more information.

Configure the Summary Table

1. In Application Composer, expand the object whose landing page you want to configure and select the Pages node.
2. On the Application Pages tab, in the Landing Page Layouts region, edit an existing custom layout. Or, select the standard layout and click the **Duplicate** icon and make your changes to the new custom layout.
3. Navigate to the summary table region and click the **Edit** icon. Some standard objects might refer to the summary table as the overview table.
4. In the Configure Summary Table region, indicate which fields should display as columns in the landing page's summary table.
5. For each field that you select, the **Display in Summary Table** check box is automatically selected, but you can clear it.
Clearing the check box means that the column won't automatically display on the landing page at runtime. However, your end users can manually display that column when they create a saved search at runtime.
6. Click **Save and Close**.
7. After editing and saving the custom landing page layout, optionally select a role under the Role column to specify which users should see this landing page at runtime.
8. If you assigned a role condition, then sign in as that role and test your changes.

Note: You must have the appropriate job role assigned to you to see the changes.

Related Topics

- [Assign Yourself Additional Job Roles Required for Testing](#)

Enable or Disable Drill Down Fields in Application Pages

This topic covers how you enable or disable drill down fields on application pages. These drill down fields enable you to edit an object from the details page of another object.

Familiarize yourself with the following concepts before you configure drill downs:

- Object relationships
- Dynamic choice list fields
- Subtabs

Overview

The ability to drill down on a field is based on relationships that exist among the objects involved. These relationships could be either implicitly defined through a dynamic choice list type fields or joins, or explicitly defined by creating a relationship using Application Composer.

Note: You can create drill down fields for custom objects, and for standard objects whose task flow has been registered for extensibility. You can't drill down to a common component or custom object.

Use Application Composer to add drill down fields to the desired pages using the Application Pages tab. You can configure drill down for the following types of pages:

- Details and summary pages (landing pages)
- Related object subtabs
- Context link subtabs

You can also drill down from one edit page to another in a hierarchical manner. For example, drill down from the Edit Leads page to an Edit Opportunity page, and then from the Edit Opportunity Page to the Edit Primary Contact page.

Adding Drill Down Fields to Pages

When configuring a details page, you move fields from the Available Fields box to the Selected Fields box. The fields related to the object for which you're modifying the page appear in a table on the right. In the table, the drill down is enabled by default for related fields.

Note: The table appears only if the selected fields are related.

You can enable or disable drill downs using the Drill Down Enabled check box in the table. The check box is selected and grayed out for standard fields of type dynamic choice lists that are delivered by default.

For work area pages, you also specify the Drill Down Column. This column lists the field values as links.

Adding Drill Down Fields to Subtabs

You can add drill down fields to the following types of subtabs:

- Related object subtabs

When creating a related object subtab, in the Drill Down Field list, specify the field whose values should appear as drill down links in a column on the subtab.

The Selected Fields table (on the right) displays the fields available for drill down. Specify the fields that you want to display as drill down links on the subtab.

- Context link subtabs

When creating a context link subtab, in the Drill Down Field list, specify the field whose values should appear as links in a column on the subtab.

You can also limit the fields you want to display on the subtab by specifying filters in the Search Criteria region.

For more information on using filters, see the related topic, [Overview of Subtabs](#).

Related Topics

- [Object Relationships](#)
- [Overview of Dynamic Choice Lists](#)

Specify Drill-Down Fields for Custom Dynamic Choice List Fields

This example shows how you specify drill-down fields for custom dynamic choice list fields in application pages.

Specifying Drill-Down Fields

To specify a drill-down field for a custom dynamic choice list field:

1. Select a sandbox to work in and make it active.
2. Navigate to Application Composer.
3. Expand Standard Objects and click **Sales Lead**.
4. Click the **Actions and Links** link.
5. Click the Application Pages tab.
6. Edit the desired custom layout in the Details Page Layouts region.
7. Select the Summary tab, and click the edit icon.
8. Select a dynamic choice list item from the **Available Fields** column.

You have now specified the drill-down fields.

Create and Add Custom Links to Application Pages

This worked example illustrates how to create and add custom links to application pages.

This example covers:

1. Creating a custom link that launches the Google homepage.
2. Adding the custom link to the Leads page.
3. Verifying that the link appears on the UI.

Creating a Custom Link

In this step, you're creating a custom link called Test.

Note: You must have a sandbox active before you begin your tasks.

1. Navigate to Application Composer.
2. Expand **Standard Objects**, and then expand **Sales Lead**.
3. Click **Actions and Links**.
4. In the Sales Lead: Actions and Links page, select **Actions > Create**
5. Enter or select the following:

Field or Region	Value
Display Label field	Enter Test . After you enter Test, the Name field automatically displays Test .
Type field	Select Link . After you select Link , the Source field automatically shows the URL option selected, and displays it as disabled.
URL Definition region	Enter " http://www.google.com " (along with double quotation marks).

6. In the script region, enter "**http://www.google.com**".
7. Click the **Validate** icon. A success message appears.
8. Click **Save**.
Next, you must add this new link called Test to the Leads page.

Adding the Link to the Leads Page

In this step, you add the Test link to the Leads page:

1. Expand **Standard Objects**, and then expand **Sales Lead**.

2. Click **Pages** under **Sales Lead**.
3. Select the Application Pages tab.
4. In the Details Page Layouts region, duplicate the standard layout to create a new layout to edit, or edit another existing layout.
5. In the Edit Application Details Page, select the vertical tab to which you want to add the custom link. In this example, select the Summary tab.
6. Click Edit icon in the Summary region. The Edit Lead page opens.
7. In the Available Fields column, locate the **Test - Link** field and move it to the Selected Fields column.
8. Click **Save and Close**.

Testing the Link

In this step, you verify the addition of the custom link.

To test whether the Test link appears on the UI:

1. Open the Leads page.
2. Edit a lead and note the Test link on the Edit Lead: <Name> Summary page.
3. Click the **Test** link. The Google homepage appears.

Subtabs

Overview of Subtabs

Every top-level object (such as an opportunity, account, or contact) has an edit page, also known as a details page. The details page is where users go to view record details, and make changes.

An object's details page typically has multiple subtabs displayed along the left side of the page. Subtabs are useful because they display details that are related to the current record but derived from another record, or even from an external source. For example, when editing a trouble ticket record, your users might want to view the list of products that are reported on the trouble ticket. You create that subtab using Application Composer.

Where Do Subtabs Appear?

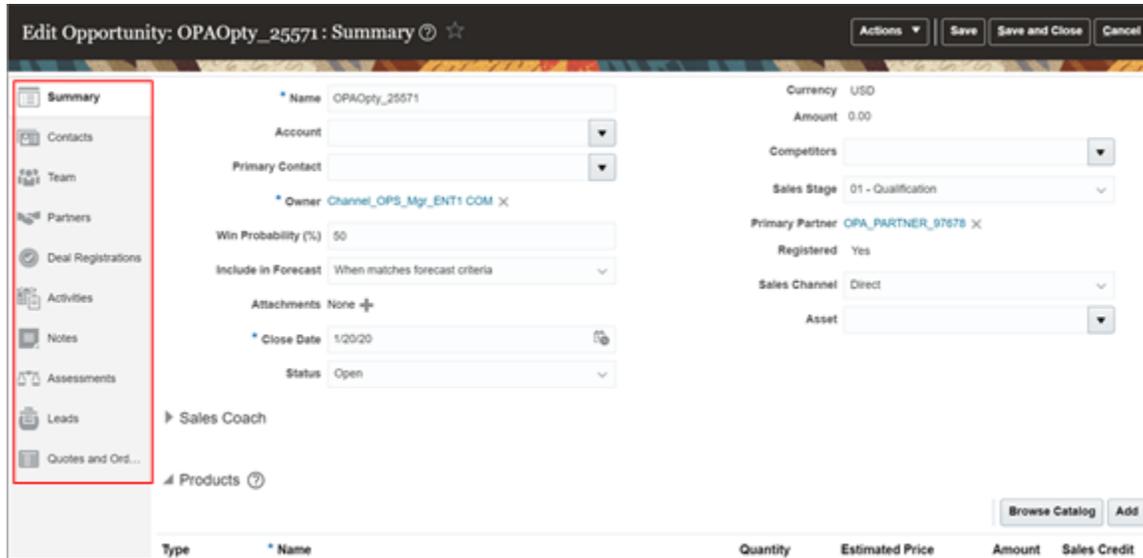
Subtabs are displayed via icons on the left side of an object's details page. Every top-level object has a details page, also known as the edit page, as part of its work area.

The details page is the page where users can view more details about an object. Depending on the security setup, users access the details page by clicking the Edit icon or by selecting the Edit menu item from the Actions menu on the summary table's toolbar. Users can also access the details page by clicking the object record name itself in the summary table.

Note that a subtab displays data in a list. Users can click any record in the list to drill down to view more details about that subtab record.

Note: When viewing a subtab on an iPad or other iOS device, advise your users to scroll to the right directly on the subtab data rows. Scrolling to the right on the subtab area outside the data rows isn't supported by iOS devices.

Let's look at the subtabs that are delivered for an object in Oracle CX Sales. In this example, these are the subtabs delivered for the Opportunity object.



Here are some subtabs that are available from the Edit Opportunity page.

- **Contacts**
This subtab lists records from the Opportunity Contact object, which is a child of the Opportunity object. Your users can click this subtab to review contacts that are related to the current opportunity record.
Tip: Child objects have a cascade delete relationship with parent objects. This means that if the parent record is deleted, then child records are also deleted.
- **Opportunity Team**
This subtab lists records from the Opportunity Team Member object, which is a child of the Opportunity object. Your users can click this subtab to review team members that are related to the current opportunity record.
- **Leads**
This subtab lists records from the Sales Lead object. This is a context link type of subtab, which displays data from any object. There is no cascade delete relationship between the Opportunity and Sales Lead objects. Your users can click this subtab to review leads that are related to the current opportunity record.

Standard Subtabs

Standard objects come with a set of standard subtabs. You can configure some regions of standard subtabs using Application Composer.

To configure standard subtabs:

1. Navigate to Application Composer.
2. Under the Objects navigation tree, expand Standard Objects, then expand any standard object.
3. Click the Pages node.
4. Click the Application Pages tab.
5. In the Details Page Layouts region, select the standard layout and click the Duplicate Layout icon to duplicate it. Click the Edit icon.
6. On the Edit Application Details Page, scroll down through the Subtabs Region and click a subtab to configure it. Note that some subtabs aren't configurable.

7. Make your subtab changes by clicking the pencil icon.

For example, you can hide and show fields. You can also apply a default search filter. See the "Adding Default Search Criteria to Subtabs" section later in this topic.

Custom Subtabs

You can add custom subtabs to both standard and custom objects. You do this by adding subtabs to a details page layout, and specifying the source of subtab data.

Let's say you want to add a new subtab to the Edit Opportunity page. In this example, let's show the customer's address. (For the sake of this example, let's assume that you previously created a custom field, Customer Address (`customerAddress_c`), on the Opportunity object to capture this information.)

Use this procedure to add a new subtab to the Edit Opportunity page.

1. Navigate to Application Composer.
2. Under the Objects navigation tree, expand Standard Objects, then expand the Opportunity object.
3. Click the Pages node.
4. Click the Application Pages tab.
5. In the Details Page Layouts region, select a layout. You can select the standard layout and click the Duplicate Layout icon to duplicate it, or select another layout. Click the Edit icon.
6. On the Edit Application Details Page, scroll down through the Subtabs Region and, at the end of the subtabs, click the Add icon.

You can add one of five types of subtabs:

- o Related object subtab
- o Child object subtab
- o Context link subtab
- o Common component subtab
- o Mashup content subtab
- o Intelligent Advisor interview subtab

These subtab types are described in "Subtab Types."

- 7.** Once you have added a custom subtab, you can then configure it.

For example, you can hide and show fields. You can also apply a default search filter. See the "Adding Default Search Criteria to Subtabs" section later in this topic.

Adding Default Search Criteria to Subtabs

Optionally add a default search filter to a subtab to help your users work more efficiently. This means that when a user opens a subtab, the records displayed are automatically filtered according to the default search filter that you configure here. Users can change the default search value, but they can't change the default search field. They can, however, further refine their search by adding additional search fields and values at runtime.

You can add a default search filter to custom subtabs, as well as to standard, extensible subtabs for select objects (such as Lead and Opportunity). Note that some standard subtabs aren't extensible, and so you can't add a default search filter to those subtabs (such as the Activities, Notes, and Social subtabs). Also, setting a default search filter isn't enabled for the Account object's Service Requests subtab.

Adding a default search filter to a subtab is optional. Skipping this step doesn't hide the subtab's search region from users (you actually can't hide this region). It just means that no search fields are automatically added to the subtab. Users can still see the search region and add fields to perform their desired search.

1. Click to edit the subtab's list region and navigate to the Configure Summary Table: Search Region section.
2. Select one default search filter field, operator, and value to apply to the subtab.

You can select from a list of both standard and custom fields. Note that you can't pick dynamic choice list fields, set ID-based fixed choice list fields, and any fields that aren't searchable. Also, if this is a subtab that was added based on a many-to-many (M:M) relationship, then you can select from only those intersection object fields that are already displayed on the subtab table.

For an example, see Configure Default Search on Standard Subtabs for Leads in the Related Topics.

Note: When configuring the search filter on standard subtabs, the search filter applies to all existing details page layouts with this subtab. Let's say you apply a search filter to the Lead subtab on an Opportunity details page layout. This means that the same search filter is applied to the Lead subtab on an Account details page layout.

Hiding Subtabs

Let's say you want to hide the Appointments subtab on the Edit Opportunity page.

Use this procedure to hide subtabs.

1. Return to the Edit Application Details Page for the Opportunity object.
2. Click the Configure Subtabs icon (green arrows).
3. In the **Configure Subtabs** dialog, in the Selected Subtabs list, double-click the Appointments subtab.
4. Click **OK**.
5. Click **Done**.
6. When you navigate to the Edit Opportunity page at runtime, the Appointments subtab is no longer available.

Related Topics

- [Subtab Types](#)
- [Set the Default Subtab for Details Pages](#)
- [Create and Reorder Subtabs in Application Details Pages](#)
- [Configure Default Search on Standard Subtabs for Leads](#)
- [Configure Default Search on Standard Subtabs for Opportunities](#)

Subtab Types

Subtabs display details that are related to the current record but derived from another object entirely, or even from an external source. Subtabs display on a top-level object's details page.

The details pages for standard objects are delivered with a set number of subtabs, but you can add more if required. You can add subtabs to the details pages for custom objects. Using Application Composer, you can create the following types of subtabs.

- Related object subtab
- Child object subtab

- Context link subtab
- Common component subtab
- Mashup content subtab
- Intelligent Advisor interview subtab

Related Topics

- [Overview of Subtabs](#)
- [Manage Security by Object](#)
- [Create and Reorder Subtabs in Application Details Pages](#)

Related Object Subtabs

A related object subtab lists records from one object that's related to another object.

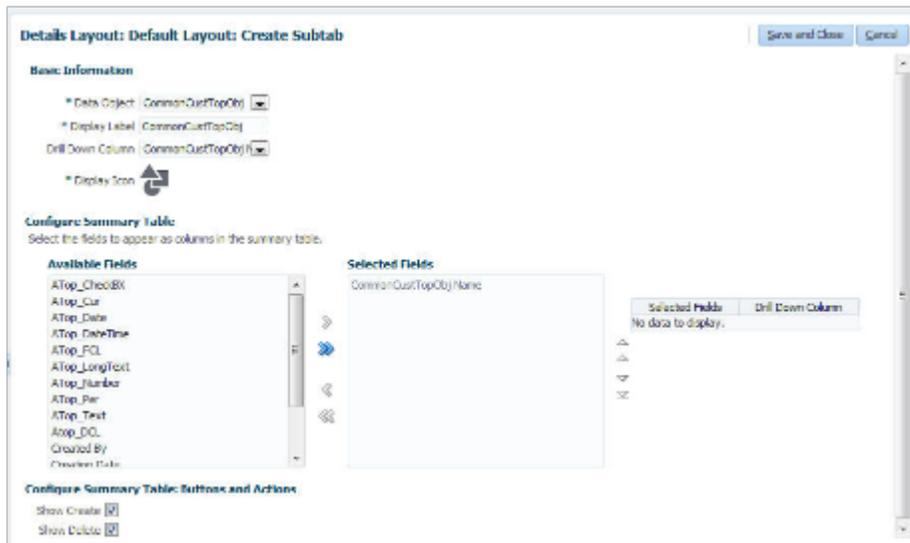
What does it mean when an object is related to another? A relationship is a foreign key association between two objects. Using Application Composer, you can create a one-to-many or many-to-many relationship between two objects. This is helpful because if a relationship exists, then you can expose the "many" object records on a subtab that's displayed on the "one" object's details page. This is useful for your users.

For example, your users might want to associate an account with a list of service requests that have been logged for that account. To enable this, you must first create a one-to-many relationship between the account and service request objects. (An account can have multiple service requests associated to it, and a given service request can have one and only one account associated with it.) Then, you can add the Service Requests subtab to the Account details page. At runtime, when your users review a particular account record, they can see all the service requests that have been logged for that account. And, depending on how you configure the subtab, they can also create new service requests, add existing service requests, or remove service requests from the subtab.

To add a related object subtab to an existing details page:

1. From any custom details page layout, click the Add icon, which displays under the existing subtabs while editing the layout.

2. Select **Related object**, then **Next**.



3. On the Create Subtab page:

- a. Select the related object that's to be exposed on the subtab, and enter the subtab display label.

The list of related objects includes those objects that:

- Are already related to the current object, with any type of one-to-many or many-to-many relationship, either custom or standard.

For example, if you previously created a one-to-many reference relationship, then you will see the "many" object in the data object list when creating the subtab for the "one" object's details page.

- Don't yet have a subtab.

After the subtab is created, you won't be able to create a second subtab for that same "many" object again.

- b. Specify the drill down column. The drill down column is the field that you want your end users to click to drill down to more details about the related object record.

If you're creating a subtab for a custom object, then the Drill Down Column field might not appear if you haven't yet created the custom object's application UI pages. To display the Drill Down Column field, you must first create the application UI pages for the custom object, sign out, and then sign back in.

- c. Specify a display icon for the subtab.
- d. Select which fields you want to display on the subtab summary table at runtime.

When selecting the fields for display on a related object subtab, join fields are not available for selection if the relationship is a many-to-many relationship.

- e. Configure the default search setting that applies to this subtab. This setting applies to all existing details page layouts with this subtab.

When configuring the default search setting, only intersection fields are available for selection if the relationship is a many-to-many relationship. See "Adding Default Search Criteria to Subtabs" in the Overview of Subtabs topic.

- f. Configure the default sorting field and its order, either **Ascending** or **Descending**.
- g. Optionally hide the Create, Add, and Remove buttons that appear on the subtab at runtime.

In a related object subtab, your users can use these buttons at runtime to:

- Create a related object record and associate it to the current record at the same time.
 - Note:** This button is available only for subtabs in application pages.
 - Add an existing related object record to the current record.
 - Remove a related object record from the subtab. This removes the association between the related object record and the current record.
- h. (Application UI only) If more than one details page layout exists, then click **Next** to optionally select other details page layouts that will display this subtab.
 - i. Click **Save and Close**.

Add a Related Object Subtab to A Details Page

This example illustrates how you add a related object subtab to a application details page.

This example covers the following:

1. Creating a related object subtab.
2. Verifying the addition of the subtab.

1. Create a Subtab

In this step, you create a related object subtab using the Sales Lead object to display fields from the Opportunity object.

For more information about creating subtabs, see Overview of Subtabs.

To create a subtab:

1. Navigate to Application Composer.
2. Under the **Standard Objects** tree, expand the **Sales Lead** object.
3. Click the **Pages** node.
4. Select the Application Pages tab.
5. Under the Details Page Layouts region, select the page layout that you want to edit. In this example, select the standard layout and click the Duplicate Layout icon to make a copy of it to edit.
6. On the Edit Layout page, select the plus icon in the subtabs region.
7. On the Create Subtab page, select the **Related object** option.
8. Click **Next**.
9. Specify the values for the fields as follows:

Field Label	Select or Enter	Description
Data Object	Opportunity	The object that this subtab is based on. This subtab will display a list of opportunities that are related to a sales lead.
Display Label	Opportunity Name	Name of the subtab.
Drill Down Column	Opportunity Name	Your users will click this column to drill down to view more details about an opportunity record.

Field Label	Select or Enter	Description
Display Icon	As applicable	This icon appears on the application page as the subtab icon.
Selected Fields	As applicable	The fields that you want to appear on the subtab.

10. Save the Opportunity Name subtab.

You have created a subtab to display opportunities on a subtab which appears on the Sales Lead details page. To view the new subtab, use the Navigator to view the Leads set of pages, and edit a sales lead record.

2. Verify the Addition of the Subtab

- Using the Navigator, select **Leads**.
- Click a lead to open the **Edit Lead** page.

The Opportunity Name subtab appears along the left side, with all the other subtabs.

You have now verified the addition of the subtab.

Related Topics

- [Subtab Types](#)
- [Enable or Disable Drill Down Fields in Application Pages](#)

Child Object Subtabs

A child object subtab lists records from a child object that are related to its parent.

A parent child relationship is a one-to-many relationship: one parent record can have many children records. When you create a child record, it's created specifically in the context of its parent. The child object's records are deleted if the parent object record is deleted. A child object doesn't have its own work area, and exists only as a subtab on the parent object's details page.

An example of a parent child relationship is the relationship between the Opportunity object, and its children, such as Opportunity Contact and Opportunity Partner. In this example, contacts and partners are created in the context of the parent opportunity record. At runtime, when your users review a particular opportunity record, they can see all the contacts and partners that have been created for that opportunity. They can edit or add new contacts and partners directly inline, in each subtab.

For custom child object subtabs that you create, your users can create or edit records inline in each child object subtab. But, you can also configure the subtab so that your users can create or edit records in a full-sized dialog window, in addition to inline editing. This is more usable if there are a large number of fields for users to fill out. This setup is described below.

To add a child object subtab to an existing details page:

1. From any custom details page layout, click the Add icon, which displays under the existing subtabs while editing the layout.
2. Select **Child object**, then **Next**.

3. On the Create Subtab page:

- a. Select the child object that's to be exposed on the subtab, and enter the subtab display label.

The list of child objects includes those objects that:

- Are already related to the current object in a parent child relationship.
- Don't yet have a subtab.

After the subtab is created, you won't be able to create a second subtab for that same child object again.

- b. Specify a display icon for the subtab.
- c. In the Configure Summary Table region, select which fields to display on the subtab summary table at runtime.
- d. In the Configure Summary Table: Search Region, add a default search filter to apply to this subtab. This setting applies to all existing details page layouts with this subtab.
- e. In the Configure Summary Table: Default Sorting region, configure the default sorting field and its order, either **Ascending** or **Descending**.
- f. In the Configure Summary Table: Buttons and Actions, optionally hide the Create, Edit, and Remove actions on the subtab at runtime.

In a child object subtab, you can hide these actions, or keep them available. If you keep these check boxes selected, then your users can:

- Create a child object record inline.
- Edit an existing child object record inline.
- Delete a child object record inline.

- g. (Application UI only) If more than one details page layout exists, then click **Next** to optionally select other details page layouts that will display this subtab.
- h. Click **Save and Close**.
- i. (Application UI only) If you want your users to create or edit new records in a full-sized dialog window in addition to inline editing, then after creating the subtab, complete these additional steps. This is more usable if there are a large number of fields for users to fill out.

- i.
 - i. On the details page layout where the child object subtab appears, click the subtab that you want to configure.
 - ii. Click the Edit pencil icon.
 - iii. Click the **Enable Separate Create Pages** and **Enable Separate Edit Pages** check boxes.

Note: The drill down column appears, but you can't change it. It's usually the child object's record name. If the child object's record name is configured as an autogenerated number, then the drill down column is removable. However, don't remove it if you want to enable separate edit pages for this subtab.

- i.
 - iv. Click **Save and Close**.

Once completed, your users can create child records in a separate full-sized dialog window. To edit existing child records, they can click the record name to drill into a full-sized edit page.

Even after you enable this full-sized record creation and editing capability, your users can still opt to do their record creation and editing inline, if they choose to. Both options are now available to them.

This feature is available for child object subtabs based on custom child objects.

Related Topics

- [Subtab Types](#)

Context Link Subtabs

A context link subtab displays a filtered list of records from any top-level object, where the filter is often based on the runtime values from the current object. The object doesn't have to be related to the current object.

Context link subtabs are read only.

To add a context link subtab to an existing details page:

1. From any custom details page layout, click the Add icon, which displays under the existing subtabs while editing the layout.
2. Select **Context link**, then **Next**.
3. On the Context Link subtab configuration page:
 - a. Select the object that's to be exposed on the subtab, and choose the subtab display label.
 - b. Specify a display icon for the subtab.
 - c. Optionally constrain the list of records displayed at runtime using a set of search criteria for the selected object, whose runtime values must match the current object record's runtime values.

Values can be literal values, or derived from the runtime values in the current object record, or from the runtime values in the current object's parent record.

Note: If your search criteria includes a fixed choice list field, then you must specify the fixed choice list's runtime value using the lookup code, not the lookup meaning.

- d. Select which fields you want to display on the subtab's read-only summary table at runtime.

You can configure fields for the main summary table which lists the child object records or related object records.

- e. Select which fields you want to display on the subtab's read-only detail form at runtime.

You can configure fields for the detail form that appears under the summary table.

- f. Configure the default search setting that applies to this subtab. This setting applies to all existing details page layouts with this subtab.
- g. Configure the default sorting field and its order, either **Ascending** or **Descending**.
- h. (Application UI only) If more than one details page layout exists, then click **Next** to optionally select other details page layouts that will display this subtab.
- i. Click **Save and Close**.

Related Topics

- [Subtab Types](#)

Common Component Subtabs

Use the common component subtab to add either a Notes subtab or a Change History subtab.

The Notes subtab shows a list of notes related to a custom, top-level object. At runtime, users can access the Notes subtab and create a note that's tied to the current record. For example, a user can record a note on a service request record.

Tip: You can optionally modify the look of the Notes subtab from the Pages node under the Note object.

The Change History subtab shows a list of changes made to the current record. Users can see a record's change history without having to leave the record, which helps to reduce issue resolution time. You can enable this subtab for custom objects, and also for a set of standard objects.

To add a common component subtab to an existing details page:

1. When editing any custom details page layout, scroll down to the bottom of the existing subtabs.

Click the Add icon.

2. Select **Common component**.

Note: If the Common component option is disabled, then both the Notes and Change History subtabs already exist on this layout.

3. Select either the **Notes** or **Change History** check box.

Note: If a check box is disabled, then that subtab already exists on this layout.

The Notes subtab isn't secured, so anyone can immediately view it at runtime.

The Change History subtab, however, is secured. If you add the Change History subtab to a standard object, make sure your users have the right privileges to view the new subtab. For custom objects, grant your users the privilege to view this new subtab by going to the custom object's security node. See "Manage Security by Object."

Related Topics

- [Subtab Types](#)
- [Display the Change History Subtab](#)

Mashup Content Subtabs

A Mashup Content subtab exposes an external Web Site right on an object's details page. The mashup content is a result of the expression that you define, which builds the intended URL.

For example, on the Contact details page, perhaps you want to add a map using Bing maps that shows the location of the contact. The Bing Maps API expects the URL to be formatted in a certain manner. In this example, write an expression using the fields: Contact Address, Contact City, and Contact State. Then, pass the URL to the Bing Maps API.

Prerequisites

Before you add a Mashup Content subtab:

- You must first register the web application that you want create the mashup for. See the Creating Mashups topic.
- You must also ensure that the registered mashup is active. See the Editing Mashups topic.

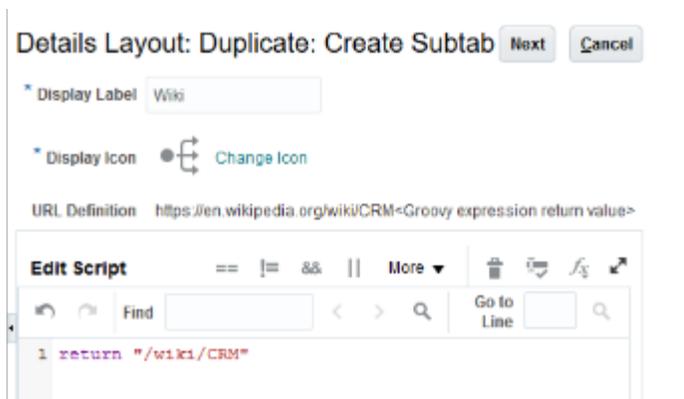
Add a Mashup Content Subtab to a Details Page

1. From any custom details page layout, click the Add icon, which displays under the existing subtabs while editing the layout.
2. Select **Mashup content**, then **Next**.

The Select Mashup Content page appears.

3. Select the mashup that you want to embed into your application.
4. Click **Insert**.

The Create Subtab page appears with the configured URL definition.



5. In the **Display Label** field, enter a name for the new subtab.
6. A default **Display Icon** appears. To change it, click **Change Icon**.
7. Based on the selected mashup type, do one of the following:
 - If the selected mashup is a parameter-based mashup, use the Add Parameters section to add values for each of the web application's URL parameters configured while registering it. These parameters are appended to the web application's URL as key-value pairs.
 - If the selected mashup is a groovy expression mashup, use the Edit Script section to edit the base URL and add to the URL definition of the registered web application.
8. (Application UI only) If more than one details page layout exists, then click **Next** to optionally select other details page layouts that will display this subtab.

9. Click **Save and Close**.

The embedded mashup appears at the bottom of the Subtabs Region page.



Note: Before end users can view the contents of your Mashup Content subtab at runtime, they must enable the settings in their selected browser to allow mixed content. Refer to the browser documentation for instructions on how to enable this option.

Related Topics

- [Subtab Types](#)
- [Overview of Mashups](#)

Intelligent Advisor Interview Subtabs

Gather highly personalized customer information by seamlessly integrating Intelligent Advisor interviews into Oracle CX Sales application pages as custom subtabs.

What's Intelligent Advisor and How Does it Work with Your Application

Intelligent Advisor is a specialized decision-making platform. It provides the benefits of a traditional technical rules platform, and implements complex policy logic that drives decision making and calculations. With a few clicks you can choose an Intelligent Advisor interview and assign it to a custom subtab. Interviews with checkpoints are automatically supported with no additional effort.

The primary key of the current parent object record is automatically seeded to the Intelligent Advisor interview to support data mapping capabilities. Data mapping enables you to pre-seed data in an Intelligent Advisor interview and to save outcomes from the interview. See the Oracle Policy Automation Documentation Library for the list of objects supporting data mappings.

Before You Begin

You must register an Intelligent Advisor Hub with your application's environment for the Intelligent Advisor interview subtab choice to appear in the list of custom subtabs.

Your application retrieves a list of available interviews from a designated Intelligent Advisor Hub. The designated Hub is assigned to your application's instance through the Intelligent Advisor Hub administration UI. You define which Intelligent Advisor Hub is assigned to your application's instance, which is a new connection type in the Intelligent Advisor Hub. The Intelligent Advisor Hub automatically configures the instance.

CAUTION: If this configuration changes after you create an Intelligent Advisor interview subtab, then the subtab won't work correctly.

See [Create a CX Sales and Fusion Service Connection in Intelligent Advisor](#) in the Oracle CX Sales Implementing Sales guide for detailed instructions.

Add an Intelligent Advisor Interview Subtab to a Details Page

1. In Application Composer, expand the object whose details page layout you want to edit.
2. Click the **Pages** node and select a custom details page layout on the Application Pages tab.
3. On the Edit Layout page, click the Add (plus) icon in the Subtabs region.
4. On the Create Subtab page, select **Intelligent Advisor interview**, then **Next**.
5. The Basic Information page appears and displays the list of deployed interviews on the registered Intelligent Advisor Hub in the **Interview** drop-down list. Retain the default interview or select another.

Note: If the registered Intelligent Advisor Hub can't be contacted when the page loads, an error message appears.

6. Optionally, change the display label and icon.
7. (Application UI only) If more than one details page layout exists, then click **Next** to optionally select other details page layouts that will display this subtab.
8. Click **Save and Close**.
9. The new Intelligent Advisor interview metadata now appears on the Details Page layout. You can edit the name of the subtab or delete it using the Edit and Delete icons. You can also click the Edit icon next to the interview name to select another interview, or to change the display label and icon.
10. When you're finished, click **Done** on the Edit Layout page.

The Intelligent Advisor Interview now appears as a new subtab on the object's page at runtime. You can optionally populate runtime data into the interviews, and you can also save interview outcomes back into the application.

Related Topics

- [Subtab Types](#)
- [Oracle Intelligent Advisor](#)
- [Create a Fusion Sales and Fusion Service Connection in Intelligent Advisor](#)

Create and Reorder Subtabs in Application Details Pages

This example illustrates how to add and reorder subtabs that appear on the application details pages. In this example, let's look at adding and reordering subtabs for the Account object.

Creating a Subtab

Let's create a related object subtab for the Edit Account application page.

Note: Before you begin, you must be in an active sandbox session. Also, it's assumed that you already created a custom child object for the Account object.

To create a subtab:

1. Navigate to the Account object in Application Composer.
2. Select the Pages node.
3. On the Application Pages tab, under Details Page Layouts, duplicate the standard layout to create a new layout to edit, or edit another existing layout.
4. On the Details Layout page, click the **+** icon at the bottom left of the page. This icon appears at the bottom of all subtabs.
5. On the Create Subtab page, select **Child object**, and click **Next**.
6. On the Create Subtab: Child Object page, enter details for the child object whose records you want to capture on this custom subtab.
7. Click **Save and Close**.

Reordering a Subtab

Next, you can configure the custom subtab so that it appears closer to the top of other subtabs.

1. On the Details Layout page, click the **Reorder Subtabs** icon which appears as two green arrows at the top of the subtabs.
2. In the Configure Subtabs dialog, highlight your custom subtab and then use the up arrow to change the order of your subtab within the existing set of subtabs.
3. Click **OK**, and then click **Done**.

Related Topics

- [Overview of Subtabs](#)
- [Subtab Types](#)

Access Parent Data When Creating Subtab Records

When you add a subtab to any "parent" page, you can also configure the subtab to default parent data into new records.

This means that when your users create a new subtab record, one or more fields can be automatically populated with data from the parent record. This capability is available when the subtab is based on either a standard or custom object, and it's available for both related object subtabs and child object subtabs. This topic describes how you set this up.

Relationships and Subtabs

Two objects can be related in either a one-to-many relationship or in a many-to-many relationship. Relationships are useful because they let you associate one object's records with the records of another object. For example, maybe your users want to track the trouble tickets that get created for an account. You enable this association between records first by creating a relationship between the objects, then by creating the subtabs.

Accessing Parent Data from Subtabs

When a user creates a new record on a subtab, you can choose to have values from the parent record automatically default into a field on the subtab. You must manually enable this capability by writing Groovy to pull data from one field into another field.

To enable this capability, use the Groovy AfterCreate trigger to access parent object data. The trigger will copy parent data to the new record created on the parent object's custom subtab. You must add this Groovy, as follows:

- If the subtab is based on a one-to-many relationship:
 - Add Groovy to the "many" object.
- If the subtab is based on a many-to-many relationship:
 - Add Groovy to the intersection object.

Let's look at the following examples in the next section.

Groovy Example for a One-to-Many Relationship

You have a Vehicle subtab on the Account object. When your users create a new Vehicle record on this subtab, you want to populate the new Vehicle record with the Account's city from the City field of the Account object.

To enable this, create an AfterCreate trigger on the Vehicle object with the following code:

```
setAttribute('AccountCity_c',Account_AccToVehicle_Src?.PrimaryAddressCity);
```

- AccountCity_c is the target field on the Vehicle object that you want to populate with the value from the Account object's City field.
- Account in Account_AccToVehicle_Src? indicates the source object from which to pull the data.
- AccToVehicle in Account_AccToVehicle_Src? is the name of the 1:M Account to Vehicle relationship.
- PrimaryAddressCity is the source data from the Account object that you use to populate the AccountCity_c target field on the Vehicle object.

Groovy Example for a Many-to-Many Relationship

You have a Country Club Membership subtab on the Account object. When your users create a new Country Club Membership record on this subtab, you want to populate the new Country Club Membership record with the Account's city from the City field of the Account object.

To enable this, create an AfterCreate trigger on the AccountCountryClub intersection object with the following code:

```
setAttribute('AccountCity_c',Account_AccToCountryClub_Src?.PrimaryAddressCity
```

- AccountCity_c is the target field on the Country Club Membership object that you want to populate with the value from the Account object's City field.

- Account in Account_AccToCountryClub_Src? indicates the source object from which to pull the data.
- AccToCountryClub in Account_AccToCountryClub_Src? is the name of the M:M Account to Country Club Membership relationship.
- PrimaryAddressCity is the source data from the Account object that you use to populate the AccountCity_c target field on the Country Club Membership object.

Subtabs Based on Self-Referencing Relationships

In the case of self-referencing object relationships, both 1:M and M:M, it's not possible to access parent object data when creating records in custom subtabs. For example, it's possible to have the same object (such as Trouble Ticket) as both the source as well as the target of a 1:M or M:M relationship. However, when creating new records in a subtab for such a relationship, parent record data can't be accessed.

Mashups

Overview of Mashups

A mashup is a window into an independent external application. The mashup content changes are driven by direct interactions with the application and are preserved when its parent object pages are updated. Use mashups to integrate any web application with your own application.

With a mashup, your users can view and use an external web application without having to leave your application.

Note: A mashup displays a web application that presents information to end users, but can't display raw data returned from a web application endpoint.

The lifecycle of a mashup instance starts when it's launched in a new UI context, and ends when the mashup parent UI context is closed. Any changes made to the mashup as a result of interacting with it are preserved until the object UI is closed and then reopened. At runtime, you can leave a particular mashup instance by navigating to a different browser window or a different record within the application, and return to find the same mashup instance in the same state as you left it. You can have multiple mashup instances at the same time, and each will maintain its state independently.

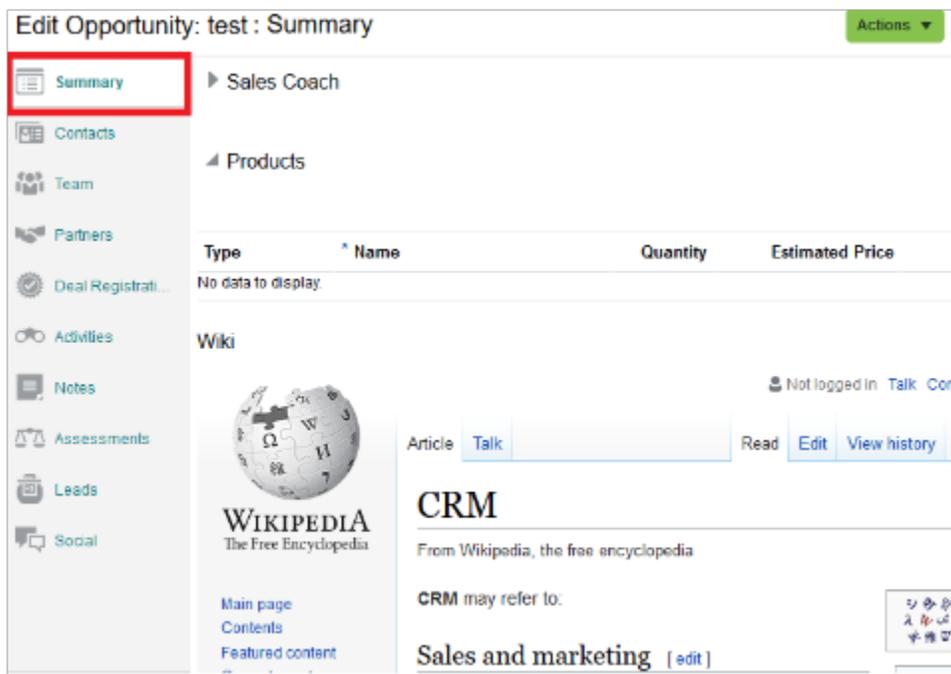
Here's how you create a mashup:

1. Register your web application in Application Composer.
2. Embed the registered web application into your application page as a new subtab or as part of the Summary subtab.

Here's a screenshot showing a sample Wikipedia article mashup, embedded as a new subtab:



Here's a screenshot showing a sample Wikipedia article mashup, embedded into the Summary subtab:



Related Topics

- [Register Your Web Application](#)
- [Embed a Registered Web Application into Your Application Page](#)

Register Your Web Application

Here are two ways you can register web applications in Application Composer:

- Use the **Parameter-based** mashup to enter the complete URL of the web application.
- Use the **Groovy expression** mashup to register only the domain or subdomain of the web application. You can define the path and add URL parameters later using a Groovy expression.

Note:

- Ensure that the web application you're registering is supported in iFrame, and doesn't have custom javascript that prevents embedding.
- Ensure that the web application you're registering doesn't have X-Frame-Options: SAMEORIGIN set in the response header. This option prevents the browser from displaying web applications that aren't hosted in the same domain as the parent page.
- If the protocol of your application page is HTTP, then use HTTP while registering the web application. If the protocol of your application page is HTTPS, then use HTTPS while registering the web application.

Use URL Parameters to Register a Web Application

1. In Application Composer's Common Setup menu, or on the Overview page, click **Mashup Content**.
2. On the Web Applications page, click **Register Web Application**.
3. On the Register Web Application page, in the **Name** field, enter a name for the web application.

This name appears in the Mashup catalog while embedding.

4. Select **Parameter-based**, if not already selected.
5. In the **URL Definition** field, enter the URL of the web application.

Register Web Application Save and Close Cancel

* Name

* Type Parameter-based Groovy expression

* URL Definition ?param1=value¶m2=value

URL Parameters Add

Specify the URL parameters for this web application mashup content. When you insert the mashup content into a page, you can specify the values for each URL parameter.

Parameter
<input type="text" value="Param1"/> ×
<input type="text" value="Param2"/> ×

- In the URL Parameters section, click **Add** to specify the parameters for the web application.

Note: Your application displays the URL parameters that you add here at the time of embedding the web application into an actual page, where you can specify the page values for each URL parameter.

- To delete a parameter, click the delete icon against it.
- Click **Save and Close**.

Use Groovy Expressions to Register a Web Application

- In Application Composer's Common Setup menu, or on the Overview page, click **Mashup Content**.
- On the Web Applications page, click **Register Web Application**.
- On the Register Web Application page, in the **Name** field, enter a name for the web application.
This name appears in the Mashup catalog while embedding.
- Select **Groovy expression**.
- In the **URL Definition** field, enter the base URL of the web application.

For example: `https://en.wikipedia.org/wiki/CRM`.

Note: You can define the path and any URL parameters later, using a Groovy Editor while embedding the web application.

Register Web Application Save and Close Cancel

* Name

* Type Parameter-based Groovy expression

URL Definition <Groovy expression return value>

- Click **Save and Close**.

The Web Applications page displays the newly registered web application. Here, you can edit the registered applications by changing their URL definition. You can also choose to set a web application as active or inactive. Note that while embedding applications, the mashup catalog only displays active web applications.

Optionally, you can enable your embedded application to call CRM APIs using JSON Web Tokens (JWT) by including `window.postMessage()` in your mashup content. See [Enable Your Embedded Application to Call CRM APIs](#) for detailed information.

Edit Registered Web Applications

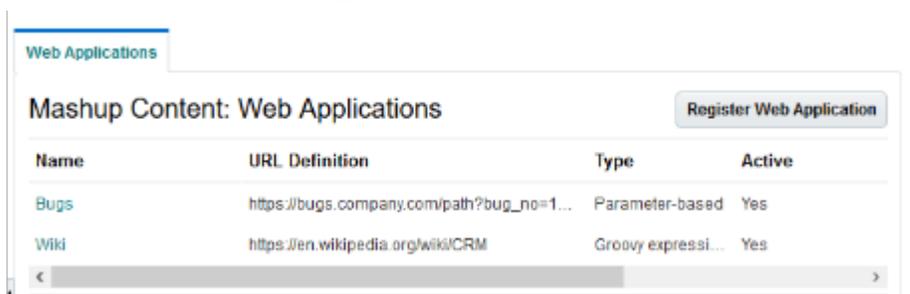
You can edit a registered web application to do the following:

- Change the web application's URL definition.
- Activate or deactivate the web application.

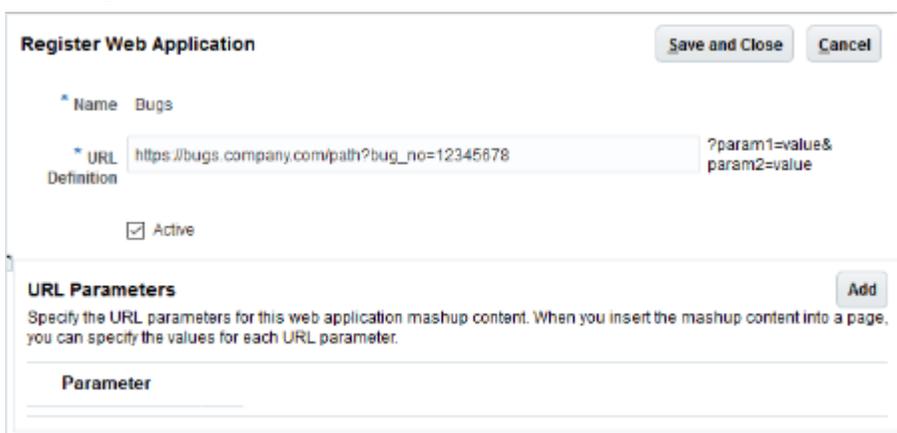
Note that any changes you make to a web application will reflect across all the application pages that it's currently embedded into.

To edit a registered web application:

1. In Application Composer's Common Setup menu, or on the Overview page, click **Mashup Content**.
2. On the Web Applications page, click the name of the web application that you want to edit.



3. To change the URL, edit the value in the **URL Definition** field.



4. To activate the web application, select the **Active** check box; to deactivate it, clear the check box.

Note: Only active web applications can be embedded into an application page.

5. Click **Save and Close**.

Embed a Registered Web Application into Your Application Page

You can choose to embed a registered web application into your own application as a new subtab or as part of the Summary subtab.

Embed a Web Application as a New Subtab

1. Click the Create Subtab icon from the Pages Overview page or from the details page layout.
2. Select **Mashup content**.
3. On the Select Mashup Content page, select the web application that you want to embed.
4. Click **Insert**.
5. On the Create Subtab page, in the **Display Label** field, enter a name for the new subtab.
6. A default Display Icon appears. You can change the icon by clicking **Change Icon**.
7. Enter the URL information depending on the type of mashup:
 - o For a parameter-based mashup, enter the parameter values in the Add Parameters section. These parameters are appended to the web application's URL as key-value pairs.
 - o For a groovy expression mashup, edit the base URL in the Edit Script section.
8. Click **Next**.
9. On the Additional Layouts page, if more than one details page layout exists, click **Next** to optionally select other details page layouts that will display this subtab.
10. Click **Save and Close**.

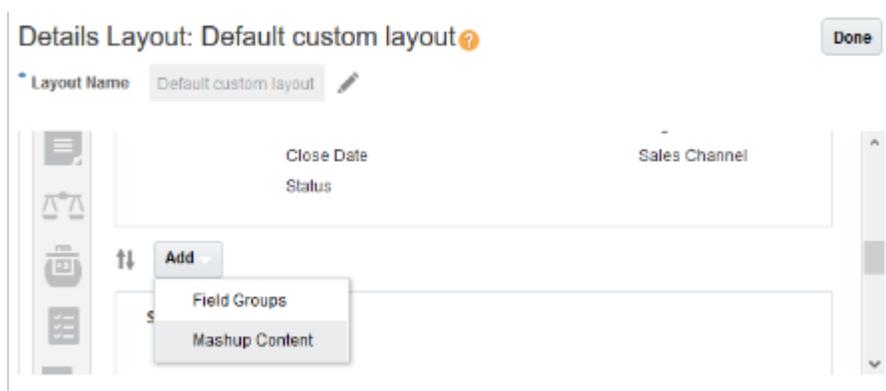
The new subtab with the web application appears as the last subtab on the Edit page.

Here's a screenshot showing a sample Wikipedia article mashup:



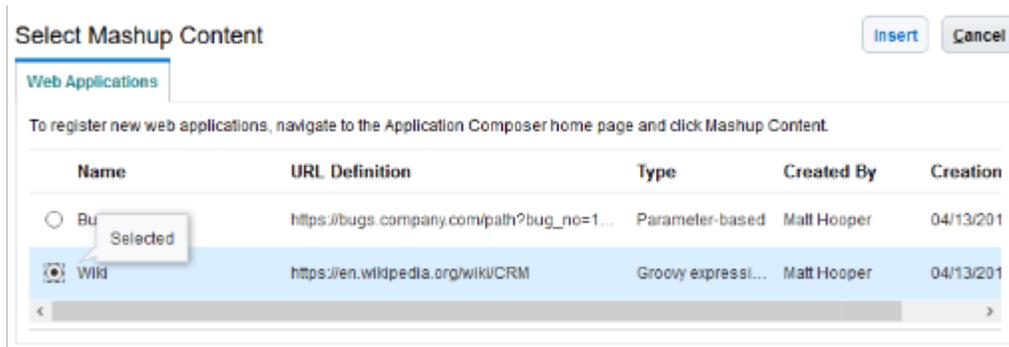
Embed a Web Application into the Summary Tab

1. In Application Composer's object navigation tree, expand the object.
2. Click the **Pages** node.
3. On the Application Pages tab, under Details Page Layouts, duplicate the standard layout to create a new layout to edit, or edit another existing layout.
4. Click **Add** and select **Mashup Content**.



5. On the Select Mashup Content page, select the web application that you want to embed.

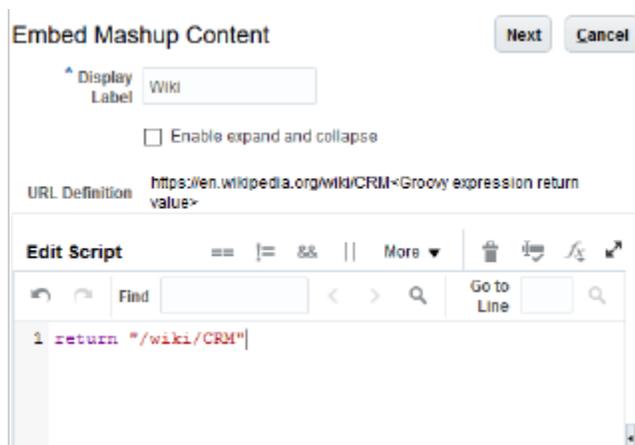
6. Click **Insert**.



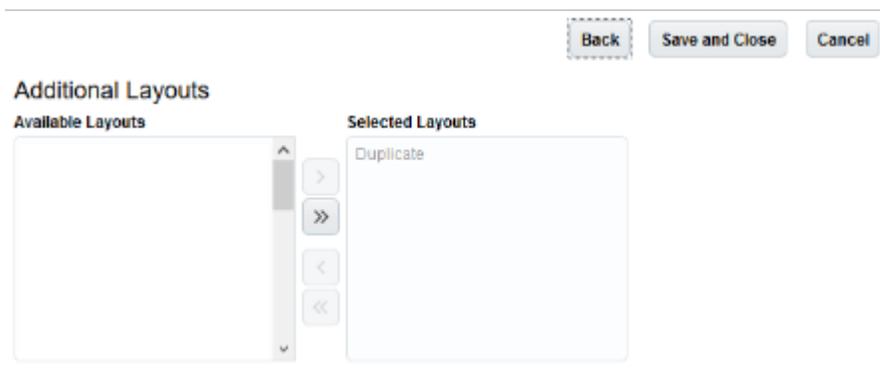
7. To allow your users to expand and collapse the embedded web application, select **Enable expand and collapse**.

8. Enter the URL information depending on the type of mashup:

- o For a parameter-based mashup, enter the parameter values in the Add Parameters section. These parameters are appended to the web application's URL as key-value pairs.
- o For a groovy expression mashup, edit the base URL in the Edit Script section. For example: return /wiki/CRM", which opens the Wikipedia's CRM page on your application page.



9. Click **Next**.

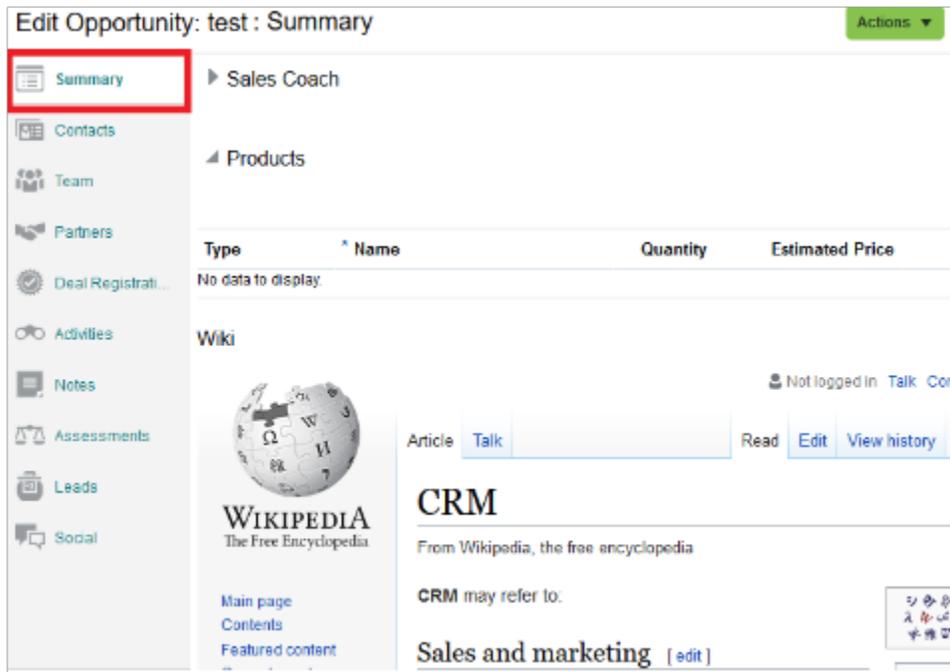


10. On the Additional Layouts page, select any other layout that you want to embed the web application into by moving it from the **Available Layouts** list to the **Selected Layouts** list.

11. Click **Save and Close**.

The embedded mashup appears at the end of the Summary subtab.

Here's a screenshot showing a sample Wikipedia article mashup.



Optionally, you can enable your embedded application to call CRM APIs using JSON Web Tokens (JWT) by including `window.postMessage()` in your mashup content. See [Enable Your Embedded Application to Call CRM APIs](#) for detailed information.

Related Topics

- [Subtab Types](#)

Enable Your Embedded Application to Call CRM APIs

You can enable your embedded application to call CRM APIs using JSON Web Tokens (JWT) by including the following Javascript in your mashup content (the code of your application):

```
<script type="text/javascript">
function init()
{
if (typeof window.addEventListener === 'function')
{
window.addEventListener('message', receiveJWTToken, false);
}
else if (typeof window.attachEvent == 'function')
{
window.attachEvent('onmessage', receiveJWTToken);
}
else
```

```
{
  throw new Error("Browser doesn't support addEventListener or attachEvent");
}
payload = {}
payload.origin = window.frames.origin;
payload.methos = 'requestJwtToken'
window.parent.postMessage(JSON.stringify(payload), '*');
}
function receiveJWTToken(token)
{
  //consume token.data;
}
init();
</script>
```

The application uses `window.postMessage()` to first get the JWT Access Token. The script creates an event listener to listen to `postMessage()`, then requests a token. You can edit the error message in the `init()` function.

Replace the logic in the `receiveJWTToken()` function to process the JWT Access Token provided by Oracle in a way that it meets your embedded application's requirements.

Enable Your Embedded Visual Builder Application to Call CRM APIs

If you embed a Visual Builder application, then make the changes described below to enable your application to call CRM APIs using JSON Web Tokens (JWT).

After making these changes, your application can then use the JWT token passed to it via the iframe messaging API from Application Composer. This token eliminates the need to perform a redirect to IDCS to establish a security cookie, as well as the need to call the server to retrieve the current user's information.

To pass a JWT token to an embedded VB application, you must:

1. Disable the service worker.
2. Configure your VB application for embedding.
3. Configure services for direct connection to FA applications.
4. Add the security provider files to your application.
5. Register the new security provider.

Note that you can do this provided that your application:

- Makes REST calls against FA APIs, only
- Doesn't request anything that requires the IDCS security token
- Doesn't use permissions or roles for functional security logic within the browser application

1. Disable Service Worker

Add the following code to the end of your application's `index.html`, right before the closing body tag.

```
<script type="text/javascript">
  vbInitConfig['SERVICE_WORKER_CONFIG'] = { disabled: true };
</script>
```

Note: This is optional but recommended in order to reduce runtime overhead.

2. Configure Your Application for Embedding

To allow your application to be embedded as an iframe, see "Embed a Web App in an Oracle Cloud Application" in the Oracle Cloud Developing Applications with Oracle Visual Builder guide.

3. Configure Services for Direct Connection to FA

To support direct connections, Visual Builder service instances must use HTTPS. Ensure that the FA back end is the same server URL that's serving the embedded content. Also ensure that FA services don't go through the VB proxy and instead make a direct connection against the FA back end.

Select one of the connection authentication options below.

- No defined authentication
- Propagate Current User Identity
- Direct (Bypass Authentication Proxy)
- Oracle Cloud Account, Token relay enabled
- User Assertion OAuth 2.0, Token relay enabled
- Client Credentials OAuth 2.0, Token relay enabled
- Resource Owner OAuth 2.0, Token relay enabled

For more information, see "Configure Connection and Authentication Types for Service Connections" in the Oracle Cloud Developing Applications with Oracle Visual Builder guide.

4. Add Security Provider Files

Copy and paste the JavaScript below into two .js files, and add them to the /webApps/{name}/resources/js directory.

Copy and paste this content into a file called `FusionEmbeddedWithNoUserSecurityProvider.js`.

```
define(['vb/types/securityProvider'],
  (SecurityProvider) => {
    class FusionEmbeddedWithNoUserSecurityProvider extends SecurityProvider {
      constructor() {
        super();
      }

      /**
       * Install our fetch handler to inject the JWT token from FA.
       */
      getServiceWorkerPlugins(config, isAnonymous) {
        return new Promise(r => r(['resources/js/InjectFATokenFetchPlugin']));
      }

      /**
       * Dummy method to return a user without any information, roles, or permissions. Used to skip the
       * traditional call to _currentusers for performance reasons
       */
      fetchCurrentUserRaw(config) {
        return new Promise((r) => {
          r({ response: {
            ok: true
          }},
            body: {
              userId: 'anonymous',
              username: 'anonymous',
              longId: 'anonymous',
              fullName: 'anonymous',
            }
          );
        });
      }
    }
  })
```

```
email: 'anonymous',
roles: [],
permissions: [],
isAdmin: false,
isAuthenticated: true,
isAnonymous: false
}
});
});
}
}

return FusionEmbeddedWithNoUserSecurityProvider;
});
```

Next, copy and paste this content into a file called `InjectFATokenFetchPlugin.js`.

```
define(['vbsw/api/fetchHandlerPlugin'], function (FetchHandlerPlugin) {
  'use strict';

  class InjectFATokenFetchPlugin extends FetchHandlerPlugin {
    constructor(context) {
      super(context);
    }

    // when initialized, start the request for the JWT token from the parent iframe
    this.waitForTokenInit = new Promise((resolve) => {
      const isEmbedded = window.location !== window.parent.location;

      if (isEmbedded) {
        window.addEventListener('message', event => {
          if (!event.origin.startsWith(window.frames.origin)) {
            resolve({
              host: event.origin,
              token: event.data
            });
          }
          else {
            return;
          }
        });
      }

      const payload = {};
      payload.origin = window.frames.origin;
      payload.method = 'requestJwtToken';
      window.parent.postMessage(JSON.stringify(payload), '*');
    });

    // TODO: if we are not in an embedded iframe (ran directly from VB) we will use the following credentials
    // below. This must be changed to match the environment and current JWT token (using postman to retrieve).
    else {
      resolve({
        host: 'https://cccn.fa.em2.oraclecloud.com',
        token: "xxx ... xxx"
      });
    }
  });

  handleRequestHook(request) {
    return this.waitForTokenInit.then((tokenInfo) => {
      if (request.url.indexOf(tokenInfo.host) === 0) {
        const headers = new Headers();
        request.headers.forEach((function(val, key) {
          if (!key.startsWith('vb-')) {
            headers.append(key, val);
          }
        }));
      }
    });
  }
});
```

```
};  
});  
headers.append('Authorization', `Bearer ${tokenInfo.token}`);  
const alteredRequest = new Request(request, {headers});  
return alteredRequest;  
}  
return request;  
});  
}  
}  
  
return InjectFATokenFetchPlugin;  
});
```

5. Register Security Provider

Change the userConfig property in the /webApps/{name}/app-flow.json file to the following:

```
"userConfig": {  
  "type": "resources/js/FusionEmbeddedWithNoUserSecurityProvider",  
  "configuration": {},  
  "embedding": "any"  
}
```

View Mashup Content

To view the content of an embedded web application:

1. From the home page or the navigator menu, select the application page that contains the embedded web application.
2. On the landing page, click the name of an object.

For example, if you have selected Opportunities, then click the name of an opportunity from the list.

Depending on where the web application is embedded, the mashup content appears either in the Summary tab or as a new subtab.

The following examples illustrate both the scenarios:

- A sample Wikipedia article embedded into the Summary tab of an application page:



- A sample Wikipedia article embedded into an application page as a new subtab:



Modify Work Area Lists

Create Work Area Lists for Others in the Organization

While all users can create personal lists (saved searches) in the different work areas, administrators can create lists for the whole organization or for select job roles.

To create a list for others, you must create the list using the Page Composer tool in a sandbox. Your changes become effective after you publish the sandbox.

Note: If you're creating lists for a specific job role, then you must first provision yourself with that job role so you can test your work before publishing the sandbox. See the steps outlined in the related topic Assign Yourself an Additional Job Role.

Before You Start

1. Create and enter a sandbox with Page Composer as the tool. Other tools don't enable making changes at the job role level, so creating a separate sandbox is a good idea if you're configuring lists for a specific job role.
 - a. Navigate to **Configuration > Sandboxes**.
 - b. Click **Create Sandbox**.
 - c. In the Click Create Sandbox page, enter a name and select **Page Composer** as the tool.
 - d. If you're making changes for a specific job role, then specify the job role for Page Composer:
 - i. Click **Edit** in the **Support Context** column for Page Composer.
 - ii. In the Edit Sandbox Context window, select the job role.
 - iii. Click **OK**.
 - e. Click **Create and Enter**.

The Sandbox toolbar appears at the top of the page.
2. Navigate to the work area you want to modify.
3. Open Page Composer by selecting the tool from the **Tools** menu in the sandbox bar at the top of the page.

Create the List

With both the sandbox and the Page Composer toolbars displayed on the top of the page, follow these steps to create your list. You create a new list by editing an existing list and saving it under a new name.

1. In the work area landing pad, click **Show Advanced Search** next to the **List** field.
2. From the Advanced Search panel **Saved Search** list, select a saved search to use as the starting point for creating a new one.

Tip: To create a list using only one field, including administrator-created fields, select a search with either Name or Close Date in the title. For opportunities and leads, select the **Close Date** saved search. For all other objects, select the name saved search, for example, the **Account Name** saved search or the **Contact Name** saved search.

3. Make your changes. You can:

- o Select a different record set to change the scope of your search. For example, selecting **My territory hierarchy** searches all the records in your territories and their subordinate territories. The available record sets vary from object to object.

To improve saved search performance, restrict your saved searches to smaller record sets. For example, rather than searching all the records you can see, search all the records in your territory hierarchy. Or restrict your searches to a smaller geographical area. For example, search all the accounts in one state instead of the whole country.

- o Add additional fields by clicking **Add**.

Note: If you're adding an administrator-created field to your search, then the field must be indexed for best search performance.

- o Select different operators for the fields in your search. While most of the operators, which differ field by field, are self-explanatory, here are some notes on the more complex:

- **Between**

Selecting the Between operator for a date field, prompts you to enter a specific date range for the search.

- Rolling-time operators, including **Yesterday, Today, Tomorrow, Last Week, This Week, Next Week, Last Month, This Month,** and **Next Month**

Searching for opportunities created this month, for example, returns opportunities created in the current calendar month. The rolling-time filters use the time zone of the signed-in user for the calculation.

- **Current User**

In some fields, including the Last Updated By and Created By fields, you can create a list that displays only the records relevant to each signed-in user. For example, when you create a list of all opportunities last updated by the current user, all of your users can view the opportunities they recently updated.

- **Is Blank, Is Not Blank, and Does Not Equal**

Selecting the Is Blank operator, makes it possible to search for records missing values in a text field or a field validated by a list of values, for example. Administrators can make these operators available on fields that aren't mandatory, by setting two system profile options. The Enable Additional Search Operators for Text Fields (ZCA_ENABLE_ADDITIONAL_TEXT_OPERATORS) profile option enables the Is Blank and Is Not Blank operators in text field searches. The Enable Additional Search Operators for List of Values (ZCA_ENABLE_ADDITIONAL_LOV_OPERATORS) enables the Is Blank, Is Not Blank, and Does Not Equal operators on fields validated by list of values (both fixed choice and dynamic choice list fields).

- o Specify which attributes you want to display in the search results table by selecting **Columns** from the **View** menu.

You can select specific columns or display all columns.

- o Reorder the filter conditions by clicking **Reorder**.

- o Delete any fields you added to the search.

You can't delete the fields provided by Oracle. You can only delete fields you added.

4. You can test your search by clicking **Search**.

5. When you're satisfied with the results, click **Save**.

The Create Saved Search window appears.

6. Enter a new name for the saved search.

You can't reuse the names of the saved searches provided by Oracle with the application.

7. Make sure the **Run Automatically** option remains selected. Selecting this option runs the query each time you select the list in the UI.

8. If you want users to see the list generated by this search when they navigate to the work area, then select the **Set as Default** option.

Note: Making a search the default doesn't override any default searches individual users may have created for their own use. Each user can personalize lists provided by administrators.

9. Click **OK** to return to the work area landing page.

Save Your Work and Publish the Sandbox

1. When you're done making your changes, you must save them by clicking **Close** in the Page Composer toolbar.

2. Publish the sandbox by clicking the sandbox name and selecting **Publish**.

Related Topics

- [Assign Yourself Additional Job Roles Required for Testing](#)
- [What gets saved when I create a saved search for searches with multiple criteria?](#)
- [Overview of Sandboxes](#)
- [Create and Activate Sandboxes](#)
- [Publish Sandboxes](#)

Remove a List by Deleting or Hiding the Underlying Saved Search



You can remove a list from use either by hiding or deleting the underlying saved search. Saved searches are the saved search criteria that generate the list. You can't delete saved searches provided by Oracle; you can only hide them.

Before You Start

1. Create and enter a sandbox with Page Composer as the tool. Other tools don't enable making changes at the job role level, so creating a separate sandbox is a good idea if you're configuring lists for a specific job role.
 - a. Navigate to **Configuration > Sandboxes**.
 - b. Click **Create Sandbox**.
 - c. In the Click Create Sandbox page, enter a name and select **Page Composer** as the tool.
 - d. If you're making changes for a specific job role, then specify the job role for Page Composer:
 - i. Click **Edit** in the **Support Context** column for Page Composer.
 - ii. In the Edit Sandbox Context window, select the job role.
 - iii. Click **OK**.
 - e. Click **Create and Enter**.
The Sandbox toolbar appears at the top of the page.
2. Navigate to the work area you want to modify.
3. Open Page Composer by selecting the tool from the **Tools** menu in the sandbox bar at the top of the page.

Hide a Saved Search

Here's how to hide a saved search. This is the only way of removing a list supplied by Oracle from selection in the work area. If you want to hide a saved search that's designated as the default search, then you must first designate a different list as the default.

1. In the work area landing pad, click **Show Advanced Search** next to the **List** field to open the Advanced Search panel.
2. From the **Saved Search** list, select a saved search different from the one you want to hide.
3. Now select **Personalize**.
4. In the personalize Saved Searches window, select the saved search you want to hide.
5. Deselect the **Show in Search List** option.

6. Click **OK**.

A saved search you hide remains available for future use but doesn't display in the work area. You can restore the saved search to the list in the future by selecting the **Show in Search List** option.

Delete a Saved Search

Here's how to delete saved searches. You can't delete those provided by Oracle.

Note: If you delete a saved search that's specified as the default, users won't see any lists displayed on the page when they navigate to the work area.

1. In the work area landing pad, click **Show Advanced Search** next to the **List** field to open the Advanced Search panel.
2. From the **Saved Search** list, select a saved search you want to delete.
3. Now select **Personalize**.
In the Personalize Saved Searches window, the **Saved Searches** list displays the name of the saved search you're about to delete.
4. Click **Delete**.
5. Click **OK**.

Save Your Work and Publish the Sandbox

1. When you're done making your changes, you must save them by clicking **Close** in the Page Composer toolbar.
2. Publish the sandbox by clicking the sandbox name and selecting **Publish**.

Related Topics

- [Overview of Sandboxes](#)
- [Publish Sandboxes](#)
- [Create and Activate Sandboxes](#)

FAQs for Extending Application Pages

What's the difference in upgrade behavior between standard layouts and duplicated layouts?

After an upgrade, the standard layouts for an object's creation and details pages automatically include any new underlying changes made as part of the upgrade.

However, duplicated layouts (copied versions of standard layouts) exist independently from the original, standard layout, and in general, do not uptake most underlying changes that come with an upgrade. In other words, duplicated layouts remain functionally identical to their preupgrade versions, even after an upgrade.

Thus, after every upgrade, carefully review all your duplicated layouts. You can re-create them if you want to pick up new changes introduced in the upgrade.

How can I change what columns are displayed for users in the work area?

The columns of information displayed in work area landing pages are tied to the underlying list. If you want to change what information displays for users in the landing pages, you must edit the list.

You can't edit the lists supplied by Oracle, but you can make a copy of the list, make your changes, and then hide the list supplied by Oracle.

How can I reset all personalization changes made by a specific end user?

Your end users can personalize a page explicitly (like creating a saved search) or implicitly (like changing table column width). You can revert all such changes made by a specific user using the Personalization page in Application Composer.

In the Common Setup region, click **Personalization**. Search for a user and then click **Reset Personalization**. All changes made by that user are removed and all pages are restored to the default content and layout as originally delivered. Changes implemented by an administrator using Application Composer and Page Composer are retained, if they exist.

End users can also revert their changes themselves, but only on a page-by-page basis.

Related Topics

- [How can I reset a page or task flow to a previously saved version?](#)

5 Groovy Scripts

Overview of Groovy Scripts

Groovy is a standard, dynamic scripting language for the Java platform. You write Groovy scripts using Application Composer's expression builder, which appears in many places as you modify existing objects or create new custom ones.

Read this chapter to learn about how and where you can use Groovy scripting in Application Composer.

Note: To fully understand all the scripting features available to you in Application Composer, you should also review the Groovy Scripting Reference guide.

In this chapter, you will learn about:

- Where you can use Groovy in your application, along with examples of one or more lines of Groovy code
- How to access view objects using the `newView()` function, for programmatic access to object data
- How to create global functions, which is code that multiple objects can share
- How to call Web services from your Groovy scripts. You might call a Web service for access to internal or external data, or, for example, to perform a calculation on your data.
- What kind of scripts will you write?

Write Groovy Scripts

You write Groovy scripts using Application Composer's expression builder, which appears in many places throughout Application Composer as you modify existing objects or create new custom ones.

- You will write shorter scripts to provide an expression to calculate a custom formula field's value or to calculate a custom field's default value, for example.
- You may write somewhat longer scripts to define a field-level validation rule or an object-level validation rule, for example.

Additional examples of where you write Groovy scripts in Application Composer are described in "Groovy Scripting: Explained."

To learn more about how to best use the features available in the expression builder when writing scripts, see "Groovy Tips and Techniques" in the Groovy Scripting Reference guide.

Performance Considerations

For optimal performance, always write your Groovy scripts to query and fetch the fewest possible rows from the database. Minimizing the number of queried rows is one of the most important performance best practices you can follow.

Many of the most significant performance impacts you're likely to encounter will be related to code that queries and fetches a large number of rows. This has several implications for application performance and functionality:

- Response time

Querying and fetching rows are two of the most expensive operations your application will typically perform and can noticeably impact performance. Large result sets magnify the impact of sub-optimal code executed while iterating over a result set.

- Scalability

Long-running operations tie up limited shared resources such as database connections and shared memory. This might increase response time of other requests because they have to wait for resources to become available.

- Functionality

If your application attempts to fetch too many rows, then the application will display a warning to the user and will not fetch any more rows. This limits the user's ability to view data in the UI and causes processing errors in scripts that iterate over rows.

To limit the potential performance impacts described above, most view objects have a 500-row Fetch Limit. Design your customizations with the Fetch Limit in mind as an absolute limit, but always strive to limit your queries to the least number of rows possible to satisfy your business requirements.

For example, use view criteria and bind parameters to tailor a view's default query and limit fetch size for view objects. See [Finding Objects Using a View Criteria](#).

Related Topics

- [Groovy Scripting](#)
- [View Objects in Scripts](#)
- [Global Functions](#)
- [How to Call SOAP Web Services](#)

Groovy Scripting

Groovy is a standard, dynamic scripting language for the Java platform for which Application composer provides deep support. This topic provides an overview of where you can use Groovy in your application and gives some samples of one or more lines of Groovy code.

For more information on Groovy scripting, see the Groovy Scripting Reference guide.

Note: Read "Supported Classes and Methods for Use in Groovy Scripts", which documents the only classes and methods you may use in your Groovy scripts. Using any other class or method will raise a security violation error when you migrate your code to later maintenance releases. Therefore, it is strongly suggested that the Groovy code you write uses only the classes and methods shown there to avoid the time-consuming task of having to rewrite your code in the future.

Groovy Scripting Terminology Explained

Throughout the document the term script is used to describe one or more lines of Groovy code that the Oracle ADF framework executes at run time. Often a very-short script is all that is required.

For example, to validate that a Commission Percentage field's value does not exceed 40%, you might use a one-line script like:

```
return CommissionPercentage < 0.40
```

In fact, this one-liner can be conveniently shortened by dropping the return keyword since the return keyword is always implied on the last line of a script:

```
CommissionPercentage < 0.40
```

For slightly more complicated logic, your script might require some conditional handling. For example, suppose the maximum commission percentage is 40% if the salesperson's job grade is less than or equal to 3, but 60% if the job grade is higher. Your script would grow a little to look like this:

```
if (JobGrade <= 3) {  
    return CommissionPercentage < 0.40  
}  
else {  
    return CommissionPercentage < 0.60  
}
```

Scripts that you write for other purposes like complex validation rules or reusable functions may span multiple pages, depending on your needs.

When a context requiring a Groovy script will typically use a short (often, one-line) script, that fact is emphasized by calling it an expression. Technically the terms script and expression are interchangeable. Anywhere you can provide a one-line expression is also a valid context for providing a multi-line script if the need arises. Whether you provide a short expression or a multi-line script, the syntax and features at your disposal are the same. You need only pay attention that your code returns a value of the appropriate type for the context in which you use it.

The Groovy Scripting: Examples topic includes all the return types. This topic highlights the expected return type for each script example.

Using Groovy Scripts in Your Application

There are a number of different contexts where you will use Groovy scripts as you modify existing objects or create new custom ones.

You will write shorter scripts to provide an expression to:

- Calculate a custom formula field's value
- Calculate a custom field's default value
- Make a custom field conditionally updatable, or
- Make a custom field conditionally required
- Define the condition for executing an object workflow

You will generally write somewhat longer scripts to define:

- A field-level validation rule
- An object-level validation rule
- A trigger to complement default processing
- Utility code in a global function, or
- Reusable behavior in an object function

If you anticipate calling the same code from multiple different contexts, any of your scripts can call the reusable code you write in either global functions or object functions. As their name implies, global functions can be called from scripts in any object or from other global functions. Object functions can be called by any scripts in the same object, or even triggered by a button in the user interface.

After exploring the Groovy basic techniques needed to understand the examples documented in the Groovy Scripting Reference guide, see "Groovy Scripting: Examples" for a concrete example of each of these usages. Also see "Groovy Tips and Techniques" in the Groovy Scripting Reference guide for getting the most out of Groovy in your application.

Related Topics

- [Classes and Methods Supported in Groovy Scripts](#)
- [Groovy Scripting Examples](#)

Server Scripts

Application Composer supports Groovy as the scripting language that you use to enhance your applications. There are many different contexts in which you can use Groovy scripts.

This topic illustrates the use of validation rules, triggers, and object functions, which you can define using the Server Scripts node for any standard or custom object. For a more detailed explanation of Groovy scripting using Application Composer, see the Groovy Scripting Reference guide.

The server scripts that you can define for any standard or custom object include the following:

- **Validation rules**
Write a script to validate either a field or an object.
- **Triggers**
Write trigger scripts to automatically execute an action whenever a specific trigger event occurs.
- **Object functions**
Write a function that can be reused in multiple contexts. For example, you can reuse an object function inside a trigger or validation rule.

Tip: Server scripts are executed synchronously, which could potentially impact performance if the scripts are long-running or performance-intensive. Before implementing such logic, always consider whether the same logic can be executed in an asynchronous way. For example, you can instead use an object workflow with a Groovy Script action.

Validation Rules

Validation rules are constraints that you can define on either a field or on an object. Write an expression or a longer script to validate a field or object before it can be saved. Define validation rules using the Server Scripts node for any standard or custom object.

- If your requirement involves a single field, use field-level validation.
A field-level validation rule is a constraint you can define on any standard or custom field. The rule is evaluated at runtime whenever the corresponding field's value is set and the user tabs out of the field. When the rule

executes, the field's value hasn't been assigned yet and your rule acts as a gatekeeper until its successful assignment. Covering the validation logic at the field level reduces overhead of Groovy executions when saving. Use this option whenever possible.

Note: Dynamic choice lists don't support field-level validation rules.

For example, consider a custom TroubleTicket object with a Priority field. You can set a field-level validation rule to validate that the number entered is between 1 and 5.

The expression (or longer script) you write must return a Boolean value that indicates whether the value is valid.

- If the rule returns true, then the field assignment will succeed so long as all other field-level rules on the same field also return true.
- If the rule returns false, then this prevents the field assignment from occurring. Also, the invalid field is visually highlighted in the UI, and the configured error message is displayed to the end user. Because the assignment fails in this situation, the field retains its current value (possibly null, if the value was null before). However, the UI component in the web page allows the users to see and correct their invalid entry to try again.

See "Defining a Field-Level Validation Rule" in the Groovy Scripting Reference guide.

- When validation is needed across multiple related fields in an object, use object-level validation rules. This is a constraint you can define on any standard or custom object.

Use object-level rules to enforce conditions that depend on two or more fields in the object. This ensures that regardless of the order in which the user assigns the values, the rule will be consistently enforced. The rule is evaluated whenever the framework attempts to validate the object. This validation can occur, for example, when submitting changes in a Web form, when navigating from one row to another, as well as when changes to an object are saved. (Rules aren't evaluated if the user saves a record without making changes.)

For example, consider a TroubleTicket object with Priority and AssignedTo fields, where the latter is a dynamic choice list field referencing Contact objects whose Type field is a Staff Member. You can set an object-level validation rule to validate that a trouble ticket of priority 1 or 2 can't be saved without being assigned to a staff member.

The expression (or longer script) you write must return a Boolean value that indicates whether the object is valid:

- If the rule returns true, then the object validation will succeed so long as all other object-level rules on the same object return true.
- If the rule returns false, then this result prevents the object from being saved, and the configured error message is displayed to the end user.

See "Defining an Object-Level Validation Rule" in the Groovy Scripting Reference guide.

Note: When defining a validation rule, don't implement business logic other than:

- Validations inside an object
- Field validation rule Groovy script

Triggers

Triggers are scripts that you can write to complement the default processing logic for a standard or custom object. When a specific event occurs, triggers automatically execute an action that you specify in the trigger definition. You can define triggers both at the object level and at the field level using the Server Scripts node for any standard or custom object. Define object triggers to extend standard processing logic, such as record creation, updates, and deletions.

When you define a trigger, you select the specific event that causes your script to automatically run. This specific event is also referred to as a trigger. Oracle supplies a set number of these trigger events that you can pick from when defining your trigger "scripts."

Choose the correct triggering point when defining your trigger:

- Field-level triggers are scripts that you write to execute an action in response to a change in another field's value. When you define a trigger at the field level, you select the **After Field Changed** trigger and the field that this trigger is watching. You then define the action that you want to happen when the field's value changes.

The **After Field Changed** trigger calculates other derived field values when the value of the field that you specify changes. Don't use a field-level validation rule to achieve this purpose because while your field-level validation rule may succeed, other field-level validation rules may fail and stop the field's value from actually being changed. Generally, because you want your field-change derivation logic to run only when the field's value changes, the After Field Changed trigger guarantees that you get this behavior.

See "Defining a Field-Level Trigger to React to Value Changes" in the Groovy Scripting Reference guide.

- Similarly, object-level triggers are scripts that execute an action when a specific event occurs. In the case of object-level triggers, you have many more trigger "events" to pick from, such as:
 - **After Create**
Fires when a new object record is created. Commonly used to set default values for fields.
 - **Before Invalidate**
Fires on the parent object when one of its child object records is created, updated, or deleted. For building in relationship logic.
 - **Before Remove**
Fires when an attempt is made to delete an object record. Can be used to create conditions that prevent deletes.
 - **Before Insert in Database**
Fires before a new object is inserted into the database. Can be used to ensure a dependent record exists or check for duplicates.
 - **Before Update in Database**
Fires before an existing object is modified in the database. Could be used to check dependent record values.
 - **Before Delete in Database**
Fires before an existing object is deleted from the database. Could be used to check dependent record values.
 - **Before Rollback in Database**
 - **After Changes Posted to Database**
Fires after all changes have been posted to the database, but before they're permanently committed. Could be used to make additional changes that will be saved as part of the current transaction.

For example, consider a Contact object with an OpenTroubleTickets field that needs to be updated any time a trouble ticket is created. You can create a trigger on the TroubleTicket object using the **After Changes Posted to the Database** trigger event. When an event occurs, your trigger can automatically update the OpenTroubleTickets field with a new count.

For a complete list of the trigger "events" that you can pick from, see "Defining an Object-Level Trigger to Complement Default Processing" in the Groovy Scripting Reference guide.

For optimal performance, follow these guidelines when using triggers:

- Don't exceed 10 triggers per object.
- Combine logically-related actions inside a single trigger. A single trigger produces better performance than multiple triggers.
- When defining a trigger, choose the correct triggering point.
- Avoid using validation logic inside triggers. Instead, use validation rules for any validation logic.
- Before using the `newView()` API, check to see if related objects or related collection accessors already exist.

If a relationship already exists, then don't use `newView()` to query an object. This avoids the firing of additional queries.

- When querying objects programmatically, select an efficient view criteria so that the underlying query is limited.

Object Functions and Global Functions

You can write reusable code as either an object function or global function. Use a function if you anticipate calling the same code from multiple different contexts. Object functions can be called by any script in the same object, or even triggered by a button in the user interface. Global functions can be called from scripts in any object or from other global functions.

- Object functions are useful for code that encapsulates business logic specific to a given object. You can call object functions by name from any other script related to the same object. In addition, you can call them using a button or link in the user interface.

The supported return types and optional parameter types are the same as for global functions. For a list of the most common types for function return values and parameters, see "Defining Utility Code in a Global Function" in the Groovy Scripting Reference guide.

See also "Defining Reusable Behavior with an Object Function" in the Groovy Scripting Reference guide.

- Global functions are useful for code that multiple objects want to share. Write user-defined functions using Groovy scripts, which can be referenced in all Groovy script editors throughout Application Composer. For example, you could create two global functions to define standard helper routines to log the start of a block of Groovy script and to log a diagnostic message.

To call a global function, preface the function name with the `adf.util.` prefix. When defining a function, you specify a return value and can optionally specify one or more typed parameters that the caller will be required to pass when invoked.

For a list of the most common types for function return values and parameters, see "Defining Utility Code in a Global Function" in the Groovy Scripting Reference guide.

- For optimal performance, the maximum number of records that can be fetched is restricted to 500 records. If the data size chosen causes the job to fail with `ExprTimeoutException`, reduce the data size using `setMaxFetchSize`. Or, use selective filtering to filter the number of records fetched through the view object to less than 500 records.

Privileged Functions

When you define either an object function or global function, the function might run on an object where the runtime user has no privileges to create or update records. Allow users without access to an object's data to run a function with full access, by doing two things:

1. While defining the function, check the Privileged check box to indicate that the function is privileged.
2. Confirm that the Privileged Script Administration role has the right level of access so that the object function can execute successfully. Access to objects isn't given automatically. Instead, you must grant access using the Application Composer security UI.

At runtime, when a user invokes a privileged function from the UI, a temporary login session is activated with the privileged role, Privileged Script Administration (`ORA_CRM_EXTN_PRIVILEGE_SCRIPT_ROLE`). This privileged role has access to the object in your function, so no permission issues exist. The temporary login session lasts only for the duration of the single function call or anything that function calls internally.

For example, a sales representative has access to opportunity records only, not account records. When a sales representative edits an opportunity, there is a button that updates a related account using a privileged Groovy

script. Even though the sales representative doesn't have update privileges for the account object, when the sales representative clicks the button, the privileged Groovy script executes by switching to the privileged role context to complete the update to the account record.

To make this happen, use the Application Composer security UI to grant account access to the privileged role, Privileged Script Administration.

Related Topics

- [Global Functions](#)
- [How Object Workflows and Groovy Scripts Work Together](#)

Object Functions

Overview of Object Functions

Object functions are useful for code that encapsulates business logic specific to a given object. You can call object functions by name from any other script related to the same object.

You can also call them using a button or link in the user interface. Object functions can be reused in multiple contexts. For example, you can reuse an object function inside a trigger or validation rule. You can also use an object function asynchronously by using a scheduled process, or by using it inside an object workflow's Groovy script action.

Let's say you want to make an asynchronous call to an external web service so that you don't have to wait for a long-running web service to complete. You can:

- Call the web service using an object function at a scheduled time using the Scheduled Processes tool. You can process a set of records on a daily or weekly basis, asynchronously, when users don't need to see immediate feedback in the user interface.

See the Using Object Functions for Scheduled Processes section in this topic.

- Call the web service using an object function from an object workflow's Groovy script action. Object workflows are also executed asynchronously and provide additional functionality, such as letting you set the conditions for when the workflow is triggered.

See How Object Workflows and Groovy Scripts Work Together in the Related Topics.

Note: You can't update records by calling object functions using the `setAttribute()` through the RESTful service due to the lack of transaction control. Instead, create a new custom field and call the object function from its field trigger, and then update this field using REST.

You can optionally define object functions to return values. You can also specify typed parameters, which the caller will be required to provide values for, when the function is invoked. The supported return types and optional parameter types are the same as for global functions. However, scheduled processes only support object functions that have no parameters and that have `String` as the return type, if the function is defined to return values. So, if you're defining an object function to be used in a scheduled process, ensure that it has no parameters defined, and if you're defining the function to return a value, ensure that its return type is `String`.

For more information on object functions, see the Groovy Scripting Reference guide.

Related Topics

- [Server Scripts](#)
- [How Object Workflows and Groovy Scripts Work Together](#)
- [Define Object Functions](#)
- [Best Practice for Scheduling Object Functions: Test Capacity Runs](#)
- [Use Object Functions for Scheduled Processes](#)
- [Submit Scheduled Processes and Process Sets](#)

Define Object Functions

Use the following general procedure to create object functions:

Note: To define object functions and then schedule them, your role must include the `ZCX_MANAGE_EXTENSIBLE_OBJECT_PRIV` privilege.

1. Sign in as an administrator.
2. Ensure that you're in an active sandbox.
3. Navigate to **Application Composer**.
4. Expand the object for which you want to create a function, and then click **Server Scripts**.
5. In the Server Scripts window, click the **Object Functions** tab.
6. Click the **Add a new Object Function** icon.
The Create Object Function page appears.
7. In the **Function Name** field, enter a name for the object function. Ensure that the name has no spaces.
8. If you want the function to return a value, in the **Returns** drop-down list, select a return type. Otherwise, retain the value as **void**, which is selected by default.
The supported return types are the same as for global functions. See the Groovy Scripting Reference guide for more information.
Note: If you're creating an object function to be used in a scheduled process and if you're defining the function to return values, you must select **String** as the return type.
9. To change the visibility of the function from the default, use the **Visibility** drop-down list. See the Controlling the Visibility of an Object Function section in the Groovy Scripting Reference guide for more information.
10. Optionally, to add parameters, in the **Parameters** section, click the **Add Parameter** icon and enter the name and type of the first parameter.
The supported parameter types are the same as for global functions. See the Groovy Scripting Reference guide for more information.
Note: Don't define any parameters if you're creating an object function to be used in a scheduled process. Object functions with input parameters aren't supported in scheduled processes.
11. Repeat the previous step for any remaining parameters.
12. Enter the Groovy code in the **Edit Script** window.
13. Click **Validate** to validate the function.
14. Click **Save and Close**.

Related Topics

- [Overview of Object Functions](#)
- [Use Object Functions for Scheduled Processes](#)
- [Best Practice for Scheduling Object Functions: Test Capacity Runs](#)

Use Object Functions for Scheduled Processes

Scheduled processes are batch jobs that capture data and allow objects to act on that data.

Use the Scheduled Processes tool to run jobs to manipulate a set of records for a specific business need, or to get printable output with information about certain records. Schedule business logic code and update a set of records periodically, on a daily, weekly, or monthly basis, asynchronously, by just specifying the object's name and its function.

Note: To schedule object functions, your role must include the ZCX_MANAGE_EXTENSIBLE_OBJECT_PRIV privilege.

You can use scheduled processes in several scenarios. Here are a few examples of when you may need to schedule processes:

- To perform mass update of records based on certain criteria.
- To trigger workflows based on criteria that will be met in the future, as a workaround for time-based workflows.
- To run custom logic with heavy updates that can be scheduled during off-hours.

The following procedure describes how to create a scheduled process for a Groovy object function.

Before you begin scheduling the object function, it's recommended that you perform the following steps:

- Record the API name of the object as the object name.
- Record the object function name.
- Ensure that the object function compiles and runs properly using validation or trigger invocation mechanism within Application Composer, where runtime message debugging is supported.
- Test capacity runs to determine the best batch size to handle large data sets. See [Best Practice for Scheduling Object Functions: Test Capacity Runs](#).

The above steps are best practices to ensure that there's no impact on existing operations or the data in your production environment.

To create a scheduled process for a Groovy object function:

1. Click **Navigator**, and then select **Scheduled Processes**.
2. In the Scheduled Processes work area, click **Schedule New Process**.
3. Leaving the type as Job, in the **Name** drop-down list, search and select **Schedule Custom Groovy Object Functions**, and then click **OK**.

4. On the Process Details page, do the following:
 - a. In the **Object Name** field, enter the object's API name. For example, OpportunityVO.

Note: The object name you provide here must be the same as the object's API name on Application Composer's Object Overview page. Also, if you have any jobs scheduled from a previous release you must change the object name to match the object's API name on Application Composer's Object Overview page.

- b. In the **Object Function** field, enter the object function name. For example, OpptyMsg.
 - c. Do one of the following:
 - Click the Advanced button to define the schedule, document output, and notifications.
 - Submit with the default schedule.

See the Submitting Scheduled Processes and Process Sets section in the Oracle Applications Cloud Using Common Features guide for detailed information.

A few things to note:

- You can write a function as complex as needed.
- You can write a function to return values in a **.txt** file.

The **.txt** file displays any values that the Groovy code returns. To view the **.txt** file of a process, click the status link for that process.

- Each scheduled process has a runtime limit of 30 minutes to run an object function. This is to prevent long-running jobs that can consume large amounts of resources to support object functions operating on large volumes of data.

Note: The 30 minute limit only applies to validation and execution of the object function operations specified for that run. As long as the required database transaction handling has begun within the allocated 30 minutes, the completion of the scheduled process can span past the 30 minute limit. This enables database inserts and updates of large data sets that often take a long time to complete their operation. Hence, the end-to-end processing time can be longer than 30 minutes.

If your Groovy script triggers a long-running process that's implemented via ESS/PLSQL, any Groovy triggers already created on the target object won't get called. This is because ESS/PLSQL operates directly at the database level, bypassing ADF and the model layer.

- Each job run constitutes a transaction. This means that all the operations specified by the Groovy script in the object function have to run to completion to complete the transaction. Otherwise, the transaction is rolled back and no records are updated.
- Once the job execution starts, the **Cancel** button won't work and you can't cancel the job.
- If a record is updated, the **Last Updated By** value changes to the user who submitted the job, and the **Last Updated Date** value changes to the date and time of the update.
- The log file displays any errors occurred during the process. To view the log file of a process, click the status link for that process.

For more information about scheduled processes and how to use them, refer to your product's implementation guide.

Troubleshoot Scheduled Processes for Groovy Object Functions

Processing a lot of object rows in a scheduled Groovy object function at the same time may require some debugging.

Here's how you can troubleshoot such scheduled processes. To help with troubleshooting:

1. Define an object function to return a string (or `StringBuilder` constituting the string). Populate information like the number of rows processed and failed, or the exceptions that were thrown. This value will be printed on the job output, which can be used for debugging.
2. Add the `println` function in object functions and enable runtime messages to help determine the exact cause for failure.

See [Runtime Messages](#) for more information about the `println` function.
3. However, the runtime messages of scheduled Groovy object functions don't display the `println` messages. So, as a workaround, you can create a custom button for the object function with the `println` function, and then invoke that button from the object's detail page.

See [Actions and Links](#) for information on creating buttons.

To view the log file and job output:

1. Click **Navigator > Tools > Scheduled Processes**.
2. Find your scheduled process and select it.

You can refresh the search results to see the latest status.

3. In the Log and Output section, click the attachment link to download the console log file.

Note: If the object function was defined to have no return values, only the console log file is displayed and is downloaded when you click the link. Otherwise, an Attachment dialog opens with the console log file and the console text file.

4. Review the log and job output.

Related Topics

- [Overview of Object Functions](#)
- [Define Object Functions](#)
- [Best Practice for Scheduling Object Functions: Test Capacity Runs](#)

Best Practice for Scheduling Object Functions: Test Capacity Runs

Due to the 30 minute limit on the object function execution, it's best to break up the work done in each object function execution. This can be best done by testing a series of capacity runs in a test environment.

Using the same object function, each job execution can work off a subset of the total data to be processed. By scheduling the same job to run at regular time intervals, the full set of data can be processed over multiple job runs.

The tests can be done in the following number of steps:

1. Determine how much data can be processed in a 30 minute time interval.

The size of data to be processed in each run is controlled by the Groovy script in the object function. You can control the following factors using Groovy script changes in your object function:

- View criteria definition

- o Data filter defined on the result set returned from the view criteria definition

Note: Database enhancements can be used to improve search effectiveness on view criteria and data filter results. For example, defining additional indexes on database columns search can return results much faster than a full table scan.

- o Maximum fetch size specified by `setMaxFetchSize`, which normally defaults to 500 rows if not defined.

If the data size chosen causes the job to fail with `ExprTimeoutException`, reduce the data size using `setMaxFetchSize`.

The amount of data that can be processed within a 30 minute limit's dependent on the complexity of the data, type of operation on that data, and the resultant database update operation complexity, all of which are subjected to traffic and resource contentions. Data complexity is a reflection of the attributes making up the object to be operated on, and the parent, child, and associated object structures that need to be traversed or updated to complete the operation. The more complex the data, the more costly it's to collect and manipulate the information needed to complete the operation.

For example, a simple custom object with very few attributes and no related parent, child, associated objects incurs the least cost whereas an out-of-the-box standard object such as Opportunity, with large number of attributes and multiple child and associated object relationships (such as account, contact, and lead) incurs a higher search and update cost.

In terms of type of operations, insert operations incur the highest cost, followed by updates, followed by reads, with the cost increasing as amount of data processed increases. For update operations, the amount of data that can be processed depends on the cost of search operation and cost of updates, with the former dictated by the view criteria and latter by the operations performed on the result set obtained from the search, as specified by the Groovy script in the object function. Database update operation is dependent on the type of operation specified in the object function. Reads don't incur any database update cost whereas insert and update operations do, increasing with the data complexity.

2. Determine how long each job takes to complete.

Besides data size, the type of database operations needed to support the object function can greatly affect the completion of the job. Database data creation, updates, and deletes (when permitted by business rules) normally take a longer time to complete than data reads. The amount of time needed to complete database updates and creates transactions can far exceed that of the object function execution time. This cost is particularly high for cases where object data complexity is high. These costs are often seen in job completion times that far exceed the 30 minute limit. The time interval between the scheduled time and completion time of the job is the time taken to complete the job.

3. Schedule each job to run slightly over the job completion time.

To ensure that the previous job is completed, it's best to add a few extra minutes to the job completion time obtained in step 2 while scheduling each job.

Related Topics

- [Overview of Object Functions](#)
- [Define Object Functions](#)
- [Use Object Functions for Scheduled Processes](#)

Global Functions

Global functions are useful for code that multiple objects want to share. You use global functions when you write Groovy scripts using the expression builder in Application Composer.

Some global functions are delivered with your cloud service, ready for your use. Or, you can define new global functions.

This topic:

- Provides a list of some predefined global functions that are provided in Application Composer.
- Explains how to define new global functions.

For more information on global functions, see the Groovy Scripting Reference guide. See "Defining Utility Code in a Global Function" for details on using global functions in your scripts.

Examples of Predefined Global Functions

This table lists the global functions that are provided for use in Application Composer.

Note: These global functions are not available for selection in the expression builder. Instead, to use these functions, manually type the function name into your script, using the full package name as described in the below table.

Global Function	Description
encodeURL()	Returns a translated URL-encoded format string using the specified encoding scheme. To use this function, use its full package definition: <code>oracle.apps.crmCommon.extensibility.content.adf.data.context.DynamicURLUtil.urlEncode(encodingStyle)</code>
getCurrencyRate()	Returns the currency conversion rate from one currency to another on the specified date. The first argument is the From currency code. The second argument is the To currency code. The third argument is the conversion rate date. To use this function, use its full package definition: <code>oracle.apps.crmCommon.extensibility.content.adf.data.util.ExtensibilityGroovyUtil.get</code>
getEndDate()	To use this function, use its full package definition: <code>oracle.apps.crm.service.svcMgmt.fwk.model.util.SvcPublicDueDateutil.getEndDate(startT duration, durationUnit, scheduleId)</code>
getObjectURL()	Returns a URL of the classic object detail page for a particular row. To use this function, use its full package definition: <code>oracle.apps.crmCommon.extensibility.content.context.ObjectURLGroovyWrapper.getObjectURL(primaryKey)</code>
getResourceUserPartyId()	Global method to get the partyId for a given username.

Global Function	Description
	To use this function, use its full package definition: <code>oracle.apps.cdm.foundation.publicModel.common.util.HzGroovyUtil.getResourceUserPartyI</code>
<code>getSession()</code>	Note: The usage of this global function is deprecated.
<code>getTrustToken()</code>	Returns the Oracle Platform Security Services trust token. The token is valid for 4 hours. To use this function, use its full package definition: <code>oracle.apps.fnd.applcore.common.SecuredTokenBean().getTrustToken()</code>
<code>getUserPartnerCompanyId()</code>	Returns the partner company's party_ID for the logged-in user, if the user is a partner user. To use this function, use its full package definition: <code>oracle.apps.cdm.foundation.publicModel.common.util.HzSessionUtil.getUserPartnerCompan</code>
<code>getUserPartyId()</code>	Returns the logged-in user's Party_ID. To use this function, use its full package definition: <code>oracle.apps.cdm.foundation.publicModel.common.util.HzSessionUtil.getUserPartyId()</code> See <i>Classes and Methods Supported in Groovy Scripts</i> .
<code>getUserPrimaryBU()</code>	Returns the logged-in user's primary business unit. To use this function, use its full package definition: <code>oracle.apps.cdm.foundation.publicModel.common.util.HzSessionUtil.getUserPrimaryBU()</code>
<code>getUserProfile()</code>	Note: The usage of this global function is deprecated. Instead, use the SecurityContext global function to get the user profile.
<code>getUserRootResourceOrgId()</code>	Returns the organization_ID for the logged-in user's organization hierarchy root resource organization. To use this function, use its full package definition: <code>oracle.apps.cdm.foundation.publicModel.common.util.HzSessionUtil.getUserRootResourceO</code>
<code>omnichannelCancelWork()</code>	To use this function, use its full package definition: <code>oracle.apps.crm.service.omnichannel.service.ExtnOmnichannelUtil.cancelWork(context, objectName, objectId, reason)</code>
<code>omnichannelPropertyChange()</code>	To use this function, use its full package definition: <code>oracle.apps.crm.service.omnichannel.service.ExtnOmnichannelUtil.propertyChange(conte objectName, objectId, oldValues)</code>
<code>sendNotification1()</code>	Sends a notification to all enabled delivery methods. The delivery methods, message text, and recipients can be configured on the notification preferences page. If notification preferences for the

Global Function	Description
	<p>trigger exist, then the values in the preferences take precedence. The first argument is the ADF object. The second argument is the messageText. The third argument is the partyId.</p> <p>To use this function, use its full package definition: <code>oracle.apps.crm.service.notifications.util.NotificationUtil.sendNotificationFromGroovy(context.source, context.newValue, context.oldValue, messageText, recipientPartyId)</code></p>
sendNotificationSimple1()	<p>Invoke from an object function Groovy script to send a notification to the Bell channel. The first argument is the ADF object. The second argument is the messageText. The third argument is the partyId.</p> <p>To use this function, use its full package definition: <code>oracle.apps.crm.service.notifications.util.NotificationUtil.sendNotificationSimpleFromGroovy(context.source, messageText, recipientPartyId)</code></p>
sendNotificationSimple2()	<p>To use this function, use its full package definition: <code>oracle.apps.crm.service.notifications.util.NotificationUtil.sendNotificationSimpleFromGroovy(context.source, requestParameters)</code></p>
serviceRegion()	<p>To use this function, use its full package definition: <code>oracle.apps.crm.service.svcMgmt.srMgmt.util.SvcUtil.getServiceRegion(partyId)</code></p>

Defining Global Functions

To define a global function:

1. In Application Composer, navigate to the Common Setup pane, which displays in the regional area.
2. Click **Global Functions**.

Note: You must be in an active sandbox.

3. On the Global Functions page, click **New**.
4. Specify the global function name and a return value.
5. Enter a description and example of the global function.
6. Optionally check the Privileged check box.

If this function might run on an object where the runtime user has no privileges to create or update records, then check this box. Users without access to an object's data can still run this function at runtime, with full access. See "Server Scripts: Explained" for more details on privileged scripts.

7. Optionally specify one or more typed parameters that the caller will be required to pass in, when the function is invoked.
8. Specify the body of the function.
9. Validate and save your function.

Related Topics

- [Server Scripts](#)

View Objects in Scripts

A view object is an Oracle ADF component that simplifies querying and working with business object rows. You access view objects when you write Groovy scripts using the expression builder in Application Composer.

To access view objects in your scripts, use the `newView()` function for an object API name. The `newView()` function accesses a custom or standard view object, and creates a new view object instance that programmatically accesses that business object's rows. For example, a common task that you will do with this new view object instance is to query some data. Do this by calling the `findByKey()` function on the view object to find a row by key.

For more information on accessing view objects, see "Accessing the View Object for Programmatic Access to Business Objects" in the Groovy Scripting Reference guide.

This topic:

- Explains why the `newView()` function is useful in your scripts.
- Explains how to access view objects, either custom or standard, from the expression builder using the `newView()` function.
- Provides a list of the standard view objects that are provided in Application Composer along with a code sample.
- Provides some best practices for optimal performance.

`newView()` Function

When you write Groovy scripts in Application Composer, you're usually in the context of a specific record from a specific object. For example, you can write a trigger script with a single line `setAttribute('Name','Acme Widgets Inc.')` and the script will be executed on the user's current record.

The `newView()` function, by contrast, lets you construct a new reference to an object which doesn't require any contextual relationship to the current record. For example, the line `def myVO = newView('OpportunityVO')` produces an instance of the Opportunity view object that your script can query and read, and then add, delete, or update rows.

Accessing View Objects

To access view objects, use the `newView()` function for an object API name from within the expression builder in Application Composer:

1. Navigate to the expression builder from Application Composer.

There are several ways to start the expression builder in Application Composer. For example, start the expression builder when editing a field to make it required.

2. In the expression builder palette, on the Functions tab, select the Other category and the `newView()` function.
3. Click Insert.

A window displays that lists the view objects you can call in your script.

The objects don't have to be related to the current object to appear in this list.

Examples of Standard View Objects

The standard objects that are delivered with your application provide view objects for use in your scripts. The previous section described how to access those view objects. This section provides some examples of standard view objects that are provided in Application Composer, and how you might use them in your scripts. Attributes that you would typically script against are also included.

For objects that aren't extensible and thus not available in Application Composer, see the SOAP Web Services documentation for your cloud service to view a list of attributes that you can script against.

Standard View Object	Description	Typical Attributes
Address	<p>Use this object to access the address for a given party in scripting, if the current object doesn't have a view link to the Address object.</p> <p>Access this Address extensible object as a child of the Account, Contact, or Household objects.</p>	Refer to the Address object in Application Composer, and review the descriptions provided for all attributes.
CodeAssignment	<p>Use this object to access classifications assigned to a given party in scripting, if the current object doesn't have a view link to this object.</p> <p>Access this object as a child of the Account or Contact objects.</p>	Refer to the Trading Community Classification Code Assignment in the SOAP Web Services documentation for your cloud service.
CommonLookup	Access application common lookups in scripting.	LookupType, LookupCode, Tag, EnabledFlag, StartDateActive, EndDateActive, Meaning, Description
Contact	<p>Use this object to access customer contact information in scripting, if the current object doesn't have a view link to this object.</p> <p>Access this Customer Contact Profile extensible object as a child of the Account, Contact, or Household objects.</p>	Refer to the Customer Contact Profile object in Application Composer, and review the descriptions provided for all attributes.
FndTreeVersion	<p>Use this object in scripting to access tree versions.</p> <p>The customer hierarchy and party hierarchy are modeled as trees.</p>	TreeStructureCode, TreeCode, TreeVersionID, Status, EffectiveStartDate, EffectiveEndDate, TreeVersionName
FndTreeNode	<p>Use this object to determine the parent/child relationships for a given hierarchy.</p> <p>The hierarchy for a given version is stored in this object.</p>	TreeStructureCode, TreeCode, TreeVersionID, TreeNodeID, ParentTreeNodeID, Depth, ChildCount, ParentPk1Value
FndTreeNodeRf	Use this object in scripting to easily access the flattened version of the given hierarchy version.	TreeStructureCode, TreeCode, TreeVersionID, TreeNodeID, Pk1Value, AncestorPk1Value, Distance, IsLeaf

Standard View Object	Description	Typical Attributes
Location	Use this object to update or create physical location fields. Address is the intersection of location and party. Address fields like city, state, and country are stored in the location. These fields are made available in the Address object for read-only purposes. Use this object if you need write access to location fields in scripting.	Refer to the Trading Community Location SDO in the SOAP Web Services documentation for your cloud service.
OrganizationParty	Use this object to get the organization party and all of its children when you have the organization PartyID in your script, and you don't have a view link from the current object to the Account object.	Refer to the Trading Community Organization Details in the SOAP Web Services documentation for your cloud service.
OrganizationProfile	Access this Account extensible object as a child of an OrganizationParty row or directly get the profile if you have a PartyID.	Refer to the Account object in Application Composer, and review the descriptions provided for all attributes.
OriginalSystemReference	Use this object to get the ID for given TCA object based on the source system and source system reference information.	Refer to the Trading Community Original System Reference in the SOAP Web Services documentation for your cloud service.
PersonParty	Use this object to get the Person Party and all of its children when you have the person PartyID in your script, and you don't have a view link from the current object to Account object.	Refer to the Trading Community Person Details in the SOAP Web Services documentation for your cloud service.
PersonProfile	Access this Contact extensible object as a child of a PersonParty row or directly get the profile if you have a PartyID.	Refer to the Contact object in Application Composer, and review the descriptions provided for all attributes.
Relationship	Use this object in scripting if you have a RelationshipID on the current object and that object doesn't have a view link to this object. Access this Relationship extensible object as a child of the Account, Contact, or Household objects.	Refer to the Relationship object in Application Composer, and review the descriptions provided for all attributes.
Resource	Use this Resource extensible object in scripting to get the resource object details if you have a user or resource PartyID, and the current object ID doesn't expose a view link to this object.	Refer to the Trading Community Resource Profile in the SOAP Web Services documentation for your cloud service.

Example of Retrieving the Resource PartyID

In this example, you populate a custom dynamic choice list field that points to the Resource object. The dynamic choice list field displays on the Notes details page.

On the Note object, create either an After Create or Before Insert trigger, depending on your requirements:

```
def partyID = oracle.apps.cdm.foundation.publicModel.common.util.HzSessionUtil.getUserPartyId()
def view_Resource = newView('Resource')
def view_Criteria = newViewCriteria(view_Resource)
def view_criteria_row = view_Criteria.createRow()
def view_condition = view_criteria_row.ensureCriteriaItem('PartyId')
view_condition.setOperator('=')
view_condition.setValue(partyID)
view_Criteria.insertRow(view_criteria_row)
view_Resource.appendViewCriteria(view_Criteria)
view_Resource.setMaxFetchSize(1)
view_Resource.executeQuery()

// Once found get access to the record attributes
def var_record = view_Resource.first()
// Printing the Resource Party Id in Runtime messages
println ("The value for Resource Party Id is "+ var_record.ResourceProfileId)
```

Performance Considerations

As a best practice, avoid unnecessary `newView` calls because the `newView()` function is relatively expensive. Every time the function is executed, a view object instance is created, reserved for programmatic use. This means that you must explicitly or implicitly execute a query and fetch rows for any view that you use in a Groovy script.

Some additional recommendations related to `newView` and loops:

- There's a limit on the number of concurrent view instances that you can create across all scripts while executing Groovy. If you exceed the maximum, then the application triggers an exception.

Avoid calling `newView` unconditionally inside of a loop so that you don't have to worry about exceeding the maximum.

- If you need to re-execute a view inside a loop to retrieve different subsets of rows, then do the following:
 - Call `newView` once outside the loop.
 - Inside the loop, update the applied view criteria and re-execute the existing view.

See *Finding Objects Using a View Criteria*.

- If you need to iterate over the same result set repeatedly inside of a loop, then call the `reset()` method at the beginning of each iteration to set the view object's current row back before the first row.

If you're repeatedly retrieving the same attribute for each row, an even better approach is to retrieve the value from each row during the first iteration and then store the value in a Map object using the row's key as the Map key (use the `getKey()` method to get the row's key).

For subsequent iterations, retrieve the value from the Map object using the row's key. This approach allows you to avoid calling the `getAttribute` method repeatedly.

- If you need to call `newView` conditionally inside of a loop, then see *Performance Best Practices for Using Loops, Methods and Strings*.

Related Topics

- [Oracle Fusion Cloud Customer Experience SOAP Web Services for Sales and Fusion Service](#)

Classes and Methods Supported in Groovy Scripts

Groovy is a standard, dynamic scripting language for the Java platform for which Application Composer provides support. This topic covers the supported classes and methods for use in Groovy scripts.

Classes and Methods

When writing Groovy scripts, you may only use the classes and methods that are documented in the following table. Using any other class or method may work initially, but will throw a run time exception when you migrate your code to later versions. Therefore, we strongly suggest that you ensure the Groovy code you write adheres to the classes and methods shown here.

For each class, in addition to the method names listed in the table, the following method names are also allowed:

- equals()
- hashCode()
- toString()

In contrast, the following methods are never allowed on any object:

- finalize()
- getClass()
- getMetaClass()
- notify()
- notifyAll()
- wait()

Note: The following supported classes and methods will expand over time depending on customer requirements and business need. Thus, periodically review this table to assess what is newly supported in each release.

Class Name	Allowed Methods	Package
ADFContext	getLocale() getSecurityContext() getUserRoles() isUserInRole()	oracle.adf.share
Array	Any constructor Any method	java.sql
Array	getArray()	oracle.jbo.domain

Class Name	Allowed Methods	Package
	getElemType() getList()	
ArrayList	Any constructor Any method	java.util
Arrays	Any constructor Any method	java.util
AttributeDef	getAttributeKind() getIndex() getJavaType() getName() getPrecision() getProperty() getScale() getUIHelper() getUpdateableFlag() isMandatory() isQueryable()	oracle.jbo
AttributeHints	getControlType() getDisplayHeight() getDisplayHint() getDisplayWidth() getFormat() getFormattedAttribute() getFormatter() getFormatterClassName() getHint() getLocaleName() parseFormattedAttribute()	oracle.jbo
AttributeList	getAttribute()	oracle.jbo

Class Name	Allowed Methods	Package
	getAttributeIndexOf() getAttributeNames() setAttribute()	
BaseLobDomain	closeCharacterStream() closeInputStream() closeOutputStream() getInputStream() getLength() getOutputStream() getcharacterStream()	oracle.jbo.domain
BigDecimal	Any constructor Any method	java.math
BigInteger	Any constructor Any method	java.math
BitSet	Any constructor Any method	java.util
Blob	Any constructor Any method	java.sql
BlobDomain	Any constructor getBinaryOutputStream() getBinaryStream() getBufferSize()	oracle.jbo.domain
Boolean	Any constructor Any method	java.lang
Byte	Any constructor Any method	java.lang
Calendar	Any constructor	java.util

Class Name	Allowed Methods	Package
	Any method	
Char	Any constructor bigDecimalValue() bigIntegerValue() booleanValue() doubleValue() floatValue() getValue() intValue() longValue()	oracle.jbo.domain
Clob	Any constructor Any method	java.sql
ClobDomain	Any constructor toCharArray()	oracle.jbo.domain
Collection	Any constructor Any method	java.util
Collections	Any constructor Any method	java.util
Comparator	Any constructor Any method	java.util
Currency	Any constructor Any method	java.util
DBSequence	Any constructor getValue()	oracle.jbo.domain
Date	Any constructor Any method	java.util
Date	Any constructor	java.sql

Class Name	Allowed Methods	Package
	Any method	
Date	Any constructor compareTo() dateValue() getValue() stringValue() timeValue() timestampValue()	oracle.jbo.domain
Dictionary	Any constructor Any method	java.util
Double	Any constructor Any method	java.lang
Enum	Any constructor Any method	java.lang
EnumMap	Any constructor Any method	java.util
EnumSet	Any constructor Any method	java.util
Enumeration	Any constructor Any method	java.util
EventListener	Any constructor Any method	java.util
EventListenerProxy	Any constructor Any method	java.util
EventObject	Any constructor Any method	java.util

Class Name	Allowed Methods	Package
Exception	Any constructor Any method	java.lang
ExprValueErrorHandler	addAttribute() clearAttributes() raise() raiseLater() warn()	oracle.jbo
Float	Any constructor Any method	java.lang
Formattable	Any constructor Any method	java.util
FormattableFlags	Any constructor Any method	java.util
Formatter	Any constructor Any method	java.util
GregorianCalendar	Any constructor Any method	java.util
HashMap	Any constructor Any method	java.util
HashSet	Any constructor Any method	java.util
Hash table	Any constructor Any method	java.util
IdentityHashMap	Any constructor Any method	java.util
Integer	Any constructor	java.lang

Class Name	Allowed Methods	Package
	Any method	
Iterator	Any constructor Any method	java.util
JboException	getDetails() getErrorCode() getErrorParameters() getLocalizedMessage() getMessage() getProductCode() getProperty()	oracle.jbo
JboWarning	Any constructor getDetails() getErrorCode() getErrorParameters() getLocalizedMessage() getMessage() getProductCode() getProperty()	oracle.jbo
Key	toStringFormat()	oracle.jbo
LinkedHashMap	Any constructor Any method	java.util
LinkedHashSet	Any constructor Any method	java.util
LinkedList	Any constructor Any method	java.util
List	Any constructor Any method	java.util

Class Name	Allowed Methods	Package
ListIterator	Any constructor Any method	java.util
ListResourceBundle	Any constructor Any method	java.util
Locale	Any constructor Any method	java.util
Long	Any constructor Any method	java.lang
Map	Any constructor Any method	java.util
Math	Any constructor Any method	java.lang
MathContext	Any constructor Any method	java.math
NClob	Any constructor Any method	java.sql
NameValuePairs	Any constructor getAttribute() getAttributeIndexOf() getAttributeNames() setAttribute()	oracle.jbo
NativeTypeDomainInterface	getNativeObject()	oracle.jbo.domain
Number	Any constructor bigDecimalValue() bigIntegerValue() booleanValue()	oracle.jbo.domain

Class Name	Allowed Methods	Package
	byteValue() doubleValue() floatValue() getValue() intValue() longValue() shortValue()	
Observable	Any constructor Any method	java.util
Observer	Any constructor Any method	java.util
PriorityQueue	Any constructor Any method	java.util
Properties	Any constructor Any method	java.util
PropertyPermission	Any constructor Any method	java.util
PropertyResourceBundle	Any constructor Any method	java.util
Queue	Any constructor Any method	java.util
Random	Any constructor Any method	java.util
RandomAccess	Any constructor Any method	java.util
Ref	Any constructor Any method	java.sql

Class Name	Allowed Methods	Package
ResourceBundle	Any constructor Any method	java.util
Row	getAttribute() getAttributeHints() getKey() getLookupDescription() Note: This method is for a dynamic choice list field and the attribute name is the ID attribute of the dynamic choice list field, DCL FOREIGN KEY(ID_C). For example, a dynamic choice list field <code>Test_c</code> <code>getLookupDescription('Test_Id_c')</code> , which returns a dynamic choice list value. getOriginalAttributeValue() getPrimaryRowState() getSelectedListDisplayValue() getSelectedListDisplayValues() isAttributeChanged() isAttributeUpdateable() remove() revertRow() revertRowAndContainees() setAttribute() validate()	oracle.jbo
RowId	Any constructor Any method	java.sql
RowIterator	createAndInitRow() createRow() findByKey() findRowsMatchingCriteria() first()	oracle.jbo

Class Name	Allowed Methods	Package
	getAllRowsInRange() getCurrentRow() getEstimatedRowCount() hasNext() hasPrevious() insertRow() last() next() previous() reset()	
RowSet	avg() count() createAndInitRow() createRow() executeQuery() findByKey() findRowsMatchingCriteria() first() getAllRowsInRange() getCurrentRow() getEstimatedRowCount() hasNext() hasPrevious() insertRow() last() max() min() next() previous() reset() sum()	oracle.jbo

Class Name	Allowed Methods	Package
Scanner	Any constructor Any method	java.util
SecurityContext	getUserName() getUserProfile()	oracle.adf.share.security
Session	getLocale() getLocaleContext() getUserData()	oracle.jbo
Set	Any constructor Any method	java.util
Short	Any constructor Any method	java.lang
Short	Any constructor Any method	java.lang
SimpleTimeZone	Any constructor Any method	java.util
SortedMap	Any constructor Any method	java.util
SortedSet	Any constructor Any method	java.util
Stack	Any constructor Any method	java.util
StackTraceElement	Any constructor Any method	java.lang
StrictMath	Any constructor Any method	java.lang

Class Name	Allowed Methods	Package
String	Any constructor Any method	java.lang
StringBuffer	Any constructor Any method	java.lang
StringBuilder	Any constructor Any method	java.lang
StringTokenizer	Any constructor Any method	java.util
Struct	Any constructor Any method	java.sql
Struct	getAttribute() setAttribute()	oracle.jbo.domain
StructureDef	findAttributeDef() getAttributeIndexOf()	oracle.jbo
Time	Any constructor Any method	java.sql
TimeZone	Any constructor Any method	java.util
Timer	Any constructor Any method	java.util
TimerTask	Any constructor Any method	java.util
Time stamp	Any constructor Any method	java.sql
Time stamp	Any constructor	oracle.jbo.domain

Class Name	Allowed Methods	Package
	compareTo() dateValue() getValue() stringValue() timeValue() timestampValue()	
TreeMap	Any constructor Any method	java.util
TreeSet	Any constructor Any method	java.util
UUID	Any constructor Any method	java.util
UserProfile	getBusinessCity() getBusinessCountry() getBusinessEmail() getBusinessFax() getBusinessMobile() getBusinessPOBox() getBusinessPager() getBusinessPhone() getBusinessPostalAddr() getBusinessPostalCode() getBusinessState() getBusinessStreet() getDateofBirth() getDateofHire() getDefaultGroup() getDepartment() getDepartmentNumber() getDescription()	oracle.adf.share.security.identitymanagment Note: Some of these methods may return <code>null</code> if the corresponding attribute of the user record is not populated in the identity store or if the identity provider does not support those methods.

Class Name	Allowed Methods	Package
	getDisplayName() getEmployeeNumber() getEmployeeType() getFirstName() getGUID() getGivenName() getHomeAddress() getHomePhone() getInitials() getJpegPhoto() getLastName() getMaidenName() getManager() getMiddleName() getName() getNameSuffix() getOrganization() getOrganizationalUnit() getPreferredLanguage() getPrincipal() getProperties() getProperty() getTimeZone() getTitle() getUIAccessMode() getUniqueName() getUserID() .getUserName() getWirelessAcctNumber()	
UserProfile	getUserPartyId()	oracle.apps.cdm.foundation.publicModel.common.util.HzS

Class Name	Allowed Methods	Package
ValidationException	getDetails() getErrorCode() getErrorParameters() getLocalizedMessage() getMessage() getProductCode() getProperty()	oracle.jbo
Vector	Any constructor Any method	java.util
ViewCriteria	createAndInitRow() createRow() createViewCriteriaRow() findByKey() findRowsMatchingCriteria() first() getAllRowsInRange() getCurrentRow() getEstimatedRowCount() hasNext() hasPrevious() insertRow() last() next() previous() reset()	oracle.jbo
ViewCriteriaItem	getValue() makeCompound() setOperator() setUpperColumns() setValue()	oracle.jbo

Class Name	Allowed Methods	Package
ViewCriteriaItemCompound	ensureItem() getValue() makeCompound() setOperator() setUpperColumns() setValue()	oracle.jbo
ViewCriteriaRow	ensureCriteriaItem() getConjunction() isUpperColumns() setConjunction() setUpperColumns()	oracle.jbo
ViewObject	appendViewCriteria() avg() count() createAndInitRow() createRow() createViewCriteria() executeQuery() findByKey() findRowsMatchingCriteria() first() getAllRowsInRange() getCurrentRow() getEstimatedRowCount() getMaxFetchSize() hasNext() hasPrevious() insertRow() last() max()	oracle.jbo

Class Name	Allowed Methods	Package
	min() next() previous() reset() setMaxFetchSize() setSortBy() sum()	
WeakHashMap	Any constructor Any method	java.util

Related Topics

- [Groovy Scripting](#)
- [Groovy Scripting Examples](#)

Groovy Scripting Examples

This topic contains examples of application changes that you can perform using Groovy scripts.

Application Composer leverages Groovy to enable you to enhance your application changes. Groovy is a standard, dynamic scripting language for the Java platform for which the Application Composer provides support.

The following examples are covered in this topic:

- Using println function
- Making fields conditionally required
- Making fields conditionally updatable
- Adding validation to fields

Note: It is assumed that a custom object called Help Request exists and is available on the Navigator menu.

Using Println Function

In this example, you add a `println` function to a trigger to view an opportunity's Win Probability. Whenever the Win Probability field is updated, the `println` function in the trigger performs an update, and you can see the output in the Runtime Messages page within Application Composer.

To view an opportunity's Win Probability:

1. Navigate to Application Composer.
 2. Expand **Standard Objects** and then expand **Opportunity**.
 3. Under **Opportunity**, click **Server Scripts**.
 4. Under the Triggers tab, select **Action > Add** for Field Triggers.
 5. In the **Trigger** field, select **Before Update in Database**.
 6. In the **Trigger Name** field, enter **TestPrintIn**.
 7. Under the Trigger Definition region, enter the following script in the expression text box:

```
println("Before Update Trigger. The new value of the Win Probability is" + nvl(WinProb, "Win Probability was null")).
```

`nvl()` ensures that the variables are null-aware.
 8. Click the **Validate** icon. Confirmation appears when the script is parsed successfully.
 9. Click **OK**.
 10. In the **Navigator** menu, click **Opportunities**.
 11. Click **Create Opportunity** in the left pane.
 12. In the **Name** field, enter **Opportunity Trigger Test**.
 13. In the **Win Probability (%)** field, enter **50**.
 14. Click the **Save and Close** button.
 15. On the Overview page, select the Opportunities tab.
 16. Click the **Opportunity Trigger Test** link in the table.
 17. In the **Win Probability (%)** field, enter **25**.
 18. Click the **Save and Close** button.
 19. Navigate to Application Composer.
 20. On the Overview page, click **Runtime Messages**.
 21. Click the **Get Latest Log Messages** button.
 22. Locate the message that you wrote in your `println` function.
- Tip:** Sort the messages in descending order to locate quickly.

Making Fields Conditionally Required

In this example, you add the following two custom fields to the Help Request object. You make one of them conditionally required based on the value you select in the other field.

1. **Priority** field of type Fixed Choice List.
2. **Justification** field of type text that's conditionally set to required depending on the value in the **Priority** field. If the value in the Priority field is set to Urgent, then the Justification field appears as a mandatory or required field. Else, the Justification field remains optional.

To create fields in application composer and make one of them conditionally required:

1. Navigate to Application Composer.
2. Expand **Custom Objects**.
3. Expand **Help Request** and click **Fields**.
4. Under Custom tab, select **Action > Create**.
5. In the Select Field Type window, select **Text** and click **OK**.
6. On the Create Text Field page, enter **Justification** in the **Display Label**.
7. Click **Save and Close**.
8. On the Fields page, select **Action > Create** under the Custom tab.

9. In the Select Field Type window, select **Choice List (Fixed)** and click **OK**.
10. On Create Fixed Choice List page, enter **Priority** in the **Display Label**.
11. Under the List of Values region, click the **Create a New Lookup Type** icon.
12. In Create Lookup Type dialog, specify the following:

Field	Value
Meaning	Help Request Priority
Lookup Type	HR_PRIORITY

13. Select **Action > Create**, and specify the following in the first row:

Column	Value
Meaning	Urgent
Lookup Code	URG
Display Sequence	1

14. Select **Action > Create**, and specify the following in the second row:

Column	Value
Meaning	Important
Lookup Code	IMP
Display Sequence	2

15. Click **Save**.
16. On Create Fixed Choice List page, select the **Fixed Value** option in the Default Value region, and then select **Important** as the default value.
17. Click the **Save and Close** button.
Fields page opens.
18. Under the Custom tab, click the **Justification** link.
Edit Text Field: Justification page opens.
19. Under Constraints region, select **Required** check box and click the expression builder icon next to it.
Expression builder dialog opens.

20. Enter the following into the (script) text box:

```
try { if(nvl(Priority_c,"") == "URGENT") { return true } else { return false } } catch(e)
{ println("Error with the Required property of the Justification field in the Help Request object") }
```

21. Click the **OK** button.
22. Under Constraints region, select **Priority** from the **Depends On** list.
23. Click the **Save and Close** button.
24. Under Help Request in the left pane, click **Pages**.
Help Request: Pages page opens.
25. Click the **Edit Summary Form** link.
Edit Details Page Summary Form page opens.
26. Under Configure Default Summary region, move the **Priority** and **Justification** fields to the **Selected Fields** box. Ensure that **Priority** field is above the **Justification** field. If it's not, use the up or down arrow button on the right to adjust the sequence.
27. Click the **Save and Close** button.
28. In the **Navigator** menu, click **Help Request**.
29. Click any active Help Request in the list to open its edit page.
30. Select **Important** from the **Priority** list.
31. Click the **Save** button.
The help request has been saved.
32. Now select **Urgent** in the **Priority** list.
An asterisk appears next to the **Justification** field that indicates a required field.
33. Click the **Save** button.
An error message appears, because you didn't enter a value in the **Justification** field before saving.
34. Enter **Laptop is on fire** in the **Justification** field.
35. Click the **Save and Close** button.
The help request is now saved.

Making Fields Conditionally Updatable

In this example, you add the following two custom fields to your Help Request object and then enter a script to make one of them conditionally updatable.

1. **Executive Sponsor Program** check box
2. **Executive Sponsor** text field

If the Executive Sponsor Program check box is selected, the Executive Sponsor field can be updated. Else, the Executive Sponsor field is disabled.

To create these fields and make one of them conditionally updatable:

1. Navigate to Application Composer.
2. Expand **Custom Objects**.
3. Expand **Help Request** and click **Fields**.
4. Under Custom tab, select **Action > Create**.
5. In the Select Field Type window, select **Text** and click **OK**.
6. On the Create Text Field page, enter **Executive Sponsor** in the **Display Label**.
7. Click **Save and Close**.
8. On the Fields page, select **Action > Create** under the Custom tab.

9. In the Select Field Type window, select **Check box** and click **OK**.
10. On the Create Text Field page, enter **Executive Sponsor Program** in the **Display Label**.
11. Click **Save and Close**.
12. On the Fields page, click the **Executive Sponsor** link.
Edit Check box Field: Executive Sponsor page opens.
13. Under Constraints region, select the **Updatable** check box and click the expression builder icon next to it.
Expression builder dialog opens.
14. Enter the following into the (script) text box:

```
try{ if(nvl(ExecutiveSponsorProgram_c,"") == "N") { return false } else { return true } } catch(e)
{ println("Error with the Updatable property of Executive Sponsor field in the Help Request object") }
```
15. Click the **OK** button.
16. Under Constraints region, select **Executive Sponsor Program** from the **Depends On** list.
17. Click the **Save and Close** button.
18. Under **Help Request** in the left pane, click **Pages**.
Help Request: Pages page opens.
19. Click the **Edit Summary Form** link.
Edit Details Page Summary Form page opens.
20. Under Configure Default Summary region, move the **Executive Sponsor** and **Executive Sponsor Program** fields to the **Selected Fields** box.
Ensure that **Executive Sponsor Program** field is above the **Executive Sponsor** field. If it's not, use the up or down arrow button on the right to adjust the sequence.
21. Click the **Save and Close** button.
22. In the Navigator menu, click **Help Request**.
23. Click any active Help Request in the list to open its edit page.
24. Select the **Executive Sponsor Program** check box, and click the **Save** button.
At this point, you can enter a name into the **Executive Sponsor** field.
25. Now clear the selection in the **Executive Sponsor Program** check box, and then click the **Save** button.
You can't enter anything in the **Executive Sponsor** field, because you haven't selected the **Executive Sponsor Program** check box.

Adding Validations to Fields

In this example, you add a validation to a field using an expression. You first add a custom text field called Corporate E-Mail to your Help Request object, and then you add a validation to check the syntax of the e-mail address.

To create a field and add validation:

1. Navigate to Application Composer.
2. Expand **Custom Objects**.
3. Expand **Help Request** and click **Fields**.
4. Under Custom tab, select **Action > Create**.
5. In the Select Field Type window, select **Text** and click **OK**.
6. On the Create Text Field page, enter **Corporate E-Mail** in the **Display Label**.
7. Click **Save and Close**.
8. In the left pane, click **Pages** under **Help Request**.
Help Request: Pages opens.

9. Under Creation Page region, click **Edit Creation Page**.
Edit Creation Page opens.
10. Under Configure Creation page region, move **Corporate E-Mail** to the **Selected Fields** box.
11. Click the **Save and Close** button.
12. Under Details Page region, click **Edit Summary Form**.
Edit Details Page Summary Form page opens.
13. In the Configure Default Summary region, move the **Corporate E-Mail** to the **Selected Fields** box.
14. Click the **Save and Close** button.
15. In the left pane, click **Server Scripts** under **Help Request**.
16. Under Validation Rules tab, select **Action > Add** under the Field Rules region.
17. Specify the following values under Create Field Validation Rule page:

Field Name	Value
Field Name	Corporate E-Mail
Rule Name	CheckValid
Error Message	Invalid syntax in Corporate E-Mail. Enter a valid syntax for e-mail address

18. In the script text box, enter:

```
newValue == null || newValue =~ /[_A-Za-z0-9-]+(\.[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\.[A-Za-z0-9]+)*(\.[A-Za-z]{2,})/
```
19. Click the **Save and Close** button.
20. In the Navigator menu, click **Help Request**.
21. Click any active Help Request in the list to open its edit page.
22. Enter **mhoop.e.oracle.com** in the **Corporate E-Mail** field.
23. Click the **Save** button.
The error message that you had specified appears, because the e-mail address doesn't have proper syntax.
24. Enter **mhoop@oracle.com** in the **Corporate E-Mail** field.
25. Click the **Save** button.
You are now able to save the Help Request after entering a valid syntax for e-mail address.

Call REST Web Services from Groovy Scripts

How to Call RESTful Web Services from Groovy Scripts

You can call RESTful web services from your Groovy scripts in Application Composer. You might call a web service for access to internal or external data, or to perform a calculation on your data.

Calling RESTful web service methods from your scripts involves two high-level steps:

1. Creating a reference to the web service.

This includes registering the web service's endpoint with a variable name that you use in your Groovy script.

2. Writing a Groovy script in the Expression Builder that calls the web service.

For each call, the script must prepare the inbound arguments to the web service, call a web service method, and then process the return value from the web service.

Note: If the web service returns an error, you can use initiators like `setAttribute()` instead of web services inside the Groovy script.

Creating a Web Service Reference

To register a web service for use in your scripts, you first select **Web Services** in the Common Setup pane in Application Composer. You can select either REST or SOAP. To register REST services, select **REST**. You then associate a web service variable name with a URL that provides the location of the resource that represents the service you want to call.

For example, you might register a web service variable name of `TwitterSearch` for a web service that your application needs to call for retrieving tweets from Twitter, in this case, about Yosemite. The URL for this web service's location might be:

```
https://api.twitter.com/1.1/search/tweets.json?q=yosemite
```

Or, you might want to use this feature with an internal web service, such as within your cloud service. In this case, the URL might be:

```
http://host:port/OpptyService/rest/v1/Oppty?q=OpptyId=##OpptyId##
```

Of course, the server name, the port number, and path name for your actual service will be different. If the port number is omitted, then it's assumed that the service is listening on the default HTTP port number 80.

Writing a Groovy Script to Call a Web Service

When you call a web service from a Groovy script, the script must prepare the arguments to the web service before calling a web service method, and then process the data returned from the web service. Your script can also pass a structured payload to and from a web service.

You insert the code for the call to the web service from the **Web Services** tab in Expression Builder. The **Web Services** list displays the set of registered web service variable names and the **Functions** list displays the available methods for a given web service.

To insert a call to a RESTful web service in a Groovy script:

1. Select the **Web Services** tab in Expression Builder.
2. Select **REST**.
3. Select a variable name from the **Web Services** list.

4. Select a method from the **Functions** list.

The code that will be inserted is shown under **Function Signature**.

The information under Function Signature includes the parameter types and also the return type to indicate the type of variable the result of the call should be assigned to. The possible return types are:

Return Value	Return Type
Void	Void
Scalar values (integer, string and so on)	The actual Java return type
Object	Map
Collection	List

5. Position the cursor at the place in the script where you want to insert the web service call.

6. Click the **Insert** button to insert the code to call the web service method.

A web service call from a Groovy script has the following syntax:

```
adf.webServices.YourServiceVariableName.MethodName(args)
```

For example:

```
adf.webServices.ContactAddressAPI.Get("7627")
```

7. Click **Submit**.

Related Topics

- [Register REST Endpoints](#)
- [Integrate with Oracle SaaS REST Services](#)
- [Integrate with External REST Services](#)

Register REST Endpoints

In the Groovy scripts that you use in Application Composer, you can include calls to both internal and external web services. For each web service that you call in your scripts, you must first register the REST endpoint that you want to access.

Creating a Web Service Reference

To create a web service reference, do the following in Application Composer:

1. Confirm that you're in an active sandbox session.

2. In Application Composer, under Common Setup, click **Web Services**.
 3. Click the **Create Web Service Reference** icon.
 4. Select **REST**, then **OK**.
 5. Enter the name for this web service reference.
This name is simply an identifier that is used in the list of web services on the Expression Builder's Web Services tab.
 6. Specify the URL of the file location for the web service that you want to integrate with.
 7. If you select an authentication scheme, then specify the required information. For secure communication with a web service, you can use various schemes for authenticating user credentials and ensuring security. The following schemes are supported for web services from Groovy scripts:
 - o None
Select this option to specify that no security scheme is used.
 - o Call with basic authentication
 - o Propagate user identity using SAML over SSL
 - o Propagate user identity using SAML
 - o Call using OAUTH
Don't use this scheme with non-Oracle web service endpoints. This scheme currently supports only resources protected with Oracle Cloud OAuth server.
- For the security schemes that require user name and password credentials, specify a credential key. The web service provider will tell you about the credentials that you must use for a particular web service.
8. Next, select and configure methods against the resource. You can register the resource operation (GET, POST, and so on) and the associated payload format type (JSON/XML). Only registered operations appear in the Groovy expression builder.

Selection	Description
Method Type	Select the check box for the method type you want to expose in the Expression Builder. <ul style="list-style-type: none"> o GET o PUT o POST o PATCH o DELETE For each method you want to expose, specify the following information.
Method Name	The name of the method you selected appears here. By default it's the same as the method type (such as POST) but you can change it.
Format	Select a format for the method, based on what the selected web service returns. <ul style="list-style-type: none"> o XML o JSON

Selection	Description
	This information is provided by your web service provider or web service documentation.
Request Payload	<p>Specify the object structure of the payload, if needed. You can do this in one of two ways:</p> <ul style="list-style-type: none"> ○ Directly provide the schema URL that represents the object structure. ○ Provide a code sample in JSON or XML format. This is an optional parameter useful for displaying reference hints in the Expression Builder. <p>To obtain a JSON code sample, execute the API externally using a REST client, such as Postman. The request payload in your Postman execution forms the Request Payload for this Code Sample parameter.</p> <p>This section is optional.</p>
Response Payload	<p>If the method will return a response (for example, GET), specify the response object structure in which you want the response payload to be returned:</p> <ul style="list-style-type: none"> ○ Schema URL A URL that provides a structure for the data but doesn't include any values. ○ Code Sample <p>If you don't have a schema available, you can select this option and paste a sample response (for example, from the service documentation) in JSON or XML format into this field. This parameter is useful for displaying reference hints in the Expression Builder.</p> <p>To obtain a JSON code sample, execute the API externally using a REST client, such as Postman. The response payload in your Postman execution forms the Response Payload for this Code Sample parameter.</p> <p>Note: You must include an entry in the Response Payload field. If you don't, then Groovy returns "null" instead of the response payload. If you don't want to include a response payload, enter { } for JSON or <a/> for XML as the code schema sample.</p>

You can always edit existing web service references, for example, to change the security scheme used or the settings used for a particular security scheme.

After you create a web service reference, the name of the web service appears in the list available on the Web Services tab in the Expression Builder. When you select a web service from the list, you can then select any of the functions provided by the web service for use in your Groovy scripts.

Tip: When managing web service references, click the **Refresh** icon in the Web Services page to make sure the list is up to date. Read "Refreshing the List of Web Service References" below for information about when you need to click **Refresh**.

Refreshing the List of Web Service References

If new methods are added for a web service, you must click the **Refresh** icon on the Web Services page so that the web service reference is updated. Otherwise, the new methods won't be available for the web service in the Expression Builder.

The Refresh icon is applicable whenever the service contract with the client changes. This can result in new methods, in the changing of the signature of existing methods, or in the deletion of existing methods.

You might also want to click **Refresh** to display any new web service references that have been created in a separate user session.

Configuring Security

Configure security differently, depending on whether you're creating a web service reference to an associated service endpoint or unassociated service endpoint.

When creating a web service reference to an associated service endpoint, such as to Oracle Java Cloud Service - SaaS Extension residing in the same identity domain, the required setup including SSO enablement is completed during association.

- Learn more about association in the Oracle Cloud Managing and Monitoring Oracle Cloud at <http://docs.oracle.com/en/cloud/get-started/subscriptions-cloud/mmocs/managing-associations-services.html>.

To synchronize users between Oracle SaaS and Oracle PaaS (JCS-SaaS Extension), review the steps in <https://docs.oracle.com/en/solutions/embed-java-saas-extension-app-in-sales-cloud/synchronizing-oracle-sales-cloud-oracle-hcm-cloud-and-oracle-erp-cloud-user-identities-and-roles-sim.html#GUID-C3F3348C-8C74-4F17-B9CE-0893CE2FA7CC>.

When creating a web service reference to an unassociated service endpoint, most likely a non-Oracle Cloud service, do the following setup:

- Retrieve the server's CA SSL certificate from the service provider. Don't use self-signed certificates.
- Add the certificate to the OPSS Keystore Service. The Keystore Service enables you to manage keys and certificates for SSL, message security, encryption, and related tasks. You use the Keystore Service to create and maintain a keystore that contain keys, certificates, and other artifacts.

The customer certificate import isn't required.

- Use a secure connection (https) and use the default port (443).
- If basic authentication isn't necessary, then use the SAML over SSL authentication scheme.

Learn more about Oracle Single Sign-On in the Oracle Cloud Understanding Identity Concepts guide at <http://docs.oracle.com/en/cloud/get-started/subscriptions-cloud/ocuid/oracle-single-sign.html#GUID-379DAC22-B3AC-4957-AF60-D45A07CC8598>.

Resolving Security Setup Errors

If some security setup hasn't been performed, then you might receive some errors when the web service is called from a Groovy script.

- A bad encryption error, when message protection is used
- A PolicyEnforcementException error when message protection security is used.

You must create a service request with Oracle Support to resolve the errors. Retrieve the server's encryption certificate and the issuer certificate from the service provider and attach them both in the service request along with the location and the error details.

Moving Application Changes

You can download the application changes you make in a "source" environment, and upload them into a "target" environment. This can save you time when working with application changes across multiple environments.

To do this, do the following:

1. Use the Configuration Set Migration page to create a set of all changes and extensions made to an application environment.
2. Then, download the configuration set and upload it into another environment.

This is often referred to as configuration set migration, or CSM.

However, web service references created in Application Composer in the source environment won't work in your target environment after the migration. Therefore, after you upload the configuration set to the target environment, you must re-create the web service references using Application Composer in the target environment, as well.

You can also compare your application changes between two environments during the migration process. Use the Configuration Delta Report to do so. This report lets you check for configuration differences so you can easily troubleshoot issues that occur in one environment, but not in another. See *Migrate Your Configurations* for detailed information.

Related Topics

- [How to Call RESTful Web Services from Groovy Scripts](#)
- [Integrate with Oracle SaaS REST Services](#)
- [Integrate with External REST Services](#)

Integrate with Oracle SaaS REST Services

You can make REST outbound calls from your application to an ADF-based REST endpoint within the same application instance. This is useful when there is a need to do cross-application calls to fetch data from objects that might not be accessible using Groovy.

This topic illustrates different ways of making a call to an ADF-based REST endpoint within the same application instance.

In this topic, you will learn how to:

1. Register a base URL for an internal REST endpoint.
2. Append to that base URL in your Groovy scripts so that you can make queries to various objects.
3. Pass URL parameters dynamically.
4. Use the finders that are provided with your cloud service.
5. Use additional parameters to modify a REST endpoint.
6. Make query parameter calls.
7. Create a POST request to create a new contact address.

Registering the Base URL

To integrate with a REST endpoint, you must first register it as a variable in Application Composer so that you can later include that variable in your Groovy scripts. When you register the REST endpoint, you don't have to specify the entire endpoint. Instead, to save time, you can register only the base URL so that the REST endpoint is reusable. Later, in your Groovy script, you can specify the rest of the endpoint. For example, in your Groovy script, you can reference the base URL and then specify if you're making a call to retrieve information about contacts or accounts.

Let's look at an example. In this example, you will register a base URL. In the next section, you will use the base URL in a Groovy script.

First, register the base URL:

1. In Application Composer, under Common Setup, click **Web Services**.
2. Click the **Create Web Service Reference** icon.
3. Select **REST**, then **OK**.
4. Enter the variable name for this reference. For example, **GetObjects**.
5. Enter the base URL for the REST endpoint that you want to integrate with. For example:

```
https://<host:port>/crmCommonApi/resources/latest/##Object##
```

6. Use basic authentication. Select **Call with basic authentication**.
7. In the Credential Key field, specify a name for the secret key that can be used to access the web service. This key name along with the user name and password is stored in the credential store.
8. Next, select and configure methods against the resource. You can register the resource operation (GET, POST, and so on) and the associated payload format type (JSON/XML). Only registered operations appear in the Groovy expression builder. In this example, configure a GET method.

Field	Value
Method Name	GET
Format	JSON
Request Payload	Schema URL
Response Payload	Code Sample
Code Sample	<pre>{ "items" : [], "count" : 0, "hasMore" : false, "limit" : 25, "offset" : 0, "links" : [{ "rel" : "self", "href" : "<host:port>/crmCommonApi/resources/11.1.12/T1_c", "name" : "T1_c", "kind" : "collection" }] }</pre>

Finally, use the base URL in a Groovy script. Read the next section to learn how.

Appending to the Base URL

After you register the base URL, you can then reference the base URL in your Groovy scripts and manipulate the REST endpoint to specify which object you want to integrate with. For example, you can reference the same base URL to query either contacts or accounts. To do this, further append to the URL to specify which object you want to integrate with.

Continuing the example from the previous section, let's now reference the REST endpoint in a Groovy script.

```
def conn = adf.webServices.GETObjects
try{
  def Object = "contacts"
  def result = conn.GET(Object)
  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Contacts:"+contacts)

}catch(Exception e){
  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Output"+conn.httpErrorResponse)
}
```

To query accounts using the same base URL, use this Groovy script:

```
def conn = adf.webServices.GetObjects
try{
  def Object = "accounts"
  def accounts = conn.GET(Object)
  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Accounts:"+accounts)

}catch(Exception e){
  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Output"+conn.httpErrorResponse)
}
```

Dynamically Passing Parameters in Your Groovy

When integrating with a REST endpoint, you can pass a parameter to refine your query. For example, when querying contacts, you might want to retrieve information for a particular party ID. You can do this without hard coding the party ID by dynamically passing the party ID in your Groovy script.

You can dynamically pass parameters based on where your Groovy is being called from. The page where the script is called from has context which you can use. For example, the page has context about the logged-in user, or about the contact record that the user is viewing. When you provide the URL, you can specify the party ID as a URL parameter.

Let's say that you defined a custom attribute to hold contact address information in a denormalized (concatenated) form. You require the denormalized form of the address to sync contact information into one of your legacy systems. You can achieve this use case by querying the Contacts API to fetch the address using Party Number as the parameter.

In this case, the REST endpoint would be:

```
https://<host:port>/crmCommonApi/resources/latest/contacts/##PartyNumber##/child/Address
```

And your Groovy script could look like this:

```
def ContactAddressAPI = adf.webServices.ContactAddressAPI
try
{
def contactAddress = ContactAddressAPI.GET("<PartyNumber>")
def address1 = contactAddress.items[0].Address1
def address2 = contactAddress.items[0].Address2
def city = contactAddress.items[0].City
def state = contactAddress.items[0].State
def country = contactAddress.items[0].Country
def postalCode = contactAddress.items[0].PostalCode

def denormalizedAddress = address1 + ", " + address2 + ", " + city + ", " + state + ", " + country + ", " +
postalCode
return denormalizedAddress //concatenated address

}
catch(Exception ex)
{
println(ContactAddressAPI.statusCode+""); //for diagnostic logging
println(ContactAddressAPI.httpErrorResponse+""); //for diagnostic logging
throw ex;
}
```

Using Finders

In your Groovy scripts, you can further refine the records that you retrieve by using finders to search a collection of data. Finders are predefined with your cloud service and are similar to a saved search. In your script, you state the finder name and include corresponding finder variables, if any, depending on the finder that you're using.

As mentioned earlier, you can attach the PartyNumber directly to the REST endpoint URL itself to retrieve a specific contact. For example:

```
https://<host:port>/crmCommonApi/resources/11.12.1.0/contacts/##PartyNumber##
```

Or, use a finder in your script. Each object in your cloud service is shipped with a set of finders. For example, the following are the available finders for the Contact REST endpoint:

- ContactPartyNumberRF: Finds contacts by party number.
- MyContacts: Finds a contact from My Contacts.
- MyBusinessContacts: Finds a contact from My Business Contacts.
- MyFavoriteContacts: Finds a contact from My Favorite Contacts.
- PrimaryKey: Finds a contact with the specified primary key.

The format to use a finder is:

```
?finder=<finderName>;<variableName>=<variableValue>,<variableName2>=<variableValue2>
```

Let's look at an example of using a finder. In this example, you will use the PrimaryKey finder to find a contact with the specified primary key. The variables for this finder are:

- PartyId (integer)

The Oracle record ID for the contact.

- **PersonProfileId (integer)**
 The unique identifier of the contact.

First, let's register the endpoint:

1. In Application Composer, under Common Setup, click **Web Services**.
2. Click the **Create Web Service Reference** icon.
3. Select **REST**, then **OK**.
4. Enter the variable name for this reference. For example, **Contact_Basic**.
5. Enter the URL for the REST endpoint that you want to integrate with. For example:
`https://<host:port>/crmCommonApi/resources/latest/contacts`
6. For security, use basic authentication. Select **Call with basic authentication**.
7. In the Credential Key field, specify a name for the secret key that can be used to access the web service. This key name along with the user name and password is stored in the credential store.
8. Next, select and configure methods against the resource. You can register the resource operation (GET, POST, and so on) and the associated payload format type (JSON/XML). Only registered operations appear in the Groovy expression builder. In this example, configure a GET method.

Field	Value
Method Name	GET
Format	JSON
Request Payload	Schema URL
Response Payload	Code Sample
Code Sample	<pre>{ "items" : [], "count" : 0, "hasMore" : false, "limit" : 25, "offset" : 0, "links" : [{ "rel" : "self", "href" : "http://<host:port>/crmCommonApi/resources/11.1.12/T1_c", "name" : "T1_c", "kind" : "collection" }] }</pre>

After you register the endpoint, you can then reference the endpoint and use the PrimaryKey finder in your Groovy script. The following example illustrates the use of both a finder, the fields parameter, and the query parameter in a single call to the Contact REST endpoint. (See the next two sections for discussions about the fields parameter and the query parameter.)

```
def conn = adf.webServices.Contact_Basic
try{
```

```
//Using finder and field parameters
def queryParams = ['finder':'PrimaryKey;PartyId=100000017340195', 'fields':'PartyId,PartyNumber']
conn.dynamicQueryParams = queryParams
def contacts = conn.GET()
println("Headers:"+conn.responseHTTPHeaders)
println("Status:"+conn.statusCode)
println("Contact after applying finder and field query parameters:"+contacts)

}catch(Exception e){
println("Headers:"+conn.responseHTTPHeaders)
println("Status:"+conn.statusCode)
println("Error:"+e)
}
```

For more information about finders and their corresponding finder variables that are available for each object, refer to the REST API documentation for your cloud service.

Passing Other Types of Parameters

Passing a parameter or using a finder are just two ways of modifying a REST endpoint. REST APIs also support queries that can filter a collection resource through the use of the `q` and `fields` parameters.

- `?q`

This query parameter defines the where clause. The resource collection will be queried using the provided expressions.

The format to use the `?q` parameter is:

```
?q=expression1;expression2
```

For example, maybe you want to retrieve all contacts in NY who belong to a specific department.

```
?q=Deptno>=10 and <= 30;Loc!=NY
```

- `?fields`

This parameter filters the resource attributes. Only the specified attributes are returned, which means that if no attributes are specified, no attributes are returned (useful to get only the links). Use this parameter to specify the fields that you want to retrieve with this call.

The format to use the `?fields` parameter is:

```
?fields=Attribute1,Attribute2
```

For more information about additional parameters that you can attach to REST calls, refer to the REST API documentation for your cloud service.

Making Query Parameter Calls

In addition to passing parameters as described earlier, you can also make query parameter calls.

You can define a query parameter and pass it directly. Or, define a payload and then pass the payload.

For example, define the query parameter itself. Note the response method is GET.

```
def queryParam = [OpptyId:'6756253']
OpptyDC.dynamicQueryParams = queryParam
def response = adf.webservices.OpptyDC.GET()
```

Or, define a payload and add it to the REST endpoint. This is especially useful when trying to create or update a record. Note the response method in the following example is POST.

```
def requestPayload = [OpptyName:'GreenServerTech', Account:'Pinnacle', Owner:'Lisa.Jones']
def response = adf.webservices.OpptyDC.Post(requestPayload)
```

Let's look at an example of defining a query parameter for a specific account record. In this example, you will define a query parameter and also pass the q parameter.

First, let's register the endpoint:

1. In Application Composer, under Common Setup, click **Web Services**.
2. Click the **Create Web Service Reference** icon.
3. Select **REST**, then **OK**.
4. Enter the variable name for this reference. For example, **GetAccountUsingSAML**.
5. Enter the URL for the REST endpoint that you want to integrate with. For example:

```
https://<host:port>/crmCommonApi/resources/latest/accounts
```

6. For security, use the Security Assertion Markup Language (SAML) over Secure Socket Layer (SSL) authentication scheme. Select **Propagate user identity using SAML over SSL**.
7. Next, select and configure methods against the resource. You can register the resource operation (GET, POST, and so on) and the associated payload format type (JSON/XML). Only registered operations appear in the Groovy expression builder. In this example, configure a GET method.

Field	Value
Method Name	GET
Format	JSON
Request Payload	Schema URL
Response Payload	Code Sample
Code Sample	<pre>{ "items" : [], "count" : 0, "hasMore" : false, "limit" : 25, "offset" : 0, "links" : [{ "rel" : "self", "href" : "http://<host:port>/crmCommonApi/resources/11.1.12/T1_c", "name" : "T1_c", "kind" : "collection" }] }</pre>

After you register the endpoint, you can then reference the endpoint and, in this example, pass both the query parameter and q parameter in your Groovy script.

```
def conn = adf.webServices.GetAccountUsingSAML
try{
    // Provide query parameter for the account object you want to receive
    def queryParams = ['q':'PartyId=300100010638186']
    conn.dynamicQueryParams = queryParams
    def accounts = conn.GET()
    println("Headers:"+conn.responseHTTPHeaders)
    println("Status:"+conn.statusCode)
    println("Account:"+accounts)

}catch(Exception e){
    println("Headers:"+conn.responseHTTPHeaders)
    println("Status:"+conn.statusCode)
    println("Error:"+e)
}
```

Creating POST Requests

You can execute standard methods such as GET, POST, PATCH, and DELETE on REST resources, using their URL. In this section, let's review POST requests.

Use a POST request to create a new item in a resource. The request content type is:

```
application/vnd.oracle.adf.resourceitem+json
```

Let's look at two POST requests. First, let's register the endpoint:

1. In Application Composer, under Common Setup, click **Web Services**.
2. Click the **Create Web Service Reference** icon.
3. Select **REST**, then **OK**.
4. Enter the variable name for this reference. For example, **Account_SAML**.
5. Enter the URL for the REST endpoint that you want to integrate with. For example:

```
https://<host:port>/crmCommonApi/resources/latest/accounts
```

6. For security, use the SAML over SSL authentication scheme. Select **Propagate user identity using SAML over SSL**.
7. Next, select and configure methods against the resource. You can register the resource operation (GET, POST, and so on) and the associated payload format type (JSON/XML). Only registered operations appear in the Groovy expression builder. In this example, configure a POST method.

Field	Value
Method Name	POST
Format	JSON
Request Payload	Code Sample
Code Sample	{"OrganizationName":"M1"}

Field	Value
Response Payload	Code Sample
Code Sample	<pre>{ "items" : [], "count" : 0, "hasMore" : false, "limit" : 25, "offset" : 0, "links" : [{ "rel" : "self", "href" : "http://<host:port>/crmCommonApi/resources/11.1.12/T1_c", "name" : "T1_c", "kind" : "collection" }] }</pre>

In the next two examples, you will create a POST request to create a new contact address using the REST endpoint that you registered earlier. Both examples show you how to set the HTTP headers in your Groovy script.

1. Set the HTTP request header content type in your POST request.

```
def conn = adf.webServices.Account_SAML
try{
  // Create new Account object by passing Organization name
  // Set Content-Type request header
  def OrganizationName =[OrganizationName:'TestOrganization1']
  def httpHeaders=['Content-Type': 'application/vnd.oracle.adf.resourceitem+json']
  conn.requestHTTPHeaders=httpHeaders
  def accounts = conn.POST(OrganizationName)

  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Account:"+accounts)

}catch(Exception e){
  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Error:"+e)
}
```

2. Set the HTTP response header content type in your POST request.

```
def conn = adf.webServices.Account_SAML
try{
  // Create new Account object by passing Organization name
  // Set Content-Type request header
  def OrganizationName =[OrganizationName:'TestOrganization2']
  def httpHeaders=['Content-Type': 'application/vnd.oracle.adf.resourceitem+json']
  conn.requestHTTPHeaders=httpHeaders
  def accounts = conn.POST(OrganizationName)

  println("Headers:"+conn.responseHTTPHeaders)
  // Retrieve Content-Type from response headers
  println("Content-Type:"+conn.responseHTTPHeaders['Content-Type'])
  println("Status:"+conn.statusCode)
  println("Account:"+accounts)

}catch(Exception e){
```

```
println("Headers:"+conn.responseHTTPHeaders)
println("Status:"+conn.statusCode)
println("Error:"+e)
}
```

Related Topics

- [How to Call RESTful Web Services from Groovy Scripts](#)
- [Register REST Endpoints](#)
- [Integrate with External REST Services](#)

Integrate with External REST Services

You can make REST outbound calls from your application to a non-ADF REST endpoint. This topic illustrates different ways of making a call to an external REST endpoint deployed to an Oracle PaaS service, such as Oracle Java Cloud Service - SaaS Extension.

In this example, the assumption is that JCS-SaaS Extension and your application are in the same Oracle Identity Domain and are associated. This means that the user identities are in sync and trust is enabled allowing for SAML-based user identity propagation.

In this example, let's assume that you created an external, non-ADF trouble tickets application and deployed it to JCS-SaaS Extension. Now, you want to make calls to that resource. This topic illustrates the following:

- Retrieving trouble tickets for a given account by passing an ID.
- Exception handling.
- Accessing elements in HTTP Response Headers.

Retrieving Trouble Tickets

First, let's retrieve trouble tickets for an account from a non-ADF REST endpoint on JCS-SaaS Extension.

Let's register the endpoint:

1. In Application Composer, under Common Setup, click **Web Services**.
2. Click the **Create Web Service Reference** icon.
3. Select **REST**, then **OK**.
4. Enter the variable name for this reference. For example, **GetTicketForAccount**.
5. Enter the URL for the REST endpoint that you want to integrate with. In this case:
`https://jcs-cakp.java.us2.oraclecloudapps.com/invokeTicket/troubleTicketApi/account/##AccountId##`
6. For security, use the Security Assertion Markup Language (SAML) over Secure Socket Layer (SSL) authentication scheme. Select **Propagate user identity using SAML over SSL**.
7. Next, select and configure methods against the resource. You can register the resource operation (GET, POST, and so on) and the associated payload format type (JSON/XML). Only registered operations appear in the Groovy expression builder. In this example, configure a GET method. (To enable the creation of new tickets, you can configure a POST method as part of this same endpoint registration.)

Field	Value
Method Name	GET

Field	Value
Format	JSON
Request Payload	Schema URL
Response Payload	Code Sample
Code Sample	<pre>{ "accountHolder": "abc@xyz.com", "requester": "Test1", "assignee": "Auto Assigned User-1", "share": true, "subject": "New keyboard", "description": "New keyboard request", "status": "New", "type": "Task", "priority": "Urgent", "tags": "New" }</pre>

After you register the endpoint, you can then reference the endpoint in your Groovy script to retrieve trouble tickets for a specific account.

The Groovy script would look like this:

```
def conn = adf.webServices.GetTicketForAccount
try{
  // Provide Account Id for which user wants to retrieve trouble ticket
  def tickets = conn.GET("6637911")

  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Output:"+tickets)
}catch(Exception e){
  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Error:"+e)
}
```

Exception Handling

In this next call, let's attempt to retrieve trouble tickets from the same non-ADF REST endpoint. However, in this example, the authentication scheme will be basic authentication, and the wrong credentials will be provided. This example illustrates how the REST endpoint behaves in the case of an unauthorized request.

Let's register the endpoint slightly differently this time. In this example, use basic authentication:

1. In Application Composer, under Common Setup, click **Web Services**.
2. Click the **Create Web Service Reference** icon.
3. Select **REST**, then **OK**.
4. Enter the variable name for this reference. For example, **TroubleTicketBasicAuth**.
5. Enter the URL for the REST endpoint that you want to integrate with. In this case:

`https://jcs-cakp.java.us2.oraclecloudapps.com/invoke/troubleTicketApi/tickets`

6. For security, use basic authentication. Select **Call with basic authentication**.
7. In the Credential Key field, specify a name for the secret key that can be used to access the web service. This key name along with the user name and password is stored in the credential store.

For the purposes of this example, enter incorrect credentials.

8. Next, select and configure methods against the resource. You can register the resource operation (GET, POST, and so on) and the associated payload format type (JSON/XML). Only registered operations appear in the Groovy expression builder. In this example, configure a GET method.

Field	Value
Method Name	GET
Format	JSON
Request Payload	Schema URL
Response Payload	Code Sample
Code Sample	<pre>{ "requester": "Mehul", "share": true, "subject": "Request for new monitor at my desk", "description": "Require bigger screen monitor", "status": "New", "type": "Task", "priority": "High", "tags": "Exchange" }</pre>

After you register the endpoint, you can then reference the endpoint in your Groovy script to retrieve trouble tickets with the wrong credentials.

The Groovy script would look like this:

```
def conn = adf.webServices.TroubleTicketBasicAuth
try{
  def tickets = conn.GET()

  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Trouble Tickets:"+tickets)

}catch(Exception e){
  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Error:"+e)
}
```

The response shows a 401 error, since your connection was created using the wrong credentials. When invocations fail, it is important to have the ability to retrieve exception headers and payloads to inspect the cause of the error. This demonstrates the ability for groovy to retrieve error payloads as well.

Accessing Elements in HTTP Response Headers

Finally, let's retrieve trouble tickets from the same non-ADF REST endpoint. In this final example, the authentication scheme will be basic authentication, and the correct credentials will be provided. This example illustrates how to retrieve HTTP response headers.

Let's modify the endpoint used in the previous example.

1. In Application Composer, under Common Setup, click **Web Services**.
2. Edit the **TroubleTicketBasicAuth** connection.
3. In the Credential Key field, specify the correct credentials for this call.
4. Click Save.

After you modify the endpoint, you can then reference the endpoint in your Groovy script to retrieve trouble tickets with HTTP response headers, using correct credentials.

The Groovy script would look like this:

```
def conn = adf.webServices.TroubleTicketBasicAuth
try{
  def tickets = conn.GET()

  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Content-Type:"+conn.responseHTTPHeaders['Content-Type'])
  println("Trouble Tickets:"+tickets)

}catch(Exception e){
  println("Headers:"+conn.responseHTTPHeaders)
  println("Status:"+conn.statusCode)
  println("Error:"+e)
}
```

Related Topics

- [How to Call RESTful Web Services from Groovy Scripts](#)
- [Register REST Endpoints](#)
- [Integrate with Oracle SaaS REST Services](#)

Call SOAP Web Services from Groovy Scripts

How to Call SOAP Web Services

You can call SOAP web services from your Groovy scripts in Application Composer. You might call a web service for access to internal or external data, or for example, to perform a calculation on your data.

Calling web service methods from your scripts involves two high-level steps:

1. Creating a reference to the web service. This includes registering the web service with a variable name that you use in your Groovy code.
2. Writing Groovy code in Expression Builder that calls the web service. For each call, the code must prepare the inbound arguments to the web service, call a web service method, and then process the return value from the web service.

Creating a Web Service Reference

To register a web service for use in your scripts, you first select **Web Services** in the Common Setup pane in Application Composer, then select **SOAP**. You then associate a web service variable name with a URL that provides the location of the Web Service Description Language (WSDL) resource that represents the service you want to call.

For example, you might register a web service variable name of `EmployeeService` for a web service that your application needs to call for working with employee data from another system. The URL for this web service's WSDL might be:

```
http://example.com:8099/Services/EmployeeService?WSDL
```

Of course, the server name, the port number, and path name for your actual service will be different. If the port number is omitted, then it's assumed that the service is listening on the default HTTP port number 80.

Read "SOAP Web Service References for Groovy Scripts: Explained" for more information about creating web service references.

Writing Groovy Code to Call a Web Service

When you call a web service from a Groovy script, the code must prepare the arguments to the web service before calling a web service method, and then process the data returned from the web service. If your code passes structured data to and from a web service, read "Using Groovy Maps and Lists with Web Services" below.

You insert the code for the call to the web service from the **Web Services** tab in Expression Builder. The **Web Services** list displays the set of registered web service variable names and the **Functions** list displays the available methods for a given web service.

To insert a call to a web service in a Groovy script.

1. Select the **Web Services** tab in Expression Builder.
2. Select a variable name from the **Web Services** list.
3. Select a method from the **Functions** list.
 The code that will be inserted is shown under **Function Signature**.
4. Click the **Insert** button to insert the code to call the web service method.

A web service call from a Groovy script has the following syntax:

```
adf.webServices.YourServiceVariableName.MethodName(args)
```

The information under function signature includes the parameter types and also the return type to indicate the type of variable the result of the call should be assigned to. The possible return types are as follows:

Return Value	Return Type
Void	Void
Scalar values (integer, string and so on)	The actual Java return type

Return Value	Return Type
Object	Map
Collection	List

Using Groovy Maps and Lists with Web Services

When passing and receiving structured data to and from a web service, a Groovy map represents an object and its properties. For example, an Employee object with properties named Empno, Ename, Sal, and Hiredate would be represented by a map object having four key-value pairs, where the names of the properties are the keys.

You can create an empty Map object using the syntax:

```
def newEmp = [:]
```

Then, you can add properties to the map using the explicit put() method like this:

```
newEmp.put("Empno", 1234)
newEmp.put("Ename", "Sean")
newEmp.put("Sal", 9876)
newEmp.put("Hiredate", date(2013, 8, 11))
```

Alternatively, and more conveniently, you can assign and update map key-value pairs using a simpler direct assignment notation like this:

```
newEmp.Empno = 1234
newEmp.Ename = "Sean"
newEmp.Sal = 9876
newEmp.Hiredate = date(2013, 8, 11)
```

Finally, you can also create a new map and assign some or all of its properties in a single operation using the constructor syntax:

```
def newEmp = [Empno : 1234,
             Ename : "Sean",
             Sal : 9876,
             Hiredate : date(2013, 8, 11)]
```

To create a collection of objects you use the Groovy List object. You can create one object at a time and then create an empty list, and call the list's add() method to add both objects to the list:

```
def dependent1 = [Name : "Dave",
                 BirthYear : 1996]
def dependent2 = [Name : "Jenna",
                 BirthYear : 1999]
def listOfDependents = []
listOfDependents.add(dependent1)
listOfDependents.add(dependent2)
```

To save a few steps, the last three lines in the preceding example can be done in a single line by constructing a new list with the two desired elements in one line like this:

```
def listOfDependents = [dependent1, dependent2]
```

You can also create the list of maps in a single operation using a combination of list constructor syntax and map constructor syntax:

```
def listOfDependents = [[Name : "Dave",
```

```
BirthYear : 1996],  
[Name : "Jenna",  
BirthYear : 1999]]
```

If the employee object in the previous codes examples has a property named Dependents that's a list of objects representing dependent children, you can assign the property using the same syntax as shown above (using a list of maps as the value assigned):

```
newEmp.Dependents = [[Name : "Dave",  
BirthYear : 1996],  
[Name : "Jenna",  
BirthYear : 1999]]
```

Lastly, note that you can also construct a new employee with nested dependents all in a single statement by further nesting the constructor syntax:

```
def newEmp = [Empno : 1234,  
Ename : "Sean",  
Sal : 9876,  
Hiredate : date(2013,8,11),  
Dependents : [  
[Name : "Dave",  
BirthYear : 1996],  
[Name : "Jenna",  
BirthYear : 1999]]  
]
```

For more information on maps and lists, see the section called *Working with Maps* in the Groovy Scripting Reference guide.

Related Topics

- [SOAP Web Service References for Groovy Scripts](#)
- [Examples of SOAP Web Service Calls in Groovy Scripts](#)

SOAP Web Service References for Groovy Scripts

In the Groovy scripts that you use in Application Composer, you can include calls to both internal and external SOAP web services.

For each web service that you call in your scripts, you must set up a web service reference that specifies the Web Services Description Language (WSDL) file location and the security scheme, if any, used to access the web service.

To create a web service reference, do the following in Application Composer:

1. Select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the **New** icon, then select **SOAP**.
3. Specify a name for the web service connection.
4. Specify the URL of the WSDL file for the web service.

Note: Protected WSDLs aren't supported.

5. Specify the user and password credentials as required for the security scheme for the web service. Read "Specifying the Security Values for the Web Service" below for information about which schemes are supported.

Note: The WSS security user name and password aren't supported for non-SSL web services due to security issues.

After you create a web service reference, the name of the web service appears in the list available in the Web Services tab in the Expression Builder. When you select a web service from the list, you can then select any of the functions provided by the web service for use in your Groovy scripts.

You can edit existing web service references, for example, to change the security scheme used or the settings used for a particular security scheme.

Tip: When managing web service references, click the **Refresh** icon in the Web Services page to make sure the list is up to date. Read "Refreshing the List of Web Service References" below for information about when you need to click **Refresh**.

Specifying Variable Names

When you create a web service reference, you specify a variable name on the Create SOAP Web Service Connection page. This name is simply an identifier that's used in the list of web services in the Expression Builder.

Specifying WSDL URLs

The WSDL file for a web service provides information about a web service that includes the following:

- **Service.** Defines one or more ports for the web service.
- **Port.** Defines an address or connection endpoint to the web service.

For each service and port there can be one or more associated security policies.

To specify a WSDL URL:

1. On the Create SOAP Web Service Connection page, enter the WSDL file in URL format, for example:

```
http://internal-hosted:7101/MathsWS-Model-context-root/UsernameTokenSecurity?wsdl
```

2. Click **Read WSDL**.

The **Service**, **Port**, and **Security Scheme** fields are then populated based on what is found in the WSDL. When there are multiple services and ports defined, the **Service** and **Port** fields have the first service and port found in the WSDL selected.

3. If a different service and port is required for this web service, select the appropriate values in **Service** and **Port**.

When you select a particular service and port, a default security scheme is selected based on the security policy defined in the WSDL.

If the port number is omitted, then it's assumed that the service is listening on the default HTTP port number 80.

When registering a WSDL, make sure that the URL is added to the allowlist in the proxy, so that it can be accessed from the Oracle network.

Specifying the Security Values for the Web Service

For secure communication with a SOAP web service, you can use various schemes for authenticating user credentials and ensuring security. The following schemes are supported for SOAP web services from Groovy scripts. Hover over each option to view the associated WSDL security policy.

- None
- Call with basic authentication
- Call with separate user credentials over SSL
- Call with separate user credentials and message protection
- Propagate user identity using SAML
- Propagate user identity using SAML and message protection

Note: If a web service is hosted on the same environment as the Groovy script that calls the web service, then the separate user name and password credentials that you provide as security values are overridden when the flow is triggered. Instead, updates are recorded as made by the signed-in user who actually called the Groovy script, not the user registered to the web service.

On the Create SOAP Web Service Connection page, you specify a credential key for the security schemes that require user name and password credentials. The web service provider will tell you about the credentials that you must use for a particular web service.

Resolving Security Setup Errors

You may receive some errors if some security setup hasn't been performed. For example, you may get a SSL certificate error when you try to create the web service reference. In this case:

1. Retrieve the server's CA SSL certificate from the service provider, and ensure that the certificate chain is proper on the WSDL host. Don't use self-signed certificates.
2. Add the certificate to the OPSS Keystore Service. The Keystore Service enables you to manage keys and certificates for SSL, message security, encryption, and related tasks. You use the Keystore Service to create and maintain a keystore that contain keys, certificates, and other artifacts. The customer certificate import isn't required.

Also, use a secure connection (https) and use the default port (443).

You may also receive errors when the web service is called from a Groovy script:

- A bad encryption error, when message protection is used
- A PolicyEnforcementException error when message protection security is used.

For these errors you must also create a service request with Oracle Support to resolve the errors. You must retrieve the server's encryption certificate and the issuer certificate from the service provider and attach them both in the service request along with the WSDL location and the error details.

Using Worked Examples of Calling Web Services from Groovy

Worked examples of creating web service connections and calling the web service from a Groovy script are provided in separate topics as listed under "Related Links" below.

The topics cover the various security schemes that are supported for calls to both internal and external web services. The topics include information about contacting your administrator to resolve security setup errors where appropriate.

Refreshing the List of Web Service References

If new methods are added for a web service, you must click **Refresh** on the Web Services page so that the web service reference is updated. Otherwise, the new methods won't be available for the web service in the Expression Builder.

The Refresh action is applicable whenever the service contract with the client changes. This can result in new methods, changing of the signature of existing methods, and deletion of existing methods.

You might also want to click **Refresh** to display any new web service references that have been created in a separate user session.

Moving Application Changes

You can download the application changes you make in a "source" environment, and upload them into a "target" environment. This can save you time when working with changes across multiple environments.

To do this, you will do the following:

1. Use the Configuration Set Migration page to create a set of all changes and extensions made to an application environment.
2. Then, download the configuration set and upload it into another environment.

This is often referred to as configuration set migration, or CSM.

See: "Moving Application Changes".

However, web service references created in Application Composer in the source environment won't work in your target environment after the migration. Therefore, after you upload the configuration set to the target environment, you must re-create the web service references using Application Composer in the target environment, as well.

You can also compare your application changes between two environments during the migration process. Use the Configuration Delta Report to do so. This report lets you check for configuration differences so you can easily troubleshoot issues that occur in one environment, but not in another. See *Migrate Your Configurations* for detailed information.

Related Topics

- [How to Call SOAP Web Services](#)
- [Call an Internal SOAP Web Service from Groovy with Separate User Credentials over SSL](#)
- [Call an External SOAP Web Service from Groovy with Separate User Credentials over SSL](#)
- [Call an Internal SOAP Web Service with Message Protection Security](#)
- [Call an External SOAP Web Service from Groovy with Message Protection](#)

Examples of SOAP Web Service Calls in Groovy Scripts

This topic explains how you call SOAP web services from Groovy scripts using simple examples.

You can call web services from your Groovy scripts in Application Composer, for example, to access internal or external data, or to perform a calculation on your data.

Note: You can't use Groovy scripts to create an XML/SOAP message containing attachments.

A web service call from a Groovy script has the following syntax:

```
adf.webServices.YourServiceVariableName.MethodName(args)
```

In the examples in this topic, the methods of a web service registered with the variable name `EmployeeService` are called.

For each web service that you call in your scripts, you must set up a web service reference in the Web Services page in Application Composer.

Retrieving an Employee by ID

The following example shows how to call a `getEmployee()` method of the web service by passing the integer 7839 as the single argument to the method.

```
// retrieve Employee object by id from remote system
def emp = adf.webServices.EmployeeService.getEmployee(7839)
// log a message, referencing employee fields with "dot" notation
println("Got employee '+emp.Ename+' with id '+emp.Empno')
// access the nested list of Dependent objects for this employee
def deps = emp.Dependents
if (deps != null) {
    println("Found "+deps.size()+" dependents")
    for (dep in deps) {
        println("Dependent: "+dep.Name)
    }
}
```

Creating an Employee Including New Dependents

The following example shows how to use Groovy script's convenient map and list construction notation to create a new employee with two nested dependents. The `newEmp` object is then passed as the argument to the `createEmployee()` method of the web service.

```
// Create a new employee object using a Groovy map. The
// nested collection of dependents is a Groovy list of maps
def newEmp = [ Ename:"Steve",
    Deptno:10,
    Job:"CLERK",
    Sal:1234,
    Dependents:[ [Name:"Timmy",BirthYear:1996],
    [Name:"Sally",BirthYear:1998]] ]
// Create the new employee by passing this object to a web service
newEmp = adf.webServices.EmployeeService.createEmployee(newEmp)
// The service returns a new employee object which may have
// other attributes defaulted/assigned by the service, like the Empno
println("New employee created was assigned Empno = "+ newEmp.Empno)
```

Merging Updates to an Employee Object and Adding a Dependent Child Object

The following example shows how to use the `mergeEmployee()` method to update fields in an employee object that is retrieved at the start of the script using a call to the `getEmployee()` method. The script updates the `Ename` field on the retrieved `emp` object and updates the names of the existing dependents. The script then adds a dependent child object before calling the `mergeEmployee()` method of the web service to save the changes.

```
// Merge updates and inserts on Employee and nested Dependents
def emp = adf.webServices.EmployeeService.getEmployee(7839)
// update employee's name to add an exclamation point!
emp.Ename = emp.Ename + '!'
def deps = emp.Dependents
// Update dependent names to add an exclamation point!
for (dep in deps) {
    dep.Name = dep.Name + '!'
}
// Add a new dependent
def newChild = [Name:"Jane", BirthYear:1997]
deps.add(newChild)
emp = adf.webServices.EmployeeService.mergeEmployee(emp)
```

Related Topics

- [How to Call SOAP Web Services](#)
- [SOAP Web Service References for Groovy Scripts](#)

Call an External SOAP Web Service from Groovy When No Security Scheme Is Required

This example shows how to create a connection to an external SOAP web service on the Internet and call the web service from a Groovy script used in Application Composer. The web service isn't secured.

For this example, the web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	http://external-hosted:7101/MathsWS-Model-context-root/NoSecurity?wsdl Note: The URL shown here's an arbitrary example. You must obtain the real WSDL URL from the service provider.
Where will the web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which web service method will be called from the Groovy script?	getSum This method returns the sum of two integer argument values.

To call a web service from a Groovy script when no security scheme is required, complete the following tasks:

1. Create the web service connection.
2. Add the web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL to use from the web service provider.
2. Create a custom field for an object that has a calculated default value.
3. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a web service connection, you specify a name for the web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that's used in the list of web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon, then click **SOAP**.
3. On the Create SOAP Web Service Connection page, enter `mathsWS` in the **Name** field.

The name must not include periods.

4. Enter `http://external-hosted:7101/MathsWS-Model-context-root/NoSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.

After you click **Read WSDL**, the **Service** and **Port** fields are filled according to values in the WSDL file. Under **Security Scheme**, the **None** radio button becomes enabled and selected.

5. Click **Save and Close**.

The web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the web services for which you have created a connection. For each web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsWS` from the **Web Services** list.
5. Select `getSum` from the **Functions** list.
6. Position the cursor at the place in the script where you want to insert the web service call.
7. Click **Insert** to insert the code to call the web service method.
8. Update the script so that two integer values are provided as arguments for the web service call.
9. Click **Submit**.
10. Verify that the web service call succeeds; in this example the custom field should have the expected default value.

Call an External SOAP Web Service from Groovy with Message Protection

This example shows how to create a connection to an external, secured SOAP web service and call the web service from a Groovy script used in Application Composer. The web service is secured with message protection.

For this example, the web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	<p>http://external-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl</p> <p>This WSDL file specifies the desired message protection security scheme.</p> <p>Note: The URL shown here's an arbitrary example. You must obtain the real WSDL URL from the service provider.</p>
Which credential key will you use?	mylogin
Where will the web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which web service method will be called from the Groovy script?	<p>getSum</p> <p>This method returns the sum of two integer argument values.</p>
What will the server encryption alias name be?	serverenckey
Is it required to ignore the time stamp in the response from the web service?	<p>Yes.</p> <p>To ignore the time stamp, you select the Disable Time Stamp Verification check box. This may be required to address interoperability issues.</p>

To call a web service from a Groovy script that's secured with message protection, complete the following tasks:

1. Create the web service connection.
2. Add the web service call to the Groovy script, and check whether the call succeeds.
3. Contact the administrator to resolve runtime exceptions.
4. Re-create the web service connection.
5. Verify that the web service call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL and the user credentials to use from the web service provider.
2. Get the server encryption certificate and the Certificate Authority (issuer) certificate from the web service provider.
3. Create a custom field for an object that has a calculated default value.
4. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a web service connection, you specify a name for the web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that's used in the list of web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon, then click **SOAP**.
3. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.

The name must not include periods.

4. Enter `http://external-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.
5. Click the New Key icon next to the **Credential Key** field.
6. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the web service provider, and click OK.
7. Select **Disable time stamp verification** so that the time stamp in the response header from the web service is ignored.
8. Click **Save and Close**.

The web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the web services for which you have created a connection. For each web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsws` from the **Web Services** list.
5. Select `getSum` from the **Functions** list.
6. Position the cursor at the place in the script where you want to insert the web service call.
7. Click **Insert** to insert the code to call the web service method.
8. Update the script so that two integer values are provided as arguments for the web service call.
9. Click **Submit**.
10. Verify that the web service call succeeds; in this example the custom field should have the expected default value.

Contacting the Administrator to Resolve Runtime Exceptions

The web service call may fail due to a number of exceptions including path certification, bad encryption, and policy enforcement exceptions. You must create a service request for your administrator to resolve the issues.

1. Create a service request for your administrator:
 - a. Retrieve the server encryption certificate and the Certificate Authority (issuer) certificate from the web service provider.
 - b. Attach the server encryption certificate and the issuer certificate to the service request, and include the WSDL location, and error details.
 - c. Submit the service request.

The administrator will add the server encryption certificate and the issuer certificate into the Oracle Fusion CRM trust store. The administrator also creates an alias for the server encryption key, which you will use in the next task.

2. Wait until your administrator informs you that the certificates have been imported, and that the server encryption alias has been created, and then close the service request.

Recreating the Web Service Connection

After your administrator has resolved runtime exceptions, you must re-create the web service connection and this time specify the server encryption key alias supplied by the administrator.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, select the web service connection you created previously, and click the Delete icon.
3. On the Web Services page, click the New icon, then click **SOAP**.
4. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.
5. Enter `http://external-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.
6. Click the New Key icon next to the **Credential Key** field.
7. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the web service provider, and click OK.
8. Select **Disable time stamp verification** so that the time stamp in the response header from the web service is ignored.
9. On the Create SOAP Web Service Connection page, enter `serverenckey` in the **Outgoing Encryption Key** field.
10. Click **Save and Close**.

The web service connection is created and the name and WSDL URL are listed on the Web Services page.

Verifying that the Web Service Call Succeeds

After you have re-created a web service connection, you must verify that the call to the web service succeeds.

1. Make sure that the Groovy script contains the code to call the web service.
2. Verify that the web service call succeeds; in this example the custom field should have the expected default value.

Call an External SOAP Web Service from Groovy with Separate User Credentials over SSL

This example shows how to create a connection to an external, secured SOAP web service and call the web service from a Groovy script used in Application Composer. The web service uses a security scheme with separate user credentials and secure sockets layer (SSL).

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	<p>https://external-hosted:7102/MathsWS-Model-context-root/UsernameTokenOverSSLSecurity?wsdl</p> <p>This WSDL file specifies the desired SSL security scheme.</p> <p>Note: The URL shown here's an arbitrary example. You must obtain the real WSDL URL from the service provider.</p>
Which credential key will you use?	mylogin
Where will the web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which web service method will be called from the Groovy script?	<p>getSum</p> <p>This method returns the sum of two integer argument values.</p>
Is it required to ignore the time stamp in the response from the web service?	<p>Yes.</p> <p>To ignore the time stamp, you select the Disable time stamp verification check box. This may be required to address interoperability issues.</p>

To call a web service from a Groovy script that's secured with SSL, complete the following tasks:

1. Create the web service connection.
2. Add the web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL and the user credentials to use from the web service provider.
2. Get the server's Certificate Authority (CA) SSL certificate from the web service provider.
3. Create a custom field for an object that has a calculated default value.

4. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a web service connection, you specify a name for the web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that's used in the list of web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon, then click **SOAP**.
3. On the Create SOAP Web Service Connection page, enter `mathsWS` in the **Name** field. The name must not include periods.
4. Enter `https://external-hosted:7102/MathsWS-Model-context-root/UsernameTokenOverSSLSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.
You must create a service request for your administrator to resolve the issue.
5. Create a service request for your administrator:
 - a. Retrieve the server's Certificate Authority (CA) SSL certificate from the web service provider.
 - b. Attach the SSL certificate to the service request, and include the WSDL location, and error details
 - c. Submit the service request.The administrator will add the SSL certificate into the Oracle Fusion CRM trust store.
6. Wait until your administrator informs you that the SSL certificate has been imported, and close the service request.
7. Repeat steps 1 through 4.
8. Click the New Key icon next to the **Credential Key** field.
9. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `myLogin`, enter the user name and password credentials supplied by the web service provider, and click OK.
10. Select **Disable time stamp verification** so that the time stamp in the response header from the Web service is ignored.
11. Click **Save and Close**.
The web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the web services for which you have created a connection. For each web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsWS` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.
6. Position the cursor at the place in the script where you want to insert the web service call.
7. Click **Insert** to insert the code to call the web service method.
8. Update the script so that two integer values are provided as arguments for the web service call.
9. Click **Submit**.
10. Verify that the web service call succeeds; in this example the custom field should have the expected default value.

Call an Internal SOAP Web Service from Groovy with Separate User Credentials over SSL

This example shows how to create a connection to a SOAP web service and call the web service from a Groovy script used in Application Composer. The web service uses a security scheme with separate user credentials and secure sockets layer (SSL).

For this example, the web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	<p>https://internal-hosted:7102/MathsWS-Model-context-root/UsernameTokenOverSSLSecurity?wsdl</p> <p>This WSDL file specifies the desired SSL authentication scheme.</p> <p>Note: The URL shown here's an arbitrary example. You must obtain the real WSDL URL from the service provider.</p>
Which credential key will you use?	mylogin
Where will the web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which web service method will be called from the Groovy script?	<p>getSum</p> <p>This method returns the sum of two integer argument values.</p>
Is it required to ignore the time stamp in the response from the web service?	<p>Yes.</p> <p>To ignore the time stamp, you select the Disable time stamp verification check box. This may be required to address interoperability issues.</p>

To call a web service from a Groovy script that's secured with SSL, complete the following tasks:

1. Create the web service connection.
2. Add the web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL and the user credentials to use from the web service provider.
2. Create a custom field for an object that has a calculated default value.

3. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a web service connection, you specify a name for the web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that's used in the list of web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon, then click **SOAP**.
3. On the Create SOAP Web Service Connection page, enter `mathsWS` in the **Name** field.

The name must not include periods.

4. Enter `https://internal-hosted:7102/MathsWS-Model-context-root/UsernameTokenOverSSLSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.
5. Click the New Key icon next to the **Credential Key** field.
6. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the web service provider, and click OK.
7. Select **Disable time stamp verification** so that the time stamp in the response header from the Web service is ignored.
8. Click **Save and Close**.

The web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the web services for which you have created a connection. For each web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsWS` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.
6. Position the cursor at the place in the script where you want to insert the web service call.
7. Click **Insert** to insert the code to call the web service method.
8. Update the script so that two integer values are provided as arguments for the web service call.
9. Click **Submit**.
10. Verify that the web service call succeeds; in this example the custom field should have the expected default value.

Related Topics

- [How to Call SOAP Web Services](#)
- [Examples of SOAP Web Service Calls in Groovy Scripts](#)
- [SOAP Web Service References for Groovy Scripts](#)

Call an Internal SOAP Web Service with Message Protection Security

This example shows how to create a connection to a SOAP web service and call the web service from a Groovy script used in Application Composer. The web service is secured with message protection.

For this example, the web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the web service connection?	mathsWS
What is the URL of the Web Services Description Language (WSDL) file that you will use?	<p>http://internal-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl</p> <p>This WSDL file specifies the desired message protection security scheme.</p> <p>Note: The URL shown here's an arbitrary example. You must obtain the real WSDL URL from the service provider.</p>
Which credential key will you use?	mylogin
Where will the web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which web service method will be called from the Groovy script?	<p>getSum</p> <p>This method returns the sum of two integer argument values.</p>
Is it required to ignore the time stamp in the response from the web service?	<p>Yes.</p> <p>To ignore the time stamp, you select the Disable time stamp verification check box. This may be required to address interoperability issues.</p>

To call a web service from a Groovy script that's secured with message protection, complete the following tasks:

1. Create the web service connection.
2. Add the web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL and the user credentials to use from the web service provider.
2. Create a custom field for an object that has a calculated default value.

3. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a web service connection, you specify a name for the web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that's used in the list of web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon, then click **SOAP**.
3. On the Create SOAP Web Service Connection page, enter `mathsWS` in the **Name** field.

The name must not include periods.

4. Enter `http://internal-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.
5. Click the New Key icon next to the **Credential Key** field.
6. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the web service provider, and click OK.
7. Select **Disable time stamp verification** so that the time stamp in the response header from the Web service is ignored.
8. Click **Save and Close**.

The web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the web services for which you have created a connection. For each web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsWS` from the **Web Services** list.
5. Select `getSum` from the **Functions** list.
6. Position the cursor at the place in the script where you want to insert the web service call.
7. Click **Insert** to insert the code to call the web service method.
8. Update the script so that two integer values are provided as arguments for the web service call.
9. Click **Submit**.
10. Verify that the web service call succeeds; in this example the custom field should have the expected default value.

Call an Internal SOAP Web Service from Groovy using SAML for ID Propagation

This example shows how to create a connection to a SOAP web service and call the web service from a Groovy script used in Application Composer.

The web service is secured by using Security Assertion Markup Language (SAML), which propagates the current user's security credentials for authentication. For this example, the web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	<p>https://internal-hosted:7102/MathsWS-Model-context-root/SamlOrUsernameTokenWithMessageProtection?wsdl</p> <p>Note: The URL shown here's an arbitrary example. You must obtain the real WSDL URL from the service provider.</p>
Where will the web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which web service method will be called from the Groovy script?	<p>getSum</p> <p>This method returns the sum of two integer argument values.</p>
Is it required to ignore the time stamp in the response from the web service?	<p>Yes.</p> <p>To ignore the time stamp, you select the Disable time stamp verification check box. This may be required to address interoperability issues.</p>

To call a web service from a Groovy script when SAML security is used, complete the following tasks:

1. Create the web service connection.
2. Add the web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL to use from the web service provider.
2. Create a custom field for an object that has a calculated default value.
3. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a web service connection, you specify a name for the web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that's used in the list of web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon, then click **SOAP**.
3. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.

The name must not include periods.

4. Enter `https://internal-hosted:7102/MathsWS-Model-context-root/SamlOrUsernameTokenWithMessageProtection?wsdl` in the **WSDL URL** field, and click **Read WSDL**.

After you click **Read WSDL**, the **Service** and **Port** fields are filled according to values in the WSDL file. Under **Security Scheme**, the **Propagate user identity using SAML** radio button becomes enabled and selected.

5. Select **Disable time stamp verification** so that the time stamp in the response header from the web service is ignored.
6. Click **Save and Close**.

The web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the web services for which you have created a connection. For each web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsWS` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.
6. Position the cursor at the place in the script where you want to insert the web service call.
7. Click **Insert** to insert the code to call the web service method.
8. Update the script so that two integer values are provided as arguments for the web service call.
9. Click **Submit**.
10. Verify that the web service call succeeds; in this example the custom field should have the expected default value.

Groovy Performance and Web Services

Fusion Application web services are designed for integration with external applications. As such, using Groovy scripts to call them involves authentication and authorization reprocessing which can impact script performance. Accordingly, try to implement business logic without relying on web services, if possible.

However, if your business requirement absolutely requires it, then:

- Ensure that the web service call is executed at the right time, ideally as late as possible in a commit life cycle. This ensures that the validations in the current transactions are passed.

Otherwise, it might lead to data inconsistency because the web service call commits automatically. For logic involving web services, consider using **After Commit**, **After Delete**, and **After Change Posted to DB** triggers.

- Avoid using web services to manipulate a business object that's directly accessible from Groovy.

Using Fields in Web Service Requests

When using fields in web service requests, there are some recommended best practices that you should follow to achieve optimal performance. Review this topic to understand the recommendations and constraints when including fields in a web service payload.

Which Fields Can Be Included in a Web Service Payload?

When creating a field in Application Composer, you can set the **Include in Service Payload** option. This option specifies whether or not the field value can be included in a web service request or response.

Long Text Field (CLOB)

Generally, long text fields are exposed only in detail pages. However, in web service calls, these fields are part of the response payload. Therefore, minimize usage where possible.

If you're going to use long text fields, then follow these recommendations:

- Limit the number of long text fields to two per object, when you use long text fields in web service calls.
- Avoid using a long text field for a text attribute, unless it needs to hold large values.
- Don't use long text fields in web service calls unless absolutely needed. These fields are high memory-consuming candidates.

Dynamic Choice List Fields

Including dynamic choice list (DCL) fields in a payload can impact the web service response time because it requires additional queries to fire. When you work with DCL fields, follow these recommendations:

- Limit the number of DCL fields included in a service payload to four fields.
- Use a good data filter to limit the number of records in the list of values (LOV).
- Only create a DCL field when absolutely needed to meet the business requirement.
- To access the object referenced by a DCL field through Groovy, use the respective DCL field's secondary Related Object Accessor field.
See *Using the Related Object Accessor Field to Work with a Referenced Object*.
- Don't write complex data security predicates for the target object because that will impact the query execution time, causing performance degradation.

Fixed Choice List Fields

Including fixed choice list (FCL) fields in a payload can impact the web service response time because it requires additional queries to fire. When you work with FCL fields, follow these recommendations:

- Limit the number of FCL fields included in a service payload to four fields.
- Only include an FCL field in the service payload when absolutely necessary.
- Don't create lookup values if they won't be used.

Formula Fields

A formula field gets evaluated whenever it's referenced. Therefore, include only necessary formula fields in the service payload. Follow these recommendations when you work with formula fields:

- Limit the usage of formula fields in the payload.
- Don't include any persistent fields in the formula field calculation logic, unless absolutely needed.

Nonindexed Fields

Only a limited number of columns are indexed in the database table for a custom object. Therefore, you should use this property only for the most frequently searched fields. After the field is created, you can't change this property.

Note that using nonindexed fields for filtering in a web service call can affect performance. When you work with nonindexed fields, follow these recommendations:

- Index only the fields most commonly used as web service filters.
- Avoid indexing fields that will be rarely or minimally used in filtering.

Runtime Messages

Use the Runtime Messages page, also known as the diagnostic dashboard, to view the diagnostic messages your scripts have written to the log. Use these diagnostic messages to assist with debugging your scripts.

On the Runtime Messages page, click the **Get Latest Log Messages** button to retrieve the latest Groovy script messages from the log file.

Runtime messages are diagnostic messages that you add to your script. They're useful for debugging your scripts if an error occurs.

Note: You can debug triggers by evaluating runtime messages. However, if you're using object workflows with Groovy scripts, then you can't use runtime messages to debug them.

To access the Runtime Messages page in Application Composer, go to the Common Tasks pane (on the bottom-left side), then click **Run Time Messages**. You must be in an active sandbox to perform this action.

Runtime messages are user-specific. Only you can see the messages that you create.

Note: The maximum limit for runtime messages in a sandbox is 250.

Use a Script to Write Messages to the Log

To write messages to the diagnostic log, use the `print` or `println` function. The former writes its value without any newline character, while the latter writes its value along with a newline. For example:

```
// Write a diagnostic message to the log. Notice how
// convenient string substitution expressions are
println("Status = ${Status_c}")
```

Find Messages

To find your messages on the Runtime Messages page:

1. In Application Composer, go to the Common Tasks pane (on the bottom-left side), then click **Run Time Messages**
2. Click the **Get Latest Log Messages** button to retrieve the latest Groovy script messages from the log file.

Your println Groovy scripts are written to different applications depending on the UI from which they're invoked (in other words, from where your scripts are triggered).

- If your script is triggered from the application UI, then the messages are written to Common Setup > Runtime Messages under the application that displays in the URL when you click the global Home icon: `http://<host>:<port>/<application>/faces/CrmFusionHome`.
For example, if the application in the URL is Customer, then navigate to Common Setup > Runtime Messages under the Customer Center application to find your runtime messages.
- If your script is triggered from the desktop UI, then the messages are displayed on the same Runtime Messages page, but under the actual web application from where your script was triggered.

For example, if a println Groovy script is tied to a Create Quote button which is displayed on a application UI Opportunity page, then the message will be printed in the Customer Center application's runtime messages. If the same action is displayed on the desktop UI Opportunity page, then the associated message will be printed in the sales and service application's runtime messages, since your script was triggered from the sales and service application.

To sort in reverse chronological order so you can see your most recent messages first, click the down-pointing arrow in the Time stamp column header.

Improve Performance

After testing is complete, consider removing the println function from your scripts to improve performance.

Troubleshoot the Runtime Messages Page

If you're having trouble viewing the runtime messages log, try enabling the runtime messages logging by doing the following:

1. In the Setup and Maintenance work area, search for the **Manage Profile Options** task.
2. On the Manage Profile Options page, click the New icon.
3. On the Create Profile Option page, create a profile option with the following details:
 - Profile Option Code: **ORACLE_ADF_BUSINESSEEDITOR_ENABLESCRIPTLOGGING**
 - Profile Display Name: **ORACLE.ADF.BUSINESSEEDITOR.ENABLESCRIPTLOGGING**
4. Fill all the other required fields with the relevant details.
You can choose logging at the site level or the user level and make it updatable.
5. Click **Save and Close**.
6. Go back to the Setup and Maintenance work area and search for the **Manage Administrator Profile Values** task.
7. On the Manage Administrator Profile Values page, search for the **ORACLE.ADF.BUSINESSEEDITOR.ENABLESCRIPTLOGGING** profile option that you just created and set its value to **True**.
8. Click **Save and Close**.

9. Send a service request to Support with the time frame in which you want the UI server to restart.

Debug Your Groovy Scripts

Use the Groovy debugger in Application Composer to debug the object functions and validations that you defined for an object. While debugging, you can also examine object and attribute values. Access the debugger from either the Custom Objects or Standard Objects page.

Accessing the Debugger

Access the debugger from either the Custom Objects or Standard Objects page in Application Composer.

To access the debugger:

1. In Application Composer, under the Objects tree, click either the Custom Objects or Standard Objects link.
2. On the resulting Objects page for either custom or standard objects, select the object that you want to debug and then click the debugger icon in the table's toolbar.
The debugger icon is a ladybug.
3. On the debugger UI, examine the object functions and validations defined in Groovy for that object.

Using the Debugger

The debugger contains multiple regions, described in the following table, which you can use to debug your scripts for an object:

Debugger Region	Description
Main toolbar	From the toolbar, you can select the object to examine and start the debugging process.
Left pane region	This region displays the object functions and validations defined for the selected object.
Main script region	This region displays the selected Groovy script.
Stack region	This region displays the call stack. For example, assume there are two functions, Function1 and Function2. Function1 calls Function2. When debugging within Function2, the Stack region displays which statement from Function2 is currently being executed, as well as information about the parent Function1 from where Function2 was called.
Variables region	This region displays variables and associated values.
Breakpoints tab	This tab displays which statement (line number) has a breakpoint. A breakpoint is a location in a Groovy script where you want the script to pause during debugging. The debugger stops at that statement.
Log tab	This tab displays all logs. If the script has any <code>println()</code> statements, then those values are captured on this tab.

To use the debugger:

1. In Application Composer, under the Objects tree, click either the Custom Objects or Standard Objects link.
2. On the resulting Objects page for either custom or standard objects, select the object that you want to debug and then click the debugger icon in the table's toolbar.
3. On the debugger UI, the left pane displays the object functions and validations defined in Groovy for that object. Select the script that you want to review.

The script is displayed in the main script region.

4. To start debugging, click one of these icons in the toolbar:

- o Step Over

Review one statement in the selected script at a time.

- o Step Into

If a statement in execution is a call to some function, and you want to debug inside that function, then click **Step Into**.

- o Step Out

If you are debugging inside a child function and you want to move the control back to the parent function, then click **Step Out**.

- o Run

Move to the next breakpoint in the script. If no further breakpoints exist, then the debugger completes its evaluation of the selected script and then closes the debugger session.

Enabling and Disabling the Debugger

The debugger is enabled by default. However, if you want to hide the debugger, or later show it again, then set the ADF: Enable Script Debugger profile option.

To set the ADF: Enable Script Debugger profile option:

1. In the Setup and Maintenance work area, select the following:
 - o Offering: Sales
 - o Functional Area: Sales Foundation
 - o Task: Manage Administrator Profile Values

The Manage Administrator Profile Values page appears.

2. In the Profile Display Name field, enter **ADF: Enable Script Debugger** and click **Search**.
3. In the Profile Values region, at the Site level, enter either **TRUE** or **FALSE**.
 - o TRUE displays the debugger.
 - o FALSE hides the debugger.

FAQ for Using Groovy Scripts

Why did my Groovy expression time out?

In general, avoid writing Groovy scripts that might require more than 60 seconds to complete.

A timeout of 60,000 milliseconds (60 seconds) is configured for Groovy expressions. If the expression requires more than 60 seconds to complete, an expression timeout (`oracle.jbo.ExprTimeoutException`) occurs and an error message is displayed.

For example:

```
Exception in expression "<object name>" object function <function name> : oracle.jbo.ExprTimeoutException  
Expression timed out. at "<object name>" object function <function name> line <line number>
```

The location where the error message is displayed depends on where the Groovy expression was executed.

If the Groovy expression is executed as a result of...	The error message appears in...
A UI operation	The UI.
A Web service update	The Web service client.
A workflow invocation	The error message is hidden from the end user.

Common situations where the Groovy expression timeout might be encountered include:

- If the Groovy expression attempts to iterate over a large collection of records, a Groovy timeout error might occur. In such a situation, any records that were initially modified are not actually committed.

For example, let's say that the following expression times out:

```
def vo=newView('TestCO_c')  
vo.executeQuery()  
while(vo.hasNext()){  
  def curRow=vo.next()  
  curRow.setAttribute('F1_c',curRow.RecordName)  
  println(curRow.RecordName+" : " + result)
```

```
}
```

Inspection of the `println` statements reveals that the `setAttribute()` operation was performed on a few records. However, if the timeout occurs while the Groovy execution is still in progress, none of the changes made are committed.

- If the Groovy expression calls a Web service, and if the Web service operation takes more than 60 seconds, a Groovy timeout error might occur. However, because the commit is part of the Web service, the commit will occur after the Web service operation is completed.

```
def response=adf.webServices.WS.Operation(<payload>);
```


6 Object Workflows

Overview

Use object workflows to automate business processes. When you create an object workflow, you state which business object you're impacting, such as Opportunity. You also define the actions that should take place in the application.

Conditions are events that occur immediately when the application implements your configured actions, unless you have defined a time rule (execution schedule) for one or more actions.

This chapter covers:

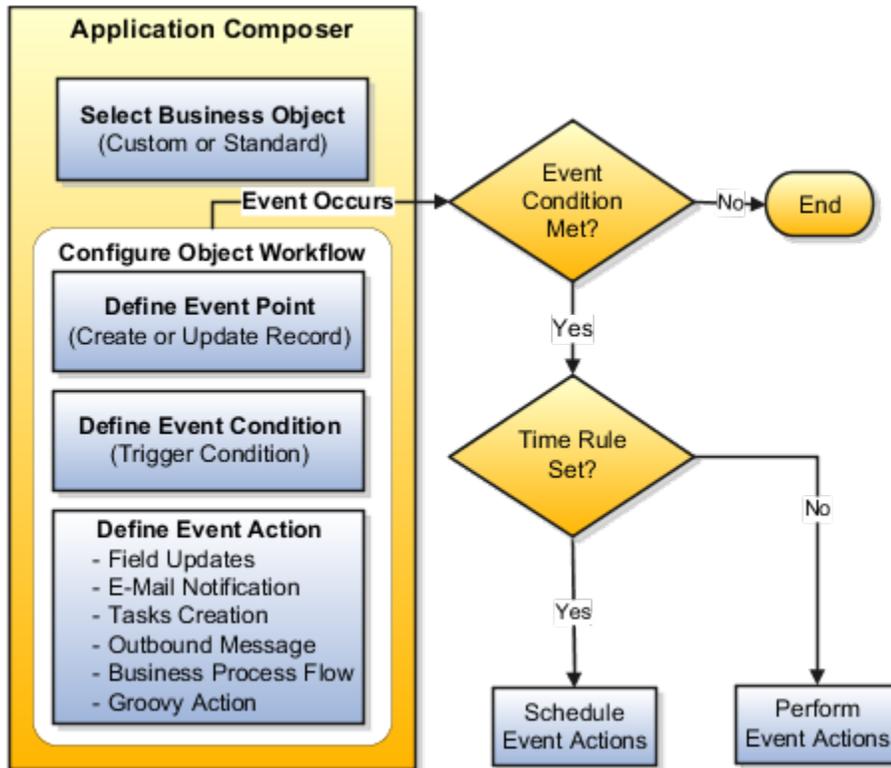
- Concepts and terminology used in object workflows.
- How you set trigger conditions for object workflows using groovy scripts.
- How you can use object workflows to configure event actions such as tasks, e-mail notifications, field updates, business process flows, and outbound messages.
- How you can create an approval flow using Oracle Business Process Management (BPM) Composer and then trigger the approval flow through object workflows.

About Object Workflows

Use object workflows to automate business processes. When you create an object workflow, you state which business object you're impacting, such as Opportunity. You also define the actions that should take place in the application.

Conditions are events that occur immediately when the application implements your configured actions, unless you have defined a time rule (execution schedule) for one or more actions.

This figure illustrates how you configure and trigger object workflows.



Access the Object Workflows Page

Here's how you access the page that you use for configuring object workflows. You must be in an active sandbox to configure an object workflow.

1. Navigate to Application Composer.
2. On the Overview page of Application Composer, click **Object Workflows**.
3. Select **Actions > Create**.

The Create Object Workflow page opens. Use this page to start configuring your object workflow.

Related Topics

- [Update Fields Using Object Workflows](#)
- [Send Email Notifications](#)
- [Create Tasks](#)
- [Send Outbound Messages to Web Services](#)
- [Object Workflows and Business Processes](#)

Commonly Used Object Workflow Terms

Familiarize yourself with the following terms before you start working with object workflows:

- **Active:** Indicates whether the workflow is in use. Only those object workflows which are in Active status can trigger event actions.
- **Business Object:** This can be either a standard object delivered with your cloud service or an object that you create based on your business need.

The business object that you use for configuring the object workflow can be either a parent object or a child object. All attributes available to you for selection when configuring a workflow belong to the object that you select for creating a workflow. This enforces that only the data relevant to the object in context is available for selection. For example, when you define a trigger condition, an expression editor lists only those fields that are relevant to the object that you have selected

- **Event Action:** An event action determines what action is expected from an object workflow when the conditions set for an object are met.

Event actions are of the following types:

- **Business Process Flow**

Use this action to trigger approval flows. For example, when the Deal Amount exceeds a threshold level, the relevant sales manager receives an approval notification. For more information on business process flow, see [How Object Workflows and Business Processes Work Together](#).

- **Email Notification**

Use this action to send automatic email alerts. For example, create a workflow that alerts the sales team that the budget availability date has changed. For more information on email notifications, see [How Object Workflows and Email Notifications Work Together](#).

- **Field Updates**

Use this action to specify the new values that you want to apply to the existing fields. For example, create a workflow that changes the Priority from Medium to High when the win probability exceeds 60 percent. For more information on field updates, see [How Object Workflows and Field Updates Work Together](#).

- **Groovy Action**

Use this action to trigger a Groovy action from your object workflow. For example, you might trigger a Groovy action to perform cross-object updates, perform post-processing of large object hierarchies, or asynchronously initiate Web services. For more information on Groovy actions, see [How Groovy Scripts and Object Workflows Work Together](#).

- **Outbound Message**

Use this action to send outbound message to a Web service at a specified endpoint URL. For example, send account details from one system to another. For more information on outbound message, see [How Object Workflows and Outbound Message Work Together](#).

- **Task Creation**

Use this action to create and assign tasks. For example, assign a follow-up task to the owner of an opportunity when the status is still open. For more information on task creation, see [How Object Workflows and Task Creation Work Together](#).

- **Event Condition:** Event point and event condition together serve as a trigger for object workflows. Event condition is an expression that supports logical, math operations, or field-value lookups. By defining an expression, you can prevent the object workflow to trigger each time a record is created or updated. See [Expression Builder](#) in this topic for an overview of how expressions are defined.

- **Event Point:** An event point is associated with an object and is an instance when an event occurs. Event points are of the following types:
 - When a record is created.
 - When a record is updated.
- **Execution Schedule:** You can set an execution schedule while defining an event action. This schedule governs when an event action should occur. Without a schedule, the event actions occur immediately. In case of multiple event actions, the Field Updates happen first before any other event action is triggered. For event actions other than Field Updates, there is no particular sequence. They're triggered based on whether or how you configure the execution schedule.

You can set a time rule for an event action based on whether that event action must occur after or before the triggering of a workflow or the occurrence of a date, and also specify the time duration in Hours, Days, or Weeks. If you schedule an event action for a time in the past, the event action executes immediately after it's triggered.

This figure illustrates the execution schedule region.

Related Topics

- [Update Fields Using Object Workflows](#)
- [Send Email Notifications](#)
- [Create Tasks](#)
- [Send Outbound Messages to Web Services](#)
- [Object Workflows and Business Processes](#)

Expression Builder for Object Workflows

Use the expression builder to write Groovy-based application logic that determines when an object workflow is triggered.

An expression builder supports building logical and math operations, including field lookups that you can optionally use to define trigger conditions. Fields in the expression builder are populated based on the object for which you're defining the workflow. The expression builder displays a warning if your expression contains an invalid attribute name; however, you must confirm whether the attribute name is actually invalid. If an attribute exists which was created at runtime, then you can safely ignore the warning.

Note: Object workflows aren't triggered when records are created through Import Management. Import Management bypasses any Groovy validation and trigger logic on an object.

Some examples of the raise conditions you can use include the following:

Example 1:

```
Status=='IN_PROGRESS' && BudgetAvailableDate==Today() + 30
```

Example 2:

```
if (isAttributeChanged('PrimaryContactPartyName') &&
```

```
PrimaryContactPartyName == 'Business World') return true; else return false;
```

Example 3:

```
WinProb>10 || WinProb<50
```

Consider the following when using the expression builder to create conditions:

- Ensure that you return a valid Boolean as part of your raise condition. Returning a non-boolean value could lead to runtime errors.
- Use `return true` or `return false` to explicitly return the Boolean value and code indentation when the evaluation logic is complex, to minimize risk of runtime errors.
- For lookup values, use the lookup code instead of the display value.
- Use `<`, `>`, `==` for comparison.
- For the event point **When a record is updated**, avoid redundant calls of the actions by always specifying which field change should trigger the object workflow, using the function `isAttributeChanged()`.
- Don't base your object workflow's `isAttributeChanged()` condition on a long text (CLOB) field. Instead, create a field-level trigger (or object-level trigger with an `isAttributeChanged()` condition) on the long text field to update a dummy text or number field. Then, you can define your object workflow condition using `isAttributeChanged(NewDummyField)`.
- Be aware of Groovy-type coercion, if you're not returning an explicit Boolean value.
- Use the logging capability to debug your condition and review the generated log by selecting **Runtime Messages** in the Common Setup pane.

For more information on Groovy scripting, see the Groovy Scripting Reference guide.

Related Topics

- [Update Fields Using Object Workflows](#)
- [Send Email Notifications](#)
- [Create Tasks](#)
- [Send Outbound Messages to Web Services](#)
- [Object Workflows and Business Processes](#)

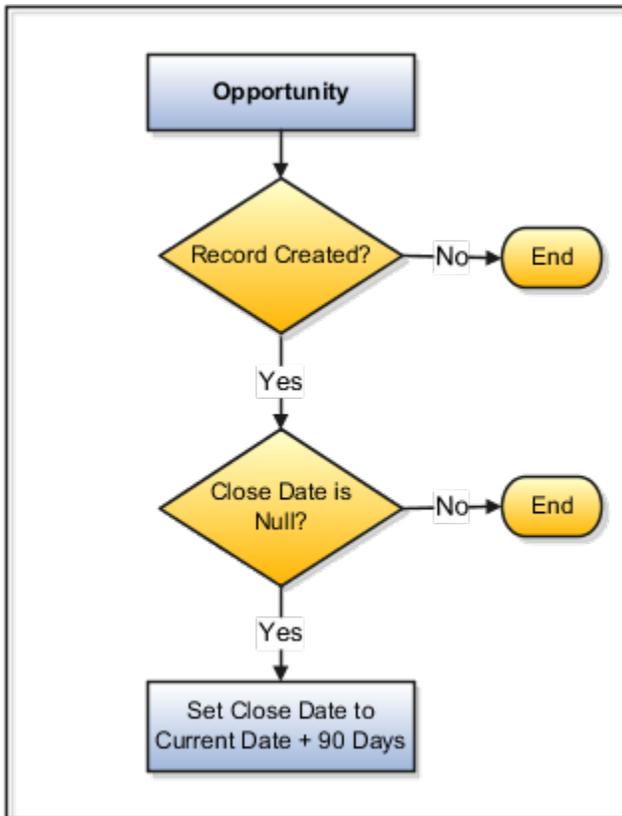
Examples of Object Workflows

Here are a few business scenarios where you use object workflows to automate business processes.

Scenario

In a sales division, the management plans to set an automated business process where an opportunity must have an initial close date set automatically to 90 days from its date of creation. Being an administrator, you must create an object workflow based on management directions.

This figure illustrates the triggering of the event action for the business scenario.



To create field updates event action:

1. From the Application Composer main page, select **Object Workflows**.
2. Click **Create**.
3. Select the **Opportunity** object and provide a meaningful **Name** and **Description**.
4. Define the trigger condition using **When a record is created** event point. Use the expression builder to set the event condition as **Close Date** is `Null`.
5. Under Actions, select **Field Updates** event action.

This opens the Create Action: Field Updates page where you configure the event action.

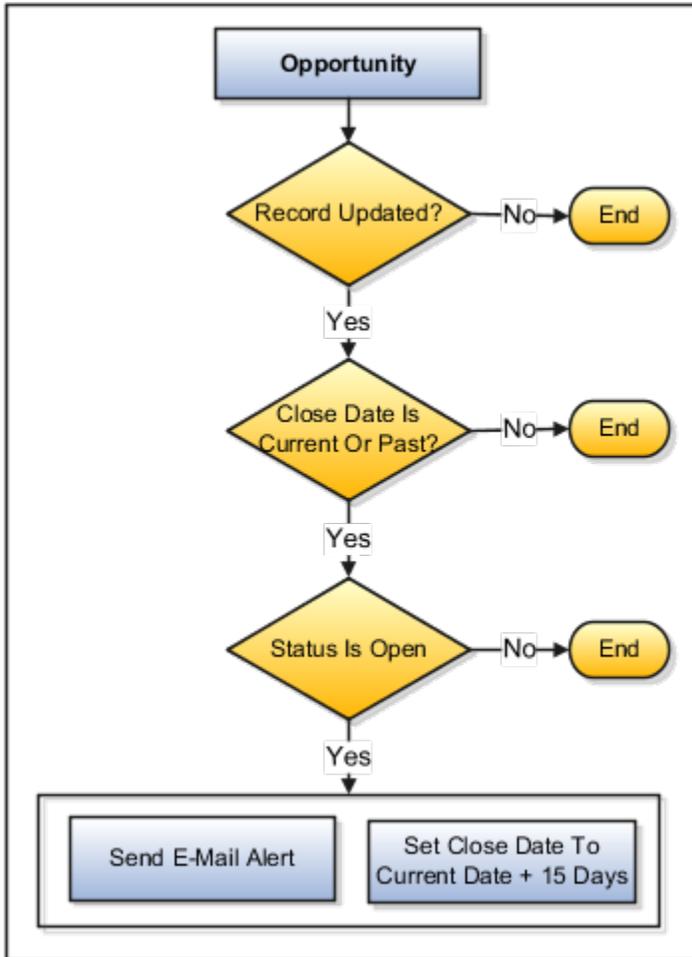
6. Provide the **Name** and **Description** for the field updates action and optionally set the **Execution Schedule**.
7. Under Field Update Details, select **Close Date** and set its value to **Creation Date** plus 90 days.
8. Save the event action.

When a user creates an opportunity, the workflow is triggered which sets the close date to 90 days from the date of creation.

Scenario

In a sales division, the management plans to set a business process that when an opportunity is updated whose close date is current or past and the status is open, an email notification is automatically sent to specified recipients and the close date is extended by 15 days. Being an administrator, you must create an object workflow based on management directions.

This figure illustrates how you configure multiple event-actions for the business scenario.



In this example, you must create a workflow with two event actions, namely, Field Updates and Email Notifications.

1. From the Application Composer main page, select **Object Workflows**.
2. Click **Create**.
3. Select the **Opportunity** object and provide a meaningful **Name** and **Description**.
4. Specify the event point as **When a record is updated**. Use the expression builder to set an event condition as `close Date is less than or equal to current Date` and `status is Open`.

You are now creating Field Updates event action.

1. On the Create Object Workflow page, select **Field Updates** event action.
This opens the Create Action: Field Updates page.
2. Provide the **Name** and **Description** for the field updates action and optionally set the **Execution Schedule** for triggering the action.
3. Under the Field Update Details, select **Close Date** and set its value to **Current Date** plus 15 days.
4. Save the event action.

You are now creating the Email Notification event action.

1. On the Create Object Workflow page, select **Email Notification** event action.
This opens the Create Action: Email Notification page.

2. Provide the **Name** and **Description** for the email notification action and optionally set the **Execution Schedule** for triggering such notifications.
3. Search and select an existing **Email Template** or create one with a related email subject and body.

When you create a template, you specify action-related text and field tokens that are populated at run time. See: Email Templates: Explained topic.

4. Locate and specify **Specific users** under **Recipient Types**. See: recipient types in Email Templates: Explained topic.
5. Click **Save**.

When a user updates an opportunity record which satisfies the trigger condition, an email is sent to the specified recipients and the **Close Date** is extended by 15 days.

Related Topics

- [About Object Workflows](#)
- [Send Email Notifications](#)
- [Update Fields Using Object Workflows](#)

Update Fields Using Object Workflows

A field update is an event action that you configure when creating an object workflow. When you configure more than one event action apart from configuring field updates, field updates always run first, and then other event actions are run in no specific order.

Note: The field update invoked from Object Workflow is a background update by a web service. If this update is performed while a record is open and you're making other updates, you get an error stating that the record has already been modified. This might also happen if you have multiple object workflows that are trying to update the same field on an object. If this happens, exit the record, reopen it, and reenter your updates. But to avoid this situation in the first place, avoid creating multiple object workflows that update the same object. Instead, consider if a synchronous object-level or field trigger, or validation rule, could achieve the same result without sacrificing performance.

Here's a summary of how to configure field updates as part of object workflows:

1. From the Application Composer overview page, select **Object Workflows**.
2. Select **Actions > Create**.
3. Select an **Object** and provide a meaningful **Name** for the workflow.
4. Define the trigger condition for the workflow.

Note: When defining the condition, ensure your Groovy script eliminates the repeated execution of this workflow.

5. Select the **Field Updates** event action.
6. On the Create Action: Field Updates page, provide a **Name** for the field updates action.
7. Specify the field to update and the new field value once the workflow is triggered. Use the **Update More Fields** option to select additional fields to update. See Configure Object Workflows for an example of how to configure field updates.

Note: If you have a record open and a workflow is triggered on a field in that record, you must exit from the record and reopen it to see the updates.

Specify Display Sequence for Fixed Choice Lists

In a fixed choice list, the list of field values can be either alphabetically arranged or have a display sequence such as High, Medium, or Low. When updating a fixed choice list field that has a display sequence, specify whether you want to populate the field with the next value, or with the previous value, in the list.

For example, for a display sequence of High, Medium, and Low, you can change from High to Medium (populate with next value) or from Low to Medium (populate with previous value) during a field update. However, if the current value is Low and you configure using populate with next value in list, the field isn't updated because Low is already the last value in the sequence.

Support for Primary and Child Objects

You can configure field updates using either a parent or a child object. When you select a child object when defining an object workflow, you can update fields only in the records of that child object; you can't update fields in the records of that child object's parent.

For example, when defining a workflow for Opportunity Team, which is a child object of Opportunity, you can configure field updates when a new team member (child record) is added to the Opportunity Team, but you can't configure field updates for records in the parent Opportunity object.

Additionally, when inserting field tokens while configuring field updates, only those tokens belonging to the selected child object is available for selection. See the Configure Object Workflows topic for an example of how to insert tokens when configuring field updates.

Object Workflows vs. Triggers

Object workflows are asynchronous. Use object workflows and the Field Updates action only when you need to update a field asynchronously, such as based on a time delay.

If you want changes to be immediately reflected on the user interface at runtime, however, use scripting and triggers, such as the Before Update trigger.

Related Topics

- [About Object Workflows](#)
- [Configure Object Workflows](#)

Object Workflows and Email Notifications

About Email Notifications

Email notifications send automated alerts to the specified recipients when the associated object workflow is triggered. You can configure email notifications using either a parent or a child object.

Email Templates

Email notifications are based on the email templates that you create. Email templates define email layouts, and ensure that the notifications triggered by the same type of business event have a consistent look and feel.

Recipient Types

The recipients available for sending the email notifications are contextual. If you're defining a workflow using a parent object, the recipients belong to the parent and its related child objects. If you're defining a workflow using a child object, then the recipients belong only to that child object. Recipient types include:

- From address
- To address
- Reply-to address
- Cc address
- Bcc address

You use field tokens to insert runtime values of fields. When inserting such tokens, only those fields belonging to the selected primary or child object are available for selection. See the [Configure Object Workflows](#) topic for an example of how to select and insert field tokens.

The following are the recipient types:

- **Fields on record:** If the selected object contains user-defined text fields that store emails, you can select those fields to send email notifications (for example, Primary Contact Email under Opportunity object). The user-defined text field can store a single email address or multiple comma-separated email addresses.
- **Relative users on record:** Contains relative users who could be either a creator of an opportunity record, a person who last updated a record, a manager or his directs, or a team of resources working on a project, and so on.
- **Roles:** Contains users assigned to a particular resource role. The email notification is sent to all enterprise groups or users that have been granted that resource role.

For information on how to manage sales users after you create them initially, see [How do I change user resource roles when job assignments change?](#) and the rest of the User Management chapter in the *Securing Sales and Fusion Service* guide.

When selecting a role, you can optionally apply an additional filter to specify the organization to which that role belongs. For example, if you want to send an email notification to a Sales Director role within a particular organization, specify that organization using the **Filter By** field in the dialog where you select the role.

- **Resources:** Resources associated with the record, such as Created By or Last Updated By.
- **Resource Groups:** Groups of resources associated with the records, such as Direct reports of Created By, or Subordinates of Last Updated By.
- **Specific users:** Contains details of individual users and their emails.
- **Specific email addresses:** Enter individual emails separated by a comma.

Note: The Search function for specific users includes a **Search** button to initiate the search; for the other recipient types, such as Roles, the application performs an automatic search after you type in your search terms and click or tab out of the search box.

Disable or Enable Notifications

If you have the BPM Workflow System Administrator Role role, then you can disable or enable these notifications for all users. For example, you can disable notifications during testing, to avoid sending test notifications to users, and then enable notifications when ready. For more information, see [Disable or Enable Workflow Notifications](#) in the Related Topics.

Note: Before you enable notifications for sales, ensure that you have enabled notifications generally. For steps, see [Disable or Enable Workflow Notifications](#) in the Related Topics. If you update the applications, be sure to go back and check these settings again.

Troubleshooting Email Notifications

If notifications are enabled but workflow emails aren't sending correctly, then check the Application Preferences page on the BPM Worklist Administration tab. Someone might have entered an email address in the Test Notification Email Address field that's no longer needed. Remove that email address and try again.

Additionally, the role of the logged-in user (who's invoking the workflow) must have access to the resource profile of the email recipient. Otherwise, email notifications won't be sent.

Related Topics

- [About Object Workflows](#)
- [Create and Manage Email Templates](#)
- [Configure Object Workflows](#)
- [Disable or Enable Workflow Notifications](#)
- [Send Test Workflow Email Notifications to One Address](#)

Send Email Notifications

Before you begin sending email notifications, ensure that email notifications are enabled for your application. For steps, see [Disable or Enable Workflow Notifications](#) in the Related Topics.

Here's how you can configure an email notification action as part of an object workflow:

1. From the Application Composer overview page, select **Object Workflows**.
2. Select **Actions > Create**.
3. Select an **Object** and provide a meaningful **Name** for the object workflow.
4. Define the trigger condition, and select **Email Notification** as the event action.
5. On the Create Action: Email Notification page, provide a **Name** for the email notification action.
6. Search for and select an existing email template, or create one.

When you create a template, you specify action-related text and field tokens that are populated at runtime.

7. Specify the recipient types.

See the [Configure Object Workflows](#) topic for an example of how to configure email notifications and email templates.

Specify Complex Notification by Email

To specify a complex email notification, where the email address is calculated rather than merely selected from a list or dynamically generated, you must define a field of type Formula to calculate the email address.

You can then use the Fields on record recipient type to select the user-defined field containing the formula.

For example, each time an opportunity is updated, you might want to send an email to notify a user saved in a user-defined object in Opportunity. You would do this by creating your own Formula text field on the Opportunity object to specify the user, then select this field using the Fields on record recipient type in the object workflow.

Troubleshoot Invalid Email Address Error

When you're creating a workflow and try to enter an email address that is different than the standard email domains, you may receive an "Invalid Email Address" message. For example, "test@dom-ain.com" has a special character after @ and can result in the error message "The value of the attribute Email Addresses is not valid".

Here's a workaround to create an object workflow with unusual email addresses:

Note: This workaround may not work for email addresses with _ after @. For example, "test@domain_xyz.com".

1. In Application Composer, create a custom text field with the unusual email address value or values, separated by commas.

See *Define Fields* and *Text Fields*.
2. Create a new object workflow and in the Email Notification section, click **Create**.
3. Provide a name for the action and select an email template.
4. In the To Address section, click the Edit icon.
5. Expand the Fields on Record section, select and move the field created in Step 1 from the list of available fields.
6. Click **Save**.
7. Continue with configuring the workflow as needed.

See *Configure Object Workflows*.

8. Test the workflow.

Related Topics

- [About Object Workflows](#)
- [Create and Manage Email Templates](#)
- [Configure Object Workflows](#)
- [Disable or Enable Workflow Notifications](#)

Create and Manage Email Templates

Email templates define the layout of the emails, which ensures that email notifications triggered by the same type of business event for a specific object have a consistent look and feel. You create a template for use with a particular type of object.

You then use this template only for that object when configuring email notifications. You can either create email templates or create templates using any email editor, and then upload them for use. You can create email templates, for example, for including your company letterhead in outbound email communications.

Note: You can't create email templates while inside an active sandbox.

Here are two ways in which you can create email templates in Application Composer:

1. When configuring email notification action.

On the Create Action: Email Notification page, select the Create Email Template icon in the Email Template field.

2. Using the Email Templates link on the Overview page of the Application Composer.

For an example on how you create email templates, see [Configure Object Workflows](#).

Here's what you can or can't do when creating email templates:

- You can perform basic formatting such as font format, character format, paragraph alignment, bullets and numbering, and so on as in any HTML based email client.

Templates are automatically converted to plain text for users who can't view HTML emails.

- For advanced HTML editing, use the Source Code Editing Mode, which is available as an icon under the Email Body region. Under this mode, the tags are set to visible. You can copy and edit the source code in any HTML editor, and then paste the edited code back. Similarly, you can use the Rich Text Editing Mode, which is available next to the Source Code Editing Mode icon.
- You can attach artifacts relevant to the email template. Attachments are included in every email, which uses the template that has an attachment. You can also send attachments as links to avoid crowding the inbox of the recipients. An attachment can be a file on a local computer or a shared file in a repository.

Note: File attachments can't exceed 10 MB.

- Specify whether an email template is Active. You can use only Active templates to create email notifications.
- You can insert fields or functions in the email body. You use functions to insert date, current date and time, or a hyperlink to a record. You can also Browse and then insert the content of a local file in the body of the email template.
- You can link directly to object details pages using primary key fields of the primary object records or using foreign keys for related objects. You can select these primary key and foreign key fields in the field drop-down list to substitute them into the direct links as tokens. The following objects are supported: Opportunity, Lead, Account, Contact, Household, Activity, and any objects you create. Refer to the description of direct linking for details on how to construct these links.
- You can specify the tokens or fields, which are populated with the values at run time.

Note: You can't use fields of type Dynamic Choice List as tokens.

- By default, the From field in workflow email notifications shows an email address without a sender name. You can change the sender's display name in the notification. See the [Change Sender in Email Notifications](#) section in this topic.
- Join fields aren't available in email templates.
- You must select **Include in Service Payload** for custom fields to be available in email templates.

Manage Email Templates

Managing email templates include tasks that enable you to search, view, duplicate, create, edit, or delete a template.

To manage email templates, select the Email Templates link on the Overview page of the Application Composer.

Options available for managing email templates include:

- Viewing the existing email templates and modifying views.
- Filtering or querying existing templates including partial search using wildcard.
- Editing an existing template or creating a duplicate. You can't change the object for which a template is defined.
- Creating a template.
- Detaching the listed templates for a full-page view.
- Viewing which templates are Active.

Note: You can use only Active templates for configuring email notifications. Saving doesn't automatically make the template Active.

Change Sender in Email Notifications

You can change the sender's display name in the email notification that's triggered from an object workflow. You can change the default previous approver to the submitter or provide another name that can act as an alias. Here's how you can do it:

1. Click the **Notifications** (bell) icon on your application page.
2. Click **Show All** to open the Notifications page.
3. On the Notifications page, click **Worklist** to open the BPM Worklist page.
4. On the BPM Worklist page, click the arrow against your name and select **Administration**.
5. On the Application Preferences page, scroll down to the **Email "From:" Display Name** field.
6. By default, this field is set to **Previous Approver**. You can either change it to **Submitter** or enter another name of your choice.
7. Click **Save**.

Based on your selection, all email notifications will now be sent with the submitter's email address or the alias that you specify.

Related Topics

- [About Object Workflows](#)
- [Send Email Notifications](#)
- [Direct Page Links for CX Sales](#)

Object Workflows and Tasks

Create Tasks

You can configure object workflows to trigger auto-creation and assignment of tasks.

You can define tasks action for only those objects that support the creation of tasks. For other objects, the tasks option is disabled.

Here's how you can create a task:

1. Navigate to Application Composer.
2. From the Overview page, click **Object Workflows**.
3. Click **Create**.
4. Select an **Object** and provide a meaningful **Name** and **Description**.
5. Define the trigger condition using **When a record is created** or **When a record is updated** event point.
6. Under the Actions region, click **Create** for Tasks Creation.

This opens the Create Action: Tasks Creation page.

7. Provide the **Name** and **Description** for the event action and optionally set the **Execution Schedule**.
8. Use the **Task Details** region to configure a task based on your requirements. Here's how you can use the tasks details region:
 - o Select a **Subject** and **Description**, as appropriate, for your task notification.
 - o Select the **Due Date** and **Start Date** for a task. When you select a date, you can also provide logical conditions for these dates, for example, a **Due Date** must be 30 days after the **Start Date**.
 - o Select an **Owner** for the task. The values in this list change depending on the object for which the task is being defined. A task can have only one owner: either an individual person or a single resource. You can search for users by name or user ID. You can store a user's PartyID or GUID in a text field you create and use the value as the task owner.
 - o Select the **Resources** for the task. The values in these lists change depending on the object for which the task is being defined. Resources can be owner, resource team, resource team with different access levels, member functions, and so on. You can store multiple comma-separated PartyIDs or GUIDs in a text field you create and use the values as task resources.

For example, when an opportunity reaches a **Close Date** and the status is **Open**, you can use the task creation action to assign a follow up task to the owner of the opportunity.

- o Select a **Primary Contact** for the task. You can select one from contacts related to the object record, or specify one by name. If the selected object contains user-defined text fields that store the contact's name or contactID, you can select those fields to be set as primary contact.
 - o Select the **Type** to which a task belongs. This field uses FND lookup for values. The delivered values are call, chat, demo, e-mail, and meeting.
 - o Assign a **Priority** to the task. The default priority is **Medium**.
9. If a standard object or an object you created has more than one one-to-many relationships defined through dynamic choice list or reference relationship with the Activity you have chosen, the **Related To** field appears at the bottom. This Related To field shows the first created relationship as default.

You can select this field only once for the object in context, after which the field appears as read only.

10. Save the event action.

Note: If an object for which you're defining this event action contains a customer, that customer is automatically included in the created task.

For an example of how you configure tasks, see [Configuring Object Workflows: Worked Example](#) topic.

Creating Tasks Using Child Objects

When you configure an object workflow using a child object, the fields that you select are specific only to that child object.

When a child-object based task is triggered, the task is assigned to the parent object of that child. You can't configure a task for a child object of another child object.

Creating Tasks Using Objects with No Associated Activity

You can create a task for a standard object or an object you created even if it doesn't have a relationship with an Activity. If you try to create a task on an object with no relationship to an Activity, you see a message letting you know that a standalone task will be created unless you first create such a relationship. After they're created, standalone activities will show in the task owner's Activity work area in the desktop UI, and in the top-level Activities card in the application UI.

Related Topics

- [About Object Workflows](#)
- [Examples of Object Workflows](#)
- [Object Relationships](#)
- [Configure Object Workflows](#)
- [Enable Task Creation for Object Workflows](#)

Enable Task Creation for Object Workflows

You can configure object workflows to trigger the automatic creation and assignment of tasks. For example, you might want to remind a sales representative to call a customer one month before their contract ends. Most objects support the creation of tasks.

However, for any object (for example, Asset) that doesn't have a standard relationship to Activity as delivered, the Task event action is disabled. To enable the Task event action, you create a relationship between the target object and Activity objects.

Create a relationship between the desired object and the Activity object in one of two ways:

- Create your own dynamic choice list field for the Activity object that points to the desired object's records.
- Create a one-to-many reference relationship with desired object as the source, and the Activity object as the target.

Tip: This type of relationship is similar to a dynamic choice list relationship. The difference is that when you want to create your own relationship between the objects but you don't need to show it in the UI as a choice list, then you might prefer to simply create the one-to-many relationship rather than the dynamic choice list relationship. With the one-to-many relationship, you don't get a choice list field for the object to add to any Activity user interface.

Once a dynamic choice list or reference relationship exists, you can create tasks for object workflows associated with the desired object.

Creating a Dynamic Choice List Field

In this example, let's create a dynamic choice list field for the Activity object that is populated by records from the Asset object.

1. In Application Composer, select the Activity object.
2. Select **Fields**.
3. Create a field of type Choice List (Dynamic).
4. Complete the general steps to configure a new dynamic choice list field. For example, set the display label for the field.
5. In the Related Object list, select the Asset object.
6. Complete the rest of the steps to create this dynamic choice list field. For example, in the List Selection Display Value field, select the field that displays as the first column in the dynamic choice list at run time.
7. Click **Submit**.

Creating a Relationship

In this example, let's create a relationship between the Activity and Asset objects.

1. Navigate to Application Composer.
2. Click the **Relationships** link under Common Setup.
3. Click the **Create** icon.
4. In the **Source Object** field, select the Asset object.
5. In the **Target Object** field, select Activity.
6. In the **Cardinality** field, select **1:M**.
7. Click **Save and Close**.

Related Topics

- [Object Relationships](#)
- [Create Tasks](#)
- [Overview of Dynamic Choice Lists](#)

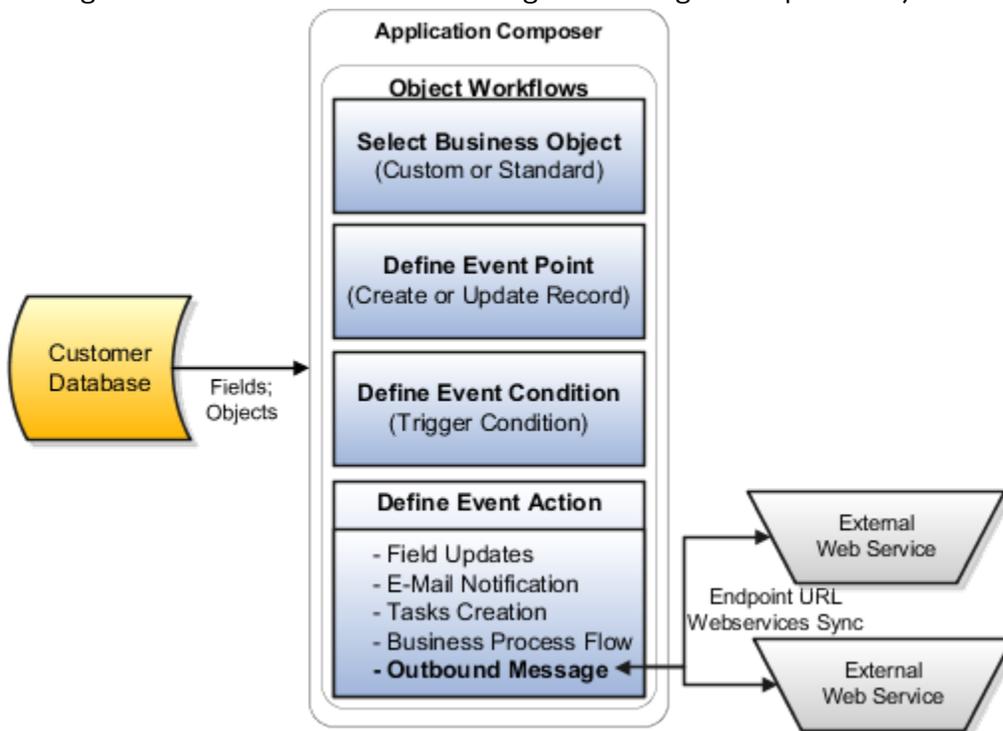
Object Workflows and Outbound Messages

Send Outbound Messages to Web Services

You can configure an object workflow to send an outbound message to a Web service at a specified endpoint URL. The endpoint URL is an external Web service that shares data with your applications, and must conform to your Oracle service WSDL.

For example, you can define an object workflow that automatically sends, based on a trigger, an outbound message containing an object like opportunity, lead, or account details from one system to another.

This figure illustrates how outbound messages are configured as part of object workflows.



You can configure outbound-messages event for a parent or a child object. When you define a workflow using a parent object, the outbound message is sent using service data objects (SDO) of that parent object as well as its child objects. When you define a workflow using a child object, the outbound message is sent using the SDO of only the child object.

To define an outbound message action:

1. From the Application Composer main page, select **Object Workflows**.
2. Click **Create**.
3. Select an **Object** and provide a meaningful **Name** and **Description** for the workflow you are creating.
4. Define the trigger condition using **When a record is created** or **When a record is updated** event point.
5. Select the **Outbound Message** event action.

This opens the Create Action: Outbound Message page.

6. Specify the **Name** and **Description** for the outbound message action and set the **Execution Schedule** for triggering the action. See: Execution Schedule in this topic.
7. Provide the **endpoint URL** of the external Web service.

To build the external Web service, use the OutboundMessageService.wsdl and .xsd files. Then, when defining the outbound message event action for the object workflow, you provide the endpoint URL of the external Web service.

For example, an endpoint URL can be `http://GlobalFusion:7011/OMTestOpportunity/OutboundMessageServiceSoapHttpPort`. At run time, a service data object containing details of the object on which the object workflow is defined is sent to the specified endpoint URL.

8. Save the event action.

Execution Schedule

While defining an event action, you can optionally set an execution schedule that governs when an event action should happen. Else, the event actions are run immediately. When multiple event actions are configured, the Field Updates

happen first before any other event action is triggered. For event actions other than Field Updates, there is no particular sequence and the event actions are triggered based on whether or how you configure the execution schedule.

If a field update event action is also scheduled along with outbound messages event action, the field updates event action is triggered first, so that the outbound messages contain the updated data.

Security Considerations

The outbound messages can use either an authentication-only client-side security policy or a transport-level security policy that protects the message during transfer. The default authentication-only policy used by object workflow outbound message is `oracle/wss10_saml_token_client_policy`. This policy includes Security Assertion Markup Language (SAML) tokens in outbound Simple Object Access Protocol (SOAP) request messages. This policy should only be used when the target web service is located within a secure network segment.

The corresponding service can use any compatible service policy, such as `oracle/wss10_saml_token_service_policy` Or `oracle/wss_saml_or_username_token_service`.

When the service is outside your firewall, you must protect the message by selecting the **Protect Message** option on the Create Action: Outbound Message page. When protected, the outbound message uses `oracle/wss_username_token_over_ssl_client_policy` with one-way secure socket layer (SSL) enabled by default. The corresponding service must use a compatible service policy, such as `oracle/wss_username_token_over_ssl_service_policy`.

Outbound Messages Protection

You must perform the following setup activities when you are using the **Protect Message** option:

1. Your applications require credentials to access the remote system to which the outbound message is being sent. Log a service request (SR) to provide these credentials to access the remote system.
2. If the remote system is using a self-signed SSL certificate, log an SR for the Oracle administrator to import the SSL certificate.

Related Topics

- [About Object Workflows](#)
- [Examples of Object Workflows](#)
- [Configure External Web Services](#)

Configure External Web Services

You can configure an object workflow to send an outbound message to a third-party Web service at a specified endpoint URL. An endpoint URL is an external Web service that receives data from your applications.

The third-party Web service must conform to the service WSDL defined by Oracle.

To build the external Web service, use the `OutboundMessageService.wsdl` and `.xsd` files. Then, when defining the outbound message event action for the object workflow, you provide the endpoint URL of the external Web service.

This topic provides the following:

- How to configure a Web service
- An example of the **OutboundMessageService.wsdl**

- An example of the **OutboundMessageService.xsd**
- The security policies available

Configuring a Web Service

To configure a Web service, you must replace the parameters in the **OutboundMessageService.xsd** file using the Web service instructions.

For more information about Web services, see the SOAP Web Services guide.

1. For a standard object, search the guide by object name.

For custom objects, search for the generic Web service for all custom objects in the corresponding application, for example, Sales Custom Business Object.

2. Extract the **.xsd** files from the live environment URL using the information provided under the service.
3. Replace the parameters in **OutboundMessageService.xsd** with the names for the object of your interest.

The parameters are marked in the **.xsd** file as parameters, `$OBJECT_TARGET_NAMESPACE$` and `$OBJECT_NAME$`.

Examples of **OutboundMessageService.wsdl** and **OutboundMessageService.xsd** are provided here for reference.

WSDL File Example

This section contains an example of the **OutboundMessageService.wsdl**, for your reference.

```
//Sample OutboundMessageService.wsdl
<wsdl:definitions
  name="OutboundMessageService"
  targetNamespace="http://xmlns.oracle.com/apps/crmCommon/content/outboundMessage/"
  xmlns:errors="http://xmlns.oracle.com/adf/svc/errors/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://xmlns.oracle.com/apps/crmCommon/content/outboundMessage/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:types="http://xmlns.oracle.com/apps/crmCommon/content/outboundMessage/types/"
  >
  <wsdl:import namespace="http://xmlns.oracle.com/adf/svc/errors/" location="ServiceException.wsdl"/>
  <wsdl:types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://xmlns.oracle.com/apps/crmCommon/content/outboundMessage/types/"
  schemaLocation="OutboundMessageService.xsd"/>
  </schema>
  </wsdl:types>
  <wsdl:message name="OutboundMessageService_processOutboundMessage">
  <wsdl:part name="parameters" element="types:processOutboundMessage"/>
  </wsdl:message>
  <wsdl:message name="OutboundMessageService_processOutboundMessageResponse">
  <wsdl:part name="parameters" element="types:processOutboundMessageResponse"/>
  </wsdl:message>
  <wsdl:portType name="OutboundMessageService">
  <wsdl:documentation/>
  <wsdl:operation name="processOutboundMessage">
  <wsdl:input message="tns:OutboundMessageService_processOutboundMessage"/>
  <wsdl:output message="tns:OutboundMessageService_processOutboundMessageResponse"/>
  <wsdl:fault name="ServiceException" message="errors:ServiceException"/>
  </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="OutboundMessageServiceSoapHttp" type="tns:OutboundMessageService">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="processOutboundMessage">
```

```
<soap:operation soapAction="http://xmlns.oracle.com/apps/crmCommon/content/outboundMessage/
processOutboundMessage"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
<wsdl:fault name="ServiceException">
<soap:fault name="ServiceException" use="literal" encodingStyle=""/>
</wsdl:fault>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="OutboundMessageService">
<wsdl:port name="OutboundMessageServiceSoapHttpPort" binding="tns:OutboundMessageServiceSoapHttp">
<soap:address location="http://adc2111013:7101/OMInterface/OutboundMessageService"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

XSD File Example

This section contains an example of the **OutboundMessageService.xsd**, for your reference. The parameters are marked in the .xsd file as parameters, \$OBJECT_TARGET_NAMESPACE\$ and \$OBJECT_NAME\$.

```
//Sample: OutboundMessageService.xsd

<schema elementFormDefault="qualified" targetNamespace="http://xmlns.oracle.com/apps/crmCommon/content/
outboundMessage/types/"
xmlns:ns0="http://xmlns.oracle.com/adf/svc/errors/" xmlns:ns1="$OBJECT_TARGET_NAMESPACE$"
xmlns:ns2="http://xmlns.oracle.com/adf/svc/types/" xmlns:tns="http://xmlns.oracle.com/apps/crmCommon/
content/outboundMessage/types/"
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://xmlns.oracle.com/adf/svc/types/" schemaLocation="BC4JService.xsd"/>
<import namespace="$OBJECT_TARGET_NAMESPACE$" schemaLocation="$OBJECT_NAME$.xsd"/>
<import namespace="http://xmlns.oracle.com/adf/svc/errors/" schemaLocation="ServiceException.xsd"/>
<element name="processOutboundMessage">
<complexType>
<sequence>
<element name="object" type="ns1:$OBJECT_NAME$"/>
</sequence>
</complexType>
</element>
<element name="processOutboundMessageResponse">
<complexType>
<sequence/>
</complexType>
</element>
</schema>
```

Security Policy

The outbound messages can use either an authentication-only client-side security policy or a transport-level security policy that protects the message during transfer. The default authentication-only policy used by object workflow outbound message is `oracle/wss10_saml_token_client_policy`. This policy includes Security Assertion Markup Language (SAML) tokens in outbound Simple Object Access Protocol (SOAP) request messages. This policy should only be used when the target web service is located within a secure network segment.

The corresponding service can use any compatible service policy, such as `oracle/wss10_saml_token_service_policy` Or `oracle/wss_saml_or_username_token_service`.

When the service is outside your firewall, you must protect the message by selecting the **Protect Message** option on the Create Action: Outbound Message page. When protected, the outbound message uses `oracle/wss_username_token_over_ssl_client_policy` with one-way secure socket layer (SSL) enabled by default. The corresponding service must use a compatible service policy, such as `oracle/wss_username_token_over_ssl_service_policy`.

Related Topics

- [Send Outbound Messages to Web Services](#)
- [Oracle Fusion Cloud Customer Experience SOAP Web Services for Sales and Fusion Service](#)

Object Workflows and Business Processes

Object Workflows and Business Processes

This topic explains how to use object workflows to trigger business process flows. When you configure object workflows, you also specify the actions that the workflow must perform when triggered. One of the actions you configure is the Business Process Flow action.

Before you begin, you might want to read "About Object Workflows" and its related topics to familiarize yourself with what object workflows are, how to set trigger conditions, and the event actions that object workflows support.

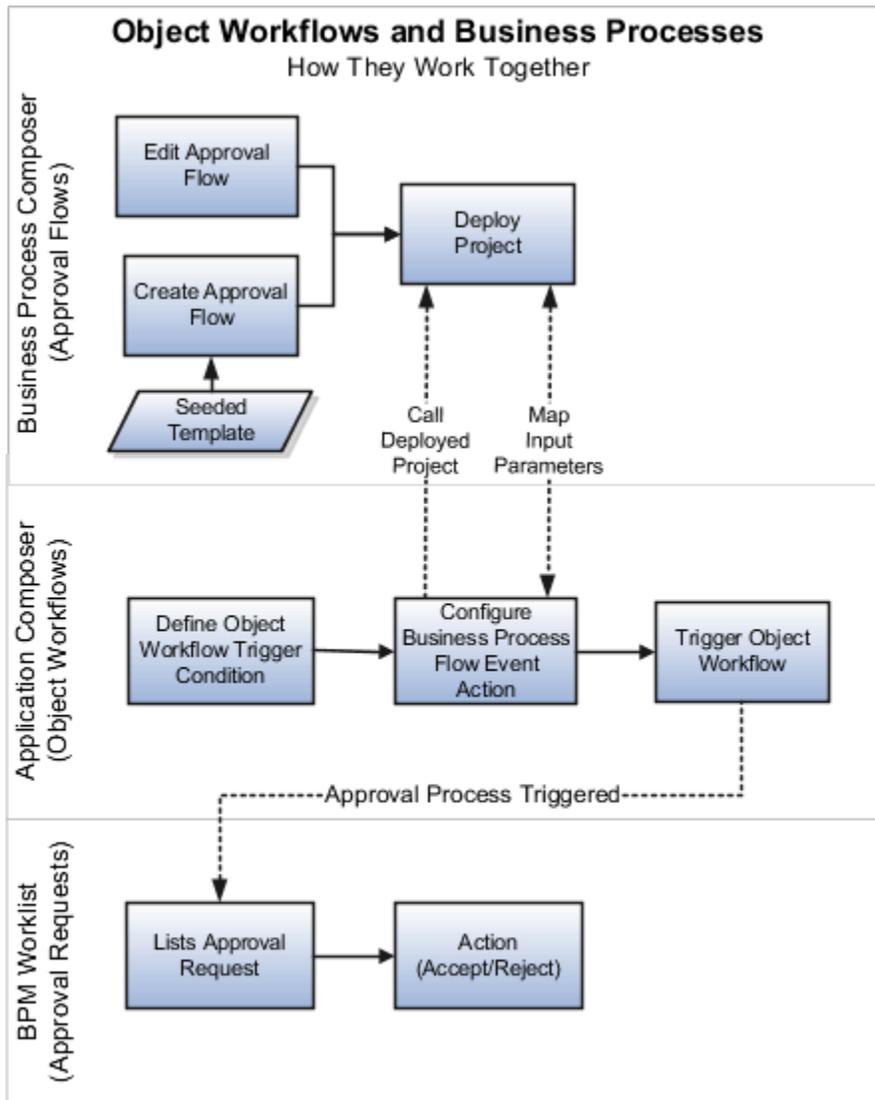
Overview

A business process flow in object workflows is essentially an approval flow. You use these approval flows to send approval requests, for example, when the following events occur:

- When the win probability of an opportunity is changed above a threshold level.
- When an object you created has been updated.
- When a new order is created.

First, create or update approval flow in Oracle Business Process Management (BPM) Composer. Then, select that approval flow in object workflow business process flow action.

This figure illustrates how object workflows and business processes work together, and the various applications involved in the end-to-end flow.



- Application Composer. Use this application to configure object workflows.

For examples of how to configure various event actions in object workflows, see "Configure Object Workflows" and "Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads" topics.

- Oracle Business Process Management (BPM) Composer. Use this application to create business processes (or approval flows) using existing or user-defined templates.

Note: Modifying templates in BPM Composer might be disabled for your implementation in this release. If so, contact Oracle Support for assistance.

For an example of how to create and deploy a business process flow, see [Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads](#) topic.

For more information on BPM Composer, refer to the Oracle Fusion Middleware Business Process Composer User's Guide for Oracle Business Process Management.

- Oracle Business Process Management (BPM) Worklist application. Use this application to take an appropriate action on the approval requests.

The Worklist application displays tasks or approvals that are assigned to a user. Your worklist tasks appear on the Home page.

Create an Approval Flow Using BPM Composer

You can create an approval flow using BPM Composer.

Oracle provides a standard set of business processes that you can use as is or as templates to create your own version of the processes. You typically create approval flows using the `ExtnBusinessProcessComposite` template, which is the standard template delivered with the product. If there are other templates available, create your approval flow based on the relevant template.

The `ExtnBusinessProcessComposite` template supports only Business Process Management Notation (BPMN), which is an industry standard notation for defining business processes. For more information on BPMN, see <http://www.bpmn.org>.

To access BPM Composer, click the Business Processes link on the Overview page of Application Composer.

To learn how to use the BPM Composer, see ["Configure Business Processes for Object Workflows."](#)

Note: Modifying templates in BPM Composer might be disabled for your implementation in this release. If so, contact Oracle Support for assistance.

You can only use the existing templates (either as provided or by copying and editing them) to create projects using the services, tasks, and business rules as provided in the template. For more information on the considerations involved when using templates, see ["Object Workflows and Business Processes"](#).

For an example of how to create an approval flow using `ExtnBusinessProcessComposite` template, see ["Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads"](#).

Configure a Business Process Flow Action in Object Workflows

Object workflows contain a set of actions that are run when the workflow is triggered. One of these actions is the business process flow action.

To configure a business process flow action:

1. Navigate to Application Composer.
2. On the Overview page, click **Object Workflows**.
3. Select **Actions > Create**.

The Create Object Workflow page opens.

4. Specify a **Name** for your workflow.
5. Specify the trigger condition in the Event Point and Condition region.
6. Under the Actions region, click **Create** for Business Process Flow.

The Create Action: Business Process Flow page opens. Use this page to configure the business process flow action, as follows:

- a. Provide a meaningful **Name** for your business process flow action.
- b. Click the Search icon in the **Project Name** field to open Search and Select: Business Process Flow dialog.
- c. On the search and select dialog, select the project (or approval flow) that will trigger when the object workflow is triggered.
- d. Specify the input parameters that you want to supply to the approval process. See "Create an Approval Flow Using BPM Composer" section in this topic.

Configure Input Parameters

While configuring the business process flow action, you use the input parameters to specify the field values to pass from the object workflow to the chosen approval flow when the object workflow is triggered.

The input parameters that you pass from object workflows to the approval flow depends on the template that you have chosen for creating the approval flow. When you use the default ExtnBusinessProcessComposite template to create an approval flow, here's how you typically configure the input parameters:

Parameter	Description
approvers	<p>Specify the people who should receive the approval request. You can specify one or more of the following:</p> <ul style="list-style-type: none"> • Resources associated with the record (such as its owner or creator, the manager of the owner, or the last person to update the record). • Resource groups (such as the direct reports or subordinates of the owner). • Specific users. • Fields on record: If the selected object contains user-defined text fields that store the approver's GUID, you can select those fields to send approval notifications to a single user or comma-separated GUIDs of multiple users. <p>Note: The user must be a resource to be used as an approver.</p>
heldEntityStatusField	<p>Specify the field that displays the status of the approval request.</p> <p>Note: Make sure to select the field from the list of values and select Literal. Don't type or paste the value.</p>
emailAddress	<p>Optionally, specify the e-mail address of one or more individuals who must receive e-mails on the outcome of the approver action.</p>
title1	<p>Optionally, add more information to the title of the notification.</p>

Parameter	Description
title2	Optionally, add more information to the title of the notification.

For more information on the considerations involved when using these input parameters, Object Workflows and Business Processes.

What Happens When the Object Workflow is Triggered

When the object workflow is triggered, the approval flow is also triggered. For example, when the approval flow that you created using the ExtnBusinessProcessComposite template is triggered, a task appears in the BPM Worklist application for the person specified in the approver field under input parameters.

You can also configure these input parameters to modify some of the notifications, but with limitations.

Note: Along with the modified notification, the approver may also receive the standard email notification. There's no configurable way to stop receiving the standard notification.

For more information on the types of notifications available and how you can modify them, Object Workflows and Business Processes.

Related Topics

- [Configure Business Processes for Object Workflows](#)
- [About Object Workflows](#)
- [Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads](#)

Configure Business Processes for Object Workflows

This topic covers the key points to consider when you work with object workflows to trigger a business process. When you configure object workflows, you also specify the actions that the workflow must perform when triggered.

One of the actions you configure is the Business Process Flow action. This business process flow action in object workflows is essentially an approval flow. You configure the approval flow using BPM Composer.

Overview

You use templates in BPM Composer to create and deploy a project, which you then call from object workflows. A project that you create and deploy is called an approval flow. You must create and deploy at least one project (or approval flow) before you configure the Business Process Flow action in object workflows.

Note: You must work directly in the mainline metadata to create an approval flow. You can't configure the business process flow task while you're in an active sandbox. For each approval flow that you create, keep track of your configurations. Whenever you update BPM Composer, you must reenter those same configurations to ensure that your approval flows still work smoothly.

First, create or update approval flow in BPM Composer. Then, select that approval flow in object workflow business process flow action.

- Application Composer. Use this application to configure object workflows.

For examples of how to configure various event actions in object workflows, see "Configure Object Workflows" and "Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads" topics.

- Oracle Business Process Management (BPM) Composer. Use this application to create business processes (or approval flows) using existing or user-defined templates.

Note: Modifying templates in BPM Composer might be disabled for your implementation in this release. If so, contact Oracle Support for assistance.

For an example of how to create and deploy a business process flow, see Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads topic.

For more information on BPM Composer, refer to the Oracle Fusion Middleware Business Process Composer User's Guide for Oracle Business Process Management.

- Oracle Business Process Management (BPM) Worklist application. Use this application to take an appropriate action on the approval requests.

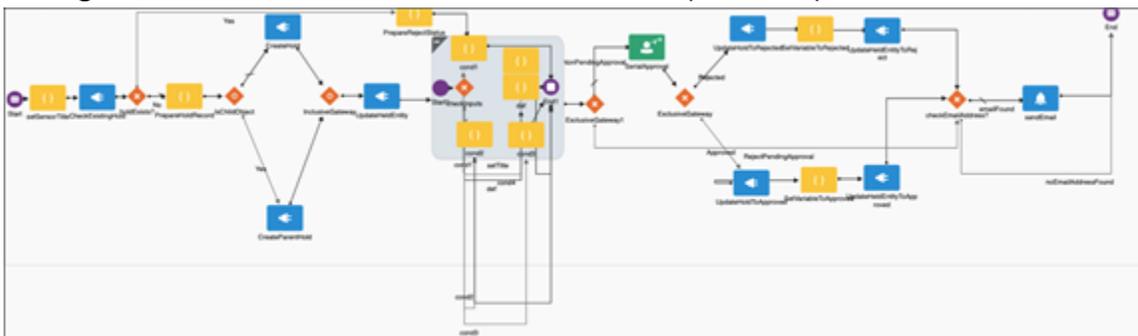
The Worklist application displays tasks or approvals that are assigned to a user. Your worklist tasks appear on the Home page.

Use the Default Template to Create Approval Process

The template delivered with the product is the ExtnBusinessProcessComposite template, which uses a basic CrmCommonSerialApprovalProcess process flow. This template contains an approval process covering a typical sales use case. Any project that you create with the seeded template uses the CrmCommonSerialApprovalProcess process.

The ExtnBusinessProcessComposite template supports only Business Process Management Notation (BPMN), which is an industry standard notation for defining business processes. For more information on BPMN, see <http://www.bpmn.org>.

This figure shows the standard ExtnBusinessProcessComposite template.



For an example of how you create and deploy a project, see "Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads".

The default ExtnBusinessProcessComposite template contains services, business rules, and tasks which you can use for configuring approval flows.

Note: Modifying templates in BPM Composer might be disabled for your implementation in this release. If so, contact Oracle Support for assistance.

You can change or add nodes or business rules for the human task implementation process. For example, you can change `SerialGivenUserNameApproval` with `SerialApprovalGroupApproval`. However, when creating or modifying a project, don't attempt the following unless you're sure:

- Deleting or modifying the services, rules, tasks, or system fields in an existing business process. It may fail validations.
- Removing or changing reserved parameters such as `owner`, `heldEntity`, `heldEntityId`, `heldEntityStatusField`, and `objectProperties`.
- Removing nodes such as `CreateHold`, `UpdateHold`, `UpdateEntity`, and so on.

Note: You can edit only those projects that have been created by launching BPM Composer from within Application Composer.

To see the services, tasks, and rules available, select the Project Home tab in BPM Composer.

Note that these human task patterns, however, aren't supported:

- `SerialSupervisorHierarchyApproval`, `SerialJobLevelHierarchyApproval`, `SerialPositionHierarchyApproval`
- `ParallelSupervisorHierarchyApproval`, `ParallelJobLevelHierarchyApproval`, `ParallelPositionHierarchyApproval`

For information on working with business rules, services, and tasks and how to model process flows, refer to the Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management.

Supply Inputs to the Business Process from Object Workflows

Using input parameters, you can pass specific values to the associated approval process when the trigger condition is met. These input parameters are specific to the template that you're using to create your approval flow. By default, you use the `ExtnBusinessProcessComposite` template, which is shipped with the product and covered in the documentation related to object workflows. If you're using some other template, refer to the documentation for that product for the parameters that you can use.

To see where these input parameters reside in BPM Composer, right click on the Start node of the approval flow and select Properties.

Notice the `heldEntityStatusField` and `approvers` parameters in the figure. These are some of the parameters you're mapping to from object workflows when using the `ExtnBusinessProcessComposite` template.

When using the standard `ExtnBusinessProcessComposite` template, these are the inputs you can supply to the associated approval process:

- `approvers`
- `heldEntityStatusField`
- `emailAddress`
- `title1`
- `title2`

While the `approvers` and `heldEntityStatusField` parameters are mandatory, the `emailAddress`, `title1`, and `title2` are optional parameters.

For more information on these inputs and how you can configure notifications using these inputs, see "Configuring Input Parameters for Modifying Notifications" section in this topic.

Using input parameters, you can pass either static or runtime values to the approval process as follows:

- To pass actual or static values to the approval flow, select either **Literal** check box in the Select Default Value dialog or manually enter parameters in single quotes, for example, 'user1'.

This figure illustrates how you select and mark the input parameters as Literal. This dialog appears when you click on the function icon in the Inputs region.

- To pass runtime values, don't select Literal in the Select Default Value dialog or just enter the parameters manually without quotes.

For example, to pass a runtime field (token) for Opportunity Region, enter Region (without quotes). When the workflow is triggered, this field is replaced with runtime value of Region.

You can also obtain approvals serially or in parallel from multiple approvers. For example, to obtain serial approvals first from user1 and then from user2 using Literal values, enter 'user1,user2'. Similarly, for runtime fields, enter user1,user2.

Note: To supply multiple, literal values use single quotes for the entire string without any space in between those values.

Configure Input Parameters for Modifying Notifications

The field values or the input parameters that you pass from object workflows to the approval flow depends on the template that you have chosen for creating the approval flow. This section contains examples assuming you're using the default ExtnBusinessProcessComposite template.

Before we go into configuring input parameters, it's essential to understand what types of notifications occur when the approval flow is triggered, and when the approval flow is approved or rejected.

- When the approval flow is triggered, the BPM worklist notification shows, for example, "Opportunity has been submitted for your approval", which is in format "<Object name> has been submitted for your approval".
- When the approval or rejection happens, the e-mail notification (based on the emailAddress parameter) contains the outcome of the action taken by the approver. The subject of the e-mail can be, for example, "Opportunity has been approved".

You can only minimally change the way the BPM Worklist and e-mail notifications appear by appending information using some of the input parameters.

To change the BPM notification, for example, let's assume you're creating a workflow using the Opportunity object and have specified only the approvers and heldEntityStatusField parameters. When such an approval flow is triggered, the notification in BPM Worklist application shows "Opportunity has been submitted for your approval", which is a standard notification.

Now, let's assume that you also want the opportunity name and region to appear in the notification. Assuming that the opportunity is Pinnacle Deal and the region is West, you can change the BPM notification to a more meaningful one by configuring the parameters as follows:

Parameter	Values	Description	Field Type
approvers	'Matt Hooper, Manager of Created By, Resource Group: Direct Reports of Created By'	Specifies one or more people who receive the approval request.	Run Time
heldEntityStatusField	'Approvalstatus_c'	Specifies the field that displays the status of the approval request. Note: Make sure to select the field from the list of values and select Literal . Don't type or paste the value.	Literal
emailAddress [Optional]	'jsmith@pinnacle.com, blopez@pinnacle.com'	Specifies e-mail addresses of one or more individuals who receive the approval or rejection notification.	Literal
title1 [Optional]	Name	Appends the name of the opportunity to the title.	Run time
title2 [Optional]	Region	Appends the name of the region (of the opportunity) to the title.	Run time

Now trigger the approval flow again. This time the notification in the BPM Worklist application shows "Opportunity Pinnacle Deal West has been submitted for your approval," which follows the format "<Object name> <title1> <title2> has been submitted for your approval".

Similarly, when an approver takes an action on the request, the title1 and title2 parameters are also appended to the e-mail notification. For example, when the request is approved, the subject of the e-mail shows "Opportunity Pinnacle Deal West has been approved".

To display the notifications in a different format, you can specify only some of the optional parameters.

Note: Along with the modified notification, the approver of the BPM may also receive the standard email notification. There's no configurable way to stop receiving the standard notification.

Specify the Trigger Condition in Object Workflows

You must ensure that the trigger condition for the object workflow contains the field that you update to trigger the workflow. For example, if you have defined your own field Submit_c of type check box, the trigger condition looks like: `isAttributeChanged('Submit_c') && Submit_c=='Y'`.

Defining an expression prevents the triggering of the object workflow each time an update or create event point occurs.

For more information on how to set trigger conditions, see [Configuring Object Workflows: Example topic](#).

If you're specifying a drop down, fixed-choice list field for capturing approval status, use the lookup type ZCX_HOLD_STATUS to configure the status values as per your business requirement. The default values provided in this

lookup type are: APPROVALPENDING, APPROVED, and REJECTED. Ensure that ZCX_HOLD_STATUS contains the default values or the values that you have specified in this lookup type for tracking the approval status.

Note: To capture the approval status, you can use only fields of type text or fixed choice list.

Related Topics

- [About Object Workflows](#)
- [Object Workflows and Business Processes](#)
- [Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads](#)

Set Up Serial Group Approval

You can create an approval workflow that requires approval from a group of people, in sequential order.

This is called serial group approval. In serial approval, a list of approvers must approve a process in order (the second person on the list can't approve until the first person has done so). You manage this using Oracle Business Process Manager (BPM) and object workflows.

This topic describes the following:

- Creating the group of approvers.
- Creating the approval flow.
- Deploying the project.
- Including the approval flow in an object workflow.
- Allowing the approval flow to complete early.

For more information about human tasks, refer to the Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.

Create the Group of Approvers

To set up serial group approval, you must first create the group for your approvers.

1. Sign in to the BPM worklist application.
2. Navigate to **Administration, Approval groups**.
3. Create a group named **CrmCommonApprovalGroup**. Note that you must give the group this name.
4. Add the members of the group (the people you want to designate as approvers).
The group can be static or dynamic.
5. Save the group.

Create the Approval Flow

Next, you create the approval flow, which is based on the standard approval flow template `ExtnBusinessProcessComposite`.

Note: Modifying templates in BPM Composer might be disabled for your implementation in this release. If so, contact Oracle Support for assistance.

1. Click **New** on the Business Processes page.

2. Select **ExtnBusinessProcessComposite** as the base template.
3. From the component palette on the right side of BPM Composer, drag an interactive task step into the approval flow in the same place where you removed the existing **SerialApproval** step.
4. Rename the new task step (for example, **GroupApproval**).
5. Select the task, click the four lines icon against it and click **Open Properties**.
6. At the bottom of the screen, search for all patterns and select the **SerialApprovalGroupApproval** task.
7. Click **Apply changes**.
8. Select the task again, click the four lines icon against it and click **Open Data Association**.
9. Map the inputs and outputs as shown in the table below.

Inputs	GroupApproval	Outputs
taskTitle	title	Not applicable
taskOwner	taskOwner	Not applicable
taskOwner	errorAssignee	Not applicable
heldEntityIdStringVar	ObjectId	Not applicable
Not applicable	outcome	taskOutcome

10. Click **Apply**.
11. Save the process.

Deploy the Project

1. From the main menu, select **Deployment, Deploy Project**.
2. In the **Deploy Project** dialog box, enter the following:

Field	Value
Revision ID	Provide an identifiable ID.
Deployer User Name	Your sign-in name.
Password	Your password.
Mark composite revision as default	Select this option.

3. Click **Deploy**.
A confirmation message appears after the project has been deployed successfully.

Include the Approval Flow in an Object Workflow

Now you can create an approval workflow that includes a serial list of approvers. To do this, create an object workflow that uses the **GroupApproval** flow that you created above.

1. Make sure you're in an active sandbox.
2. In Application Composer, click **Object Workflows**.
3. Enter a name for the object workflow, and select an object (for example, Opportunity).
4. Click **Create**.
5. Click the **Create** icon next to **Business Process Flow**.
6. Enter a name for the flow.
7. Click **Search** next to **Project Name**.
8. Select **GroupApproval** from the list and click **OK**.
9. Click **Save**.

When the object workflow is invoked, the first member of the group receives the approval notification. After the first person approves the flow, it's routed to the second person in the group and so on.

The flow is considered complete when all the members of the approval group approve the request.

Allow the Approval Flow to Complete Early

In most cases, if the first approver rejects the request, then you will want the flow to immediately complete and not move on to the next approver. You can set up your approval flow to do this by configuring the Early completion settings in BPM Worklist.

1. Click the **Notifications** icon on the global header.
2. Click **More Details**.
3. In BPM Worklist, click your user name and select **Administration**.
4. Click **Task Configuration** at the top of the page.
5. In the Search box, search for task you're using for the object workflow, such as **SerialApprovalGroupApproval**.
6. Click the **Edit task** pencil icon, then click the Configuration tab.
7. Select the **Complete task immediately when participant chooses** check box, and select **Reject**.
8. Click the **Save** icon and optionally enter comments.
9. Click the **Commit task** icon and optionally enter comments.

Set Up Parallel Approval

You can create an approval workflow that requires approval from a group of people, in any order.

This is called parallel group approval. In parallel approval, all members of a list of approvers must approve a process in any order (the order of approvers in the list isn't relevant, but all approvers must approve it). You manage this using Oracle Business Process Manager (BPM) and object workflows.

This topic describes the following:

- Creating the group of approvers.
- Creating the approval flow.
- Deploying the project.

- Including the approval flow in an object workflow.

For more information about human tasks, refer to the Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.

Create the Group of Approvers

To set up parallel group approval, you must first create the group for your approvers.

1. Sign in to the BPM worklist application.
2. Navigate to **Administration, Approval groups**.
3. Create a group named **CRMCommonApprovalGroup**. Note that you must give the group this name.
4. Add the members of the group (the people you want to designate as approvers).
5. Save the group.

Create the Approval Flow

Next, you create the approval flow, which is based on the standard approval flow template `ExtnBusinessProcessComposite`.

Note: Modifying templates in BPM Composer might be disabled for your implementation in this release. If so, contact Oracle Support for assistance.

1. Click **New** on the Business Processes page.
2. Select **ExtnBusinessProcessComposite** as the base template.
3. Remove the standard human task step named **SerialApproval**.
4. From the component palette on the right side of BPM Composer, drag an interactive task step into the approval flow in the same place where you removed the existing **SerialApproval** step.
5. Rename the new task step (for example, **ParallelApproval**).
6. Select the task, click the four lines icon against it and click **Open Properties**.
7. At the bottom of the screen, search for all patterns and select the **ParallelApprovalGroupApproval** task.
8. Click **Apply changes**.
9. Select the task again, click the four lines icon against it and click **Open Data Association**.
10. Map the inputs and outputs as shown in the table below.

Inputs	GroupApproval	Outputs
taskTitle	title	Not applicable
taskOwner	taskOwner	Not applicable
taskOwner	errorAssignee	Not applicable
"REJECT"	defaultOutcome	Not applicable
100	VotingPercentage	Not applicable
Not applicable	outcome	taskOutcome

Inputs	GroupApproval	Outputs
parentIdVar	parentObjectId	Not applicable
heldEntityIdStringVar	ObjectId	Not applicable

In parallel approval, you must pay attention to the parameter **votingPercentage**. This parameter determines how many group members need to approve the request for the record to be considered approved. By setting the parameter to be 100, you specify that every approver in the list must approve the request.

11. Click **Apply**.
12. Save the process.

Deploy the Project

Finally, you deploy the project.

1. From the main menu, select **Deployment, Deploy Project**.
2. In the **Deploy Project** dialog box, enter the following:

Field	Value
Revision ID	Provide an identifiable ID.
Deployer User Name	Your sign-in name.
Password	Your password.
Mark composite revision as default	Select this option.

3. Click **Deploy**.

A confirmation message appears after the project has been deployed successfully.

Include the Approval Flow in an Object Workflow

Now you can create an approval workflow that includes a parallel list of approvers. To do this, create an object workflow that uses the **ParallelApproval** flow that you created above.

1. Make sure you're in an active sandbox.
2. In Application Composer, click **Object Workflows**.
3. Enter a name for the object workflow, and select an object (for example, Opportunity).
4. Click **Create**.
5. Click the **Create** icon next to **Business Process Flow**.
6. Enter a name for the flow.

7. Click **Search** next to **Project Name**.
8. Select **GroupApproval** from the list and click **OK**.
9. Click **Save**.

When the object workflow is triggered, all the users that are part of **CrmCommonApprovalGroup** will see the approval request. Based on the voting percentage, the outcome of the approval task is determined. In this example, because we mapped **votingPercentage** to 100, all the members of the group must approve the task before the record is finally approved.

The flow is considered complete when the specified percentage of members of the approval group (in this case, 100%) approve the request. If any member of the group rejects the request, then the record's status is changed to Rejected.

Multiple Approvals on a Single Record

You can set up a record to request a single approval or multiple approvals. If you request multiple approvals, you can do it in one of two ways:

- Single approval status field
- Multiple approval status fields (one for each approval)

Single Approval Status Field for Multiple Approvals

If you use a single approval status field (for example, one named "Approval Status"), then each stage of the approval process is requested individually, one after the other. After each approval is requested and obtained, the field is set to Approved. When the record is resubmitted for the next approval, the field is set to Pending Approval and the process is repeated until all required approvals are obtained.

Using Multiple Approval Status Fields for Multiple Approvals

The best practice is to use multiple approval status fields, which makes it easier to differentiate each approval on the record. This is especially true if some business logic depends on each approval.

Multiple approvals are processed in serial fashion; you can submit a new approval only after the previous approval process on the record has been completed.

Here's an example of how you might set up multiple approval status fields.

1. Start by creating a fixed choice list (for example, Stage Status) including each record that will need approval. In this example, these are:
 - Technical Proposal
 - Financial Proposal
 - Contract
2. Next, create a fixed-choice list including each approval status (you can use the standard list values from ZCX_HOLD_STATUS):
 - Technical Approval Status
 - Financial Approval Status
 - Contract Approval Status
3. Create a new BPM process using Default Serial Approval Process.

4. Create three workflows with the following criteria:

- o `heldEntityStatusField = 'TechnicalApprovalStatus_c' OWF criteria: isAttributeChanged('StageStatus') && StageStatus = 'Technical Proposal'`
- o `heldEntityStatusField = 'TechnicalApprovalStatus_c' OWF criteria: isAttributeChanged('StageStatus') && StageStatus = 'Technical Proposal'`
- o `heldEntityStatusField = 'ContractApprovalStatus_c' OWF criteria: isAttributeChanged('StageStatus') && and StageStatus = 'Contract'`

For the first approval, the Sales officer prepares the Technical Proposal document with revenue lines and sends it for manager approval. After it is approved, the **Technical Approval Status** field is changed to Approved. The same document is submitted to the customer for signature.

For the second approval, the sales officer prepares the Financial Proposal document (which lists payment terms) and submits it for approval by the Regional Manager and the CEO. After it is approved, the **Financial Approval Status** field is changed to Approved. The same document is submitted to the customer for signature.

For the third approval, after several stages in Opportunity, the Contract document is prepared and sent for approval by the CEO and Regional Manager. After it is approved, the **Contract Approval Status** field is changed to Approved. The document is submitted to customer for contract activation. If the customer rejects the terms, revise the contract and repeat the third approval process.

Error Behavior

If multiple approvals are submitted in parallel, then error notifications should be sent to the opportunity owner, who can manually ensure that the approval is resubmitted after the current approval is completed.

Configure Object Workflows to Trigger Approval Requests for Creating Sales Leads

This topic contains an end-to-end example showing how you configure an object workflow to trigger approval requests for creating sales leads.

You're a sales administrator and your management has asked you to create a workflow to automate the business process as follows:

1. An approval process should be triggered when a sales representative creates a sales lead with a deal amount greater than USD 100000.
2. When the approval process is triggered, the sales manager of that representative should automatically receive a request for approving the deal.

You use three applications and two roles to configure and test the automated flows; therefore, for ease of understanding, the end-to-end procedure has been logically organized under the following steps:

1. Creating a field to track the approval status.
You use Application Composer to create a custom field to track the status of approvals.
2. Publishing the sandbox and verifying the addition of the field.
You publish the sandbox to bring the custom field into the mainline metadata.
3. Creating and deploying an approval flow.
You use BPM Composer to create and deploy an approval flow.

Note: Modifying templates in BPM Composer might be disabled for your implementation in this release. If so, contact Oracle Support for assistance.

4. Creating object workflow to trigger the approval flow.
You create an object workflow using the Sales Lead object and set the trigger condition on the Deal Amount field.
5. Triggering and testing the object workflow and approval flow.
You trigger the object workflow, test the triggering of the associated approval flow, approve the request using Worklist, and then verify the approval.

Step 1: Create a Field to Track the Status of Approvals

In this step, you create a field to track the status of the approval and add this field to the desired pages.

1. Sign in using Administrator role.
2. Create a sandbox and activate it.
For more information on sandboxes, see Sandboxes: Explained topic.
3. Navigate to Application Composer.
4. In the left pane, expand **Standard Objects**.
5. Expand **Sales Lead**.
6. Click **Fields**.
Fields page opens.
7. Under the Custom tab, select **Action > Create**.
Select Field Type dialog box appears.
8. Select **Text**.
9. Click **OK**.
Create Text Field page opens.
10. In the Appearance region, enter **Large Deal Approval** in the **Display Label** field.
11. Click **Save and Close**.
12. In the left pane, select **Pages** under **Sales Lead**.
Sales Lead: Pages page opens.
13. Under Desktop Pages tab, click **Create Lead**.
Create Lead page opens.
14. In the Configure Detail Form region, move the **Large Deal Approval** field from the **Available Fields** box to the **Selected Fields** box.
15. Click **Save and Close**.
You have now added a custom approval field to the Create Lead page. You're back to Sales Lead: Pages page.
16. Under Desktop Pages tab, click **Show Details**.
Show Details page opens.
17. In the Configure Detail Form region, move the **Large Deal Approval** field from the **Available Fields** box to the **Selected Fields** box.
18. Click **Save and Close**.
You have now added a custom approval field to the Show Details page.

Note: You're currently working within your sandbox. You have created a custom field and have added that field to two pages. Configurations done within a sandbox for object workflows aren't available to the mainline metadata unless published.

Next, publish your sandbox to make your changes available for use in the mainline metadata.

Step 2: Publish the Sandbox and Verify the Addition of the Field

In this step, you publish your sandbox and verify that the Large Deal Approval field has been added to Create Lead and Show Details pages.

For more information on creating and managing sandboxes, and the considerations involved when working with sandboxes, see *Sandboxes: Explained* topic and its related topics.

To publish the sandbox and verify the addition of the fields:

1. On the Sandboxes page, select your sandbox and click **Publish**.
2. Click the **Navigator** menu at the top.
3. Click **Lead Qualification** under **Sales**.
4. Click **Create Lead**.
Create Lead page opens.
5. Verify that **Large Deal Approval** field has been added to the Create Lead page.
6. Click **Cancel**. You create a deal later in this example.

Step 3: Create and Deploy an Approval Flow

In this step, you create and deploy a business process (also called project) for approval flow using BPM Composer.

Note: Modifying templates in BPM Composer might be disabled for your implementation in this release. If so, contact Oracle Support for assistance.

For more information on working with business process flows, see related topic *Object Workflows and Business Processes*.

To create and deploy a process using BPM Composer:

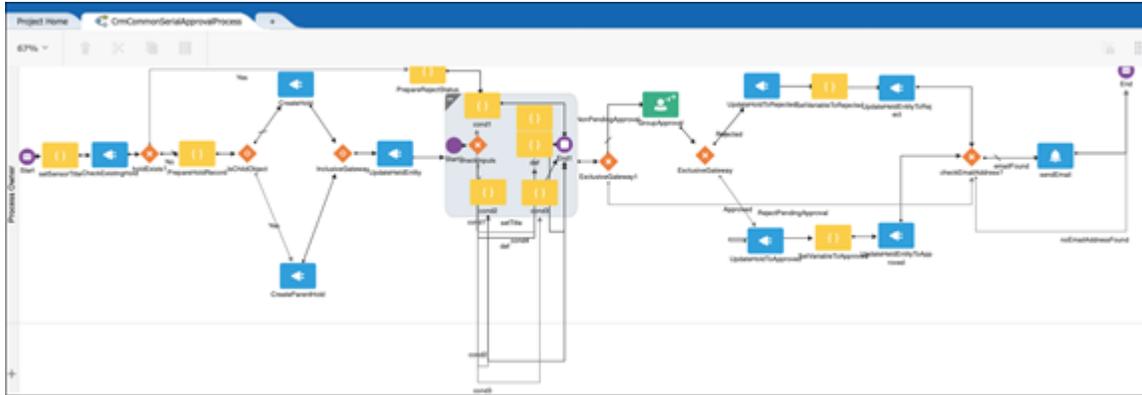
1. Navigate to Application Composer.
2. Select **Business Processes**.
Business Processes page opens.
3. Select **Actions > Create**.
The Create Business Process Flow dialog box appears.
4. Enter **Sales Lead Approval** in the **Name** field.
5. Click **OK**.
BPM Composer opens in a separate Window.

Note: If a blocker is enabled on your browser, BPM Composer may not open after you click OK, and your browser may get locked. If that happens, use the browser back button to go back to the Business Processes option in Application Composer, and click Edit for the business process (approval flow) that you were creating.

For this example, you're using a seeded template called ExtnBusinessProcess on an as is basis. You can edit this template to suit your business case.

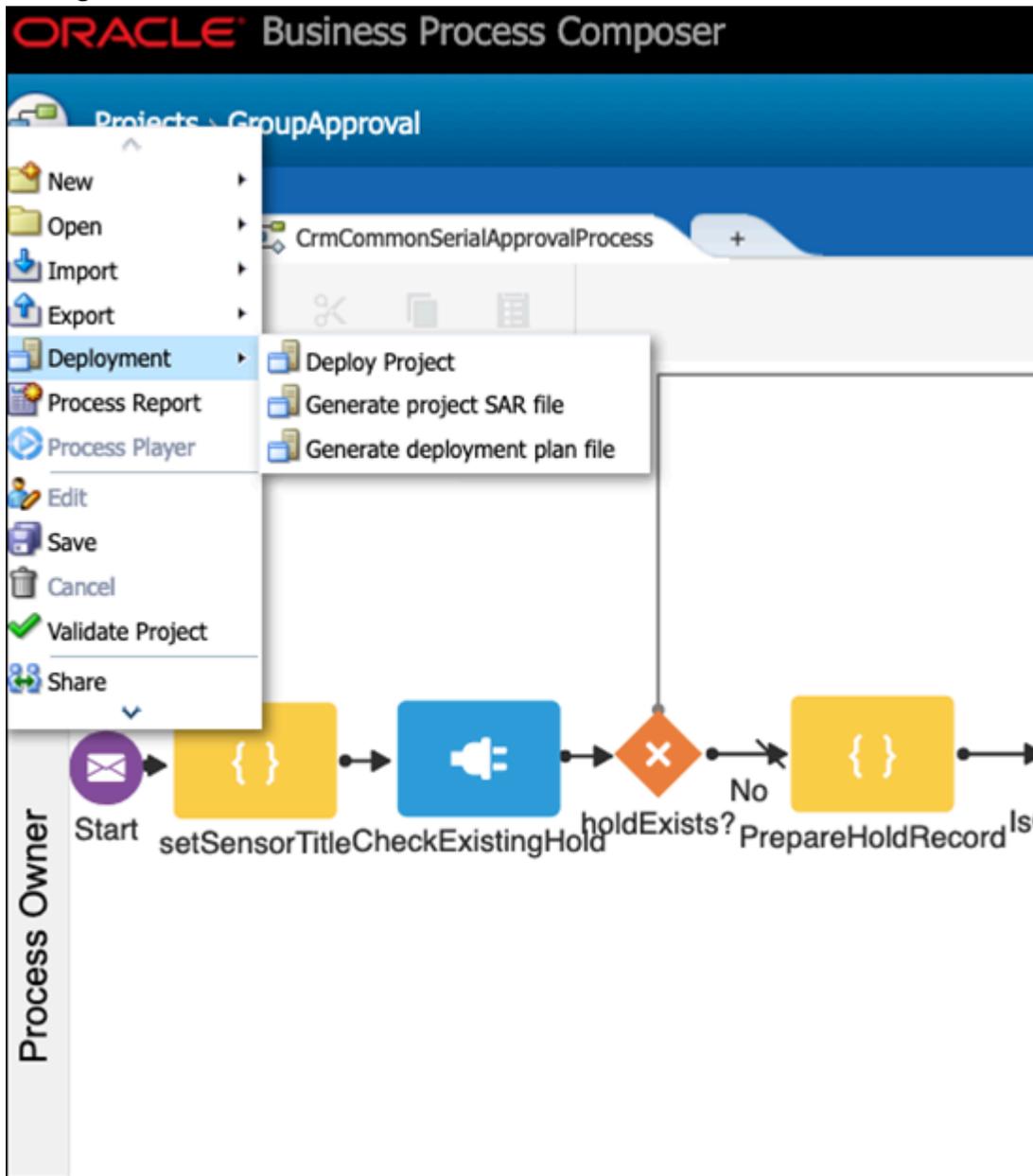
For more information on this template, see the *Object Workflows and Business Processes* topic.

This figure shows the seeded template.



6. Click **Deploy Project** using the main menu.

This figure shows the main menu.



The Deploy Project dialog box appears.

7. In the Deploy Project dialog box, enter the following information:

Field	Value
New Revision ID	Provide an identifiable ID.
Deployer Name	Your sign in name.

Field	Value
Password	Your password.
Mark composite revision as default	Select this option.
Version	Current project.

8. Click **Deploy**.
A confirmation message appears after your project has been deployed successfully.
9. Close BPM Composer.
10. Confirm as follows whether the project has been deployed successfully:
 - a. On the Overview page of Application Composer, click **Business Processes**.
The Business Processes page opens.
 - b. A green check-mark appears under the **Deployed** column for the project you created.

You have now successfully deployed a project with the name Sales Lead Approval.

Next, configure an object workflow to call the Sales Lead Approval project.

Step 4: Create Object Workflow to Trigger Approval Flow

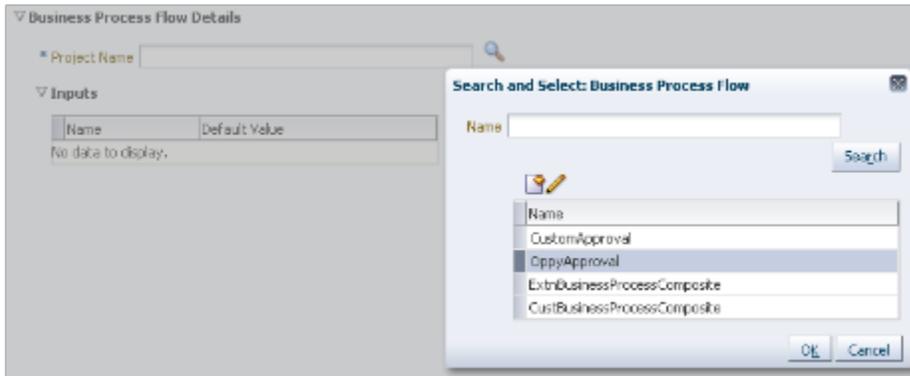
In this step, you configure an object workflow with the required trigger condition and specify the inputs to pass from object workflow to the Sales Lead Approval process.

To configure an object workflow:

1. Navigate to Application Composer.
2. On the Overview page, click **Object Workflows**.
The Object Workflow page opens.
3. Select **Actions > Create**.
The Create Object Workflow page opens.
4. Select **Sales Lead** from the **Object** list.
5. Enter **Deal amount more than USD 100000** in the **Name** field.
6. In the Event Point and Condition region:
 - a. Select **When a record is created** option.
 - b. Click the **Expression Builder** icon.
Expression Builder opens.
Enter `if(DealAmount>100000 && CurrencyCode=='USD') return true;`
 - c. Click **OK**.
7. In the Actions region, select **Business Process Flow**. Create Action: Business Process Flow page opens.
8. Configure the event action as follows:

- a. Enter **Sales Lead Approval** in the **Name** field.
- b. In the **Project Name** field, search and select **Sales Lead Approval** project that you had created and deployed.

This figure shows how you search and select a deployed business process (approval flow).



- c. Click **OK**.
- d. In the Inputs region, specify the input parameters that you want to pass to the deployed approval flow using the **Select Default Value** button. You can also type-in these parameters.

For more information on input parameters, see *Object Workflows and Business Processes* topic. You must mark these values as **Literal** to pass the values as is. Literal values appear enclosed in single quotes.

Note: In this example, we are mapping only the mandatory parameters. For more information on the available input parameters in the default template, see *Object Workflows and Business Processes* topic.

Name	Default Value	Description of the Parameter
approvers	'SALES_MANAGER'	One or more recipients of the approval request.
heldEntityStatusField	'LargeDealApproval_c'	The field that holds the status of the approval.

- 9. Click **Save** to save the event action.
- 10. Click **Save** to save the object workflow.

You have now configured the object workflow.

Next, test whether the object workflow calls the approval flow when the trigger condition is satisfied

Step 5: Trigger and Test the Object Workflow and Approval Flow

In this step, first trigger the object workflow and then test whether the approval flow runs fine.

- 1. Sign in using Sales Representative role.
- 2. Click the **Navigator** menu at the top.
- 3. Click **Lead Qualification** under **Sales**.

4. Click **Create Lead**.

Create Lead page opens.

5. Enter **New Deal** in the **Name** field.

6. In the **Deal Size** field, select **USD** and enter **120000**.

7. Click **Save and Close**.

Creating a lead with deal size (or amount) of more than USD 100000 triggers the object workflow, because you had set the trigger condition for object workflow as: `if(DealAmount>100000 && CurrencyCode=='USD') return true;`

To verify whether the workflow has been triggered, check the status in the approval field as follows:

a. Edit **New Deal** lead.

b. Note the status in the **Large Deal Approval** field, which is set to **APPROVALPENDING** by default.

You're currently signed in using Sales Representative role. You must first sign out and then sign in using Sales Manager role to approve the request.

8. Sign out, and then sign in using Sales Manager role.

A new task or request appears on the Home page under the Worklist region.

9. Approve the request.

You're currently signed in as a Sales Manager. You must first sign out and then sign in as a Sales Representative to verify whether the status in the **Large Deal Approval** field has changed.

10. Sign out, and then sign in using Sales Representative role.

11. Click the **Navigator** menu at the top.

12. Click **Lead Qualification** under **Sales**.

Overview page opens.

13. Under Leads tab, click **New Deal**.

Edit Lead: New Deal page opens.

14. Verify that the status in the **Large Deal Approval** field has been set to **APPROVED**.

You have now successfully triggered the object workflow to run the associated approval flow.

Related Topics

- [Configure Business Processes for Object Workflows](#)
- [Object Workflows and Business Processes](#)
- [About Object Workflows](#)
- [Overview of Sandboxes](#)

Configure an Approval Process for Opportunities

In this example, you learn how to set up approvals for opportunities using a business process flow provided by Oracle.

The approval process outlined in the diagram is triggered when a condition you enter as a Groovy script is true. In this example, the process is triggered when a salesperson enters an opportunity revenue amount greater than USD 100,000. Here's what happens when the process is triggered:

1. The approval process sends approval requests to one or more users.

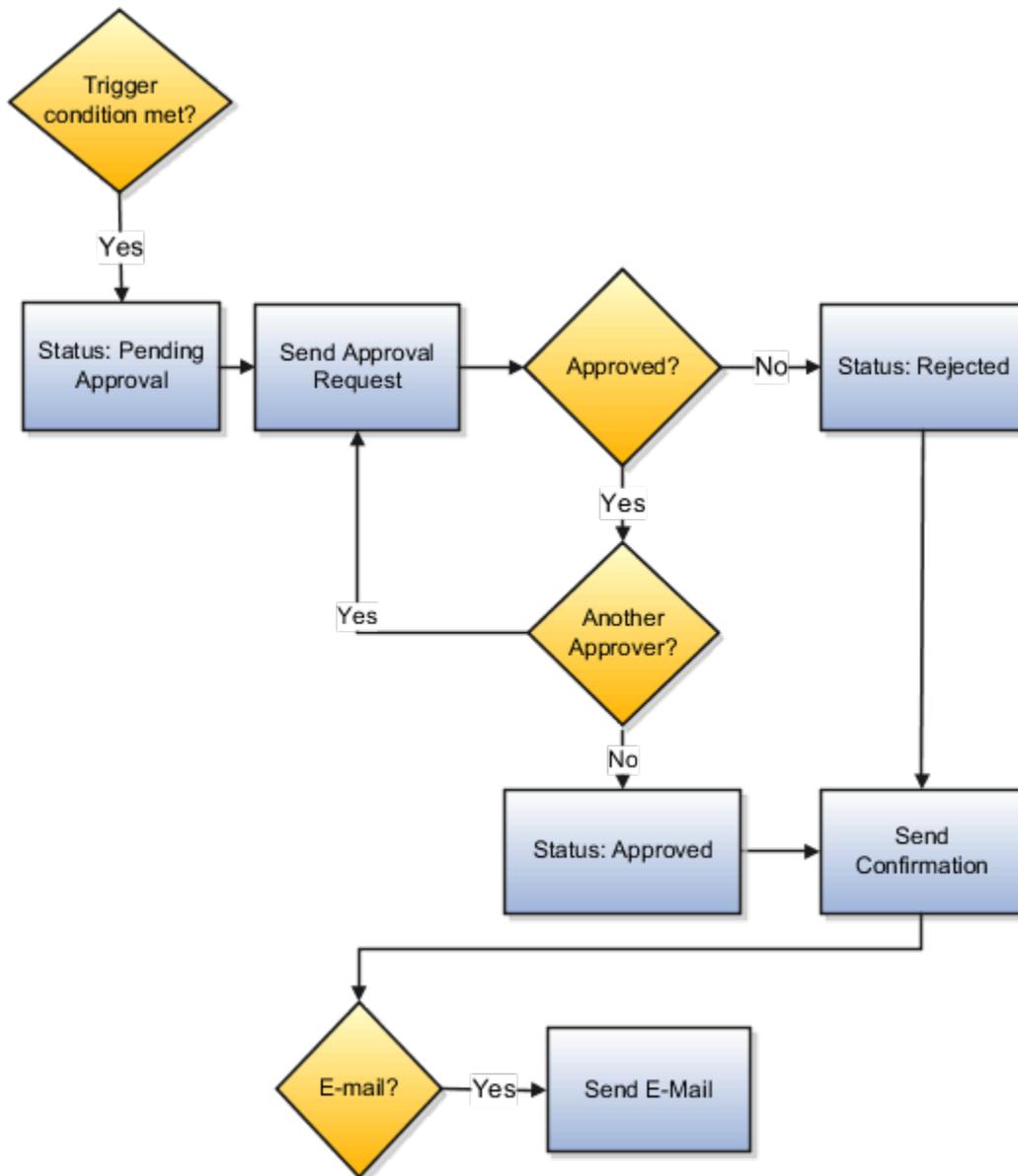
In this example, the notification is sent to one user. If you specify multiple users, then all users must approve before the opportunity is considered approved.

You can enter identifying information into the subject line of the approval notification to help approvers understand what they're approving. In this example, you enter the opportunity name and the account. The body of the notification includes the name of the person who updated the opportunity.

2. The approval process updates the approval status to PENDING APPROVAL to indicate that an approval request was sent.

In this example, you create a custom field to display the status on the Edit Opportunity page.

3. After the approver views the notification and clicks the Approve or the Reject button, the process updates the status to APPROVED or REJECTED and sends a confirmation back to the user who updated the opportunity.
4. Optionally, the process can send an informational e-mail about the process outcome to the recipients of your choice.



You create the object workflow that triggers the approval process using Application Composer while working in a sandbox, so you can test the process before publishing it.

The overall configuration of the approval process includes these steps:

1. Create a custom text field to display the approval status.
2. Add the custom field to the Edit Opportunity page.
3. Create an object workflow that triggers the approvals business process provided by Oracle and supplies the required inputs, including the name of the user who receives the approval notification.
4. Test the approval process before publishing it.

Create a Custom Field to Display the Approval Status

1. Sign in as a sales administrator or a setup user.
2. Activate an existing sandbox with Application Composer as one of the available tools, or create a new one.

The sandbox name appears at the top of the page.

3. Navigate to Application Composer.
4. In the left pane, expand **Standard Objects**.
5. Expand **Opportunity**.
6. Click **Fields**.
7. On the Fields page, Custom tab, select **Create** from the **Action** menu.
8. In the Select Field Type window, select **Text**.
9. Click **OK**.
10. On the Create Text Field page, in the **Display Label** field in the Appearance region, enter `Large Deal Approval`.

The applications creates the technical name and API Name based on your entry. The API name is the name followed by an underscore and the letter c (`_c`). You enter the API name as one of the parameters of the workflow to indicate this is the field that displays the approval status.

11. Click **Save and Close**.

You're now ready to display the custom field on the Edit Opportunity page.

Display the Custom Field on the Edit Opportunity Page

1. In the Application Composer left pane, select **Pages** under **Opportunity**.
2. On the Opportunity: Pages page, Application Pages tab, scroll down to Details Page Layouts section and edit any custom layout. If none exists, then duplicate the standard layout and edit the resulting custom layout.
3. In the Subtabs region, click **Edit** for the Summary.
4. In the Edit Summary page, move the **Large Deal Approval** field from the **Available Fields** box to the **Selected Fields** box.
5. Click **Save and Close**.
6. Click **Done**.
7. Click the **Application Composer** link at the top left corner of the screen to return to the Application Composer Overview page.

Create an Object Workflow to Trigger the Approval Flow

With the sandbox still active, configure an object workflow with the trigger condition and the inputs to pass to the approvals process.

1. On the Application Composer page, make sure that **Sales** is still selected as the **Application**.
2. Click **Object Workflows**.
3. On the Object Workflow page, select **Create** from the **Actions** menu.
4. On the Create Object Workflows page, select **Opportunity** from the **Object** list.
5. Enter **Deal amount more than USD 100000** in the **Name** field.

6. In the Event Point and Condition region:
 - a. Select the **When a record is updated** option.
 - b. Click the **Groovy Builder** icon, highlighted with callout 1 in the screenshot

In the Expression Builder window, enter the Groovy script: `if (nvl(Revenue,0)>100000 && isAttributeChanged('Revenue')) return true;`

You can ignore the warning.

- c. Click **OK**.
7. In the Actions region, click **Create** (the plus icon on the right side of the page) for **Business Process Flow**.
8. In the Create Action: Business Process Flow page, enter **Opportunity Approval** in the **Name** field.
9. In the **Project Name** field, search and select **ExtnBusinessProcessComposite**. This is the name of the approval business process provided by Oracle.
10. The Inputs region lists the parameters you can pass to the business process. In the Inputs region, make the following entries:
 - a. In the **approvers** field, enter the user name of the user who's to receive the approval notifications in single quotation marks. For example: `'lisa.jones'`.

- b. In the **heldEntityStatusField**, enter the API name of the custom field you created in single quotation marks. For example: `'LargeDealApproval_c'`.
- c. In the **emailAddress** field, enter the e-mail addresses in single quotations of people you want to be notified of the process outcome. Separate multiple e-mails with commas. For example:
`'JoanneWong@nomail.com', 'tanakas@nomail.com'`
- d. In the **title1** and **title2** fields, enter the two fields that you want to display in the subject line of the notification:

The notification subject line appears as: Opportunity <title1><title2> Is Submitted for Your Approval

In this example, we are adding the opportunity name and the account fields, so make the following entries in the **Default Value** field:

- For **title1**, enter `Name`.
- For **title2**, enter `AccountName`.

You can view all of the fields available for insertion, by clicking **Select Default Value**, the function icon to the right of the field, highlighted by callout 1 in the screenshot.

Create Action : Business Process Flow

Object Opportunity

Type Business Process Flow

* Name oppty approval

Description

▶ Execution Schedule

▲ Business Process Flow Details

* Project Name ExtnBusinessProcessComposite

▲ Inputs

Name	Default Value
approvers	'lisa.jones'
heldEntityStatus...	'LargeDealApproval_c'
emailAddress	
title1	Name
title2	AccountName

Here's how to insert a field:

- i. Select the field.
 - ii. Click **Insert** to move the field to the Value region.
 - iii. Click **OK**.
11. Click **Save and Close**.

Test the Approval Flow

To test the approval flow, create an opportunity and enter a revenue amount greater than USD 100,000. Then sign in as the approver to view the notification and approve.

1. While signed in as the same user who performed the setup, click the **Home** icon to return to the springboard.
2. Double-click on the **Sales** icon and on the **Opportunities** icon to open the Opportunities work area.

3. Click **Create Opportunity**.
4. In the Create Opportunity page, enter a name.
5. Click **Save and Continue**.
6. On the Edit Opportunity page, click **Add** in the Products region to add a revenue line to the opportunity.

Notice that the **Large Deal Approval** field you added to the page doesn't display a status. This is because the approval flow is not triggered yet.

7. Enter a product group or a product and an amount greater than 100000 to trigger the approval workflow.
8. Click **Save and Close**.

Saving triggers the approvals process.

9. Wait a few minutes to let the approval process run and view the opportunity again. The **Large Deal Approval** field now displays **PENDING APPROVAL**.
10. Sign out by clicking the user name and the **Sign Out** link.
11. Sign in as the approver.
12. Click **Notifications** (the bell icon) at the top of the screen.

The Recent Notifications window displays the new approval notification.

13. Click the notification subject line to display the notification body and click **Approve**.
14. Sign out and sign in again as the administrator.
15. Click **Notifications** (the bell icon) at the top of the screen.

The Recent Notifications window displays the approval confirmation.

16. Navigate back to the Opportunities work area and view the opportunity you created.
17. The **Large Deal Approval** field displays **APPROVED**.
18. You must publish the Sandbox if you want to make the opportunity approval available for use.

Object Workflows and Groovy Scripts

How Object Workflows and Groovy Scripts Work Together

You can use object workflows to trigger Groovy scripts. When you configure object workflows, you also specify the actions that the workflow must perform when triggered.

One of the actions you configure is the Groovy Script action. Use an object workflow with a Groovy Script action to perform long-running or performance-intensive Groovy scripts.

Before you begin, you might want to read *Object Workflows: Explained* and its related topics to familiarize yourself with what object workflows are, how you set trigger conditions, and the event actions that object workflows support.

Overview

A Groovy script action in an object workflow lets you call an external Groovy object function that has been previously defined. For example, you might trigger a Groovy action to perform cross-object updates, perform post-processing of large object hierarchies, or asynchronously call Web services.

As a simple example, let's assume that when you close an opportunity, you also want to close all the associated activities for that opportunity.

Using a trigger could cause a delay if the opportunity includes many associated activities, so instead you define an object workflow to trigger when the opportunity's status is updated to Lost. The object workflow calls a Groovy object function that queries the activities related the opportunity and sets their status to "Canceled."

Note: Typically, real-world processing logic used in Groovy script actions is much more complex than the simple example described here.

You configure a Groovy Script action in object workflows by defining an object function, then creating an object workflow that, when launched at run time, executes the object function. For examples of how you configure various event actions in object workflows, see [Configuring Object Workflows](#) and [Configuring Object Workflows to Trigger Approval Requests for Creating Sales Leads](#).

Configuring a Groovy Script Action in Object Workflows

To configure a Groovy Script action:

1. Navigate to Application Composer.
2. On the Overview page, click **Object Workflows**.
3. Select **Create** from the **Actions** menu.

The Create Object Workflow page opens.

4. Specify a name for your workflow.
5. Specify the trigger condition in the **Event Point and Condition** region.

6. Under the **Actions** region, click **Create** for Groovy Script.

The screenshot displays the 'Create Object Workflow' interface. At the top, there are 'Save and Close' and 'Cancel' buttons. Below them, the 'Object' is set to 'Acc_Child' and the 'Name' is 'Groovy Action'. A 'Description' text area is present but empty. The 'Event Point and Condition' section has 'When a record is created' selected. The 'Condition' field contains the example 'Status==IN_PROGRESS'. The 'Actions' list at the bottom includes 'Field Updates', 'E-Mail Notification', 'Task Creation', 'Outbound Message', 'Business Process Flow', and 'Groovy Script', each with a plus sign to expand it.

The Create Action: Groovy Script page opens. You use this page to specify the Groovy object function you want to call.

This figure shows the Create Action: Groovy Script page.

The screenshot shows the 'Create Action: Groovy Script' form. At the top right are 'Save' and 'Cancel' buttons. The form contains the following fields and sections:

- Object:** Acc_Child
- Type:** Groovy Script
- Name:** A text input field.
- Description:** A larger text area.
- Execution Schedule:** A section with a dropdown for frequency (0), a dropdown for unit (Days), a dropdown for timing (After), and a dropdown for trigger (Workflow is triggered).
- Script:** A section with a 'Method Name' dropdown and a plus sign icon to add a new function.

7. Provide a meaningful name for your Groovy Script action.
8. Specify an **Execution Schedule**.
9. Select a **Method Name** from the drop-down list. This list displays all Groovy object functions that are visible to the action. You can also click the plus sign next to the Method Name field to display the Create Object Function page and define a new function. See Defining an Object Function for Use in Object Workflows.

The script appears in read-only form in the **Script** field.

Note: If your Groovy Script action triggers a long-running process that's implemented via ESS/PLSQL, any Groovy triggers already created on the target object won't get called. This is because ESS/PLSQL operates directly at the database level, bypassing ADF and the model layer.

10. Click **Save**.

Defining an Object Function for Use in Object Workflows

If you haven't previously defined an object function to select from the **Method** list on the **Create Action - Groovy Script** page when creating your object workflow, you can define one directly from that page. To define an object function for use with the Groovy Script action:

1. On the **Create Action - Groovy Script** page, click the plus sign next to the **Method Name** field.
The Create Object Function page displays.
2. Enter a name for the function.

3. Select **void** in the **Returns** field.

The screenshot shows the 'Create Object Function' dialog box. The 'Definition' section includes a 'Function Name' field with the value 'Groovy_action_function', a 'Returns' dropdown menu set to 'void', and a 'Description' text area. The 'Visibility' dropdown menu is set to 'Callable by External Systems'. The 'Parameters' section is empty. The 'Edit Script' section is also empty. The dialog box has 'Validate', 'Save and Close', and 'Cancel' buttons at the top right.

4. Select **Callable by External Systems** in the **Visibility** field.
5. Do not specify any input parameters.
6. Enter the script in the **Edit Script** field.
7. Click the **Validate** button to validate the script, and fix any errors.
8. Click **Save and Close**.

Your object function is now ready to be called by a Groovy Script action in your object workflow. You can select it from the **Method** list when configuring an object workflow's Groovy Script action.

Related Topics

- [About Object Workflows](#)

Configure Object Workflows

This example demonstrates how to create an object workflow and set a trigger condition for invoking (or launching) the workflow. The trigger condition that will launch the workflow will be when the budget availability date for an opportunity is updated.

As part of the example, you configure these three event actions:

- **Field Updates:** Set the opportunity Close Date to 7 days from the new budget availability date. Additionally, lower the Level of Risk for the opportunity and set the Strategic Value to Medium.

- **Task Creation:** Create a task for the opportunity owner to follow up with the customer.
- **E-mail notification:** First, create an e-mail template for sending e-mail notifications. Then, notify the entire opportunity team about the change in the budget availability date using e-mail.

This example has been split into the following steps:

1. Creating a workflow and setting its trigger condition.
2. Configuring a Field Updates event action.
3. Configuring a Task Creation event action.
4. Creating an E-Mail Template and then creating an E-Mail Notification event action.
5. Creating an Opportunity record.
6. Editing the Opportunity Record to Trigger the Workflow and Verifying the Invoked Event Actions.

1. Creating an Object Workflow and Setting its Trigger Condition

In this example, you create an object workflow using an Opportunity object and create a trigger condition for the workflow using Groovy script. When the budget availability date is changed, the workflow will be triggered.

1. Click the **Navigator** menu.
2. Click the **Application Composer** link.

Note: You might need to click the more >> link if you don't see Application Composer.

The Overview page of Application Composer appears. This page shows the various tasks available to modify and extend your application. In this activity, you are configuring a workflow for the Opportunity object.

3. In the **Overview** region, click the **Object Workflows** link.

The Object Workflows page appears. You can use this page to search for an existing workflow or create a workflow. In this activity, you create a workflow.

4. Click the **Actions** menu below the Search region.
5. Click the **Create** menu item.

The Create Object Workflow page appears.

6. You must first select an object for which you are creating a workflow. Click the **Object** list.
7. Click the **Opportunity** list item.
8. Enter "Budget Date Revised" in the **Name** field.
9. In the Event Point and Condition region, click the **When a record is updated** option.
10. You set the trigger condition using Groovy script. Launch the expression builder to create the condition. In the Event Point and Condition region, click the expression builder button to open the expression builder.

The Expression Builder dialog appears.

11. Before you proceed, ensure that the **Fields** tab is selected.
12. Identify the correct application programming interface (API) name for the field you want to use for defining your trigger condition. Under the **Display Label** column in the Opportunity: Fields table, locate and click the **Date Budget Available** cell.
13. Insert Date Budget Available into the expression builder. Click the **Insert** button.
14. Enter your script in the Expression area using the BudgetDateAvailable field you just inserted. Write a script that meets all of these conditions:
 - a. Only the BudgetAvailableDate is updated.
 - b. The BudgetAvailableDate is not null.

- c. The opportunity record that you create is updated.

The following script has been written for you using the BudgetDateAvailable field:

```
if(isAttributeChanged('BudgetAvailableDate') && 'BudgetAvailableDate' != null && contains(Name, ' 50  
Solar Green Servers')) { return true; } else { return false; }
```

15. Validate your script. Click the **Validate** button.
16. Click the **OK** button.
17. You have set the event point and trigger condition for your object workflow. You now create a Field Updates event action. In this event action, you specify the values with which the Close Date, Level of Risk, and Strategic Value values will be replaced when the field updates action is triggered.

2. Configure a Field Update Event Action

Continuing from the previous step, you are on the Create Object Workflow page. In this step, you configure Field Updates event action for the object workflow and set new values for fields when the workflow is triggered.

- Set the Close Date to 7 days from the new budget availability date.
 - Lower the Level of Risk from High to Low.
 - Set the Strategic Value to Medium.
1. In the Actions region, click the **Create** button on the right of the Field Updates action.
The Create Action: Field Updates page appears.
 2. Enter "Update Close Date" in the **Name** field.
 3. In the Execution Schedule region, keep the default setting, which is to update fields right away when the workflow is triggered.
 4. In the Field Update Details region, click the **Field to Update** list.
 5. Click the **Close Date** list item.
 6. Click the **Value** list.
 7. Click the **Date Budget Available** list item.
 8. Make sure the '+' operator is selected, and enter "7" in the **Days** field.
 9. Click the **Update More Fields** link.
 10. Click the **Field to Update** list.
 11. Click the **Level of Risk** list item.

You will lower the Level of Risk for an opportunity when the workflow is triggered. The values in the Level of Risk field are in a descending order of High, Low, and None; therefore, when you select the **Populate with next value in list** option for Level of Risk, the risk level will be lowered by one step. For example, if the initial value is High, it will change to Low when the workflow is invoked, because Low is the next value in the list.

12. Click the **Populate with next value in list** option.
13. Click the **Update More Fields** link.
14. Click the **Field to Update** list.
15. Click the **Strategic Value** list item.
16. Click the list on the right of the **Value** group of options.
17. Click the **Medium** list item.
18. Save the Field Update event action. In the upper-right region of the page, click the **Save** button.

You're back to the Create Object Workflow page. Now you create a Task Creation event action for the sales team to follow up with the customer.

3. Configuring a Task Creation Event Action

Continuing from the previous step, you are on the Create Object Workflow page. In this example, you configure a Task Creation event action for an object workflow. A task will be created for the opportunity owner to follow up with the customer.

1. In the Actions region, click the **Create** button on the right of the Task Creation event action.
2. Enter "Follow Up Call" in the **Name** field.
3. In the Task Details region, enter "Follow up with customer on budget available date" in the **Subject** field.
4. You now insert a field token in the Subject field. Place your cursor where you want the token to appear. In the **Subject** field, click after "customer".
5. Click the field-token list on the right of the **Subject** field.
6. Click the **Customer** list item.
7. Click the **<< Insert** button.

Notice that the `[$TargetPartyName$]` token is inserted in the Subject where you placed your cursor. You need to adjust trailing or leading text-spaces as required.

8. Enter "**Opportunity Budget Available Date is changed. Follow up with within three days of the new available date. Primary Contact Email ID:** " in the **Description** field.
9. Use the field-token list on the right of the Description field to insert the following tokens one by one into the description text:
 - o **Opportunity Name** (`[$Name$]`)
 - o **Primary Contact** (`[$PrimaryContactPartyName$]`)
 - o **Primary Contact E-Mail** (`[$PrimaryContactEmailAddress$]`)
10. Click after "Opportunity" in the Description text.
11. Click the field-token list on the right of the **Description** field.
12. Click the **Opportunity Name** list item.
13. Click the **<<Insert** button.
14. Click after "Follow up with" in the Description text.
15. Click the field-token list on the right of the Description field.
16. Click the **Primary Contact** list item.
17. Click the **<<Insert** button.
18. Click after "Email ID:" in the Description text.
19. Click the **Primary Contact E-Mail** list item.
20. Click the **<<Insert** button.
21. You will now set the task's Due Date to be 3 days after the new Budget Available Date. Click the **Due Date** list.
22. Click the **Date Budget Available** list item.
23. Make sure the **plus sign +** operator is selected, and use the increment button (up arrow) to set the value in the **Days** field to **3**.
24. Click the **Start Date** list.
25. Click the **Date Budget Available** list item. Accept the default **0** (zero) in the **Days** field.
26. Click the **Owner** list.
27. Click the **OwnerResourcePartyid** list item.
28. Click the **Assignees** list.
29. Click the **OpportunityResource Team** list item.
30. Click the **Category** list, then click the **Call, outbound** list item..
31. Click the **Priority** list, then click the **1 - Very high** list item.

32. Save the **Task Creation** event action. In the upper-right region of the page, click the **Save** button.
33. You are back to the Create Object Workflow page. In the upper-right region of the page, click the **Save** button. This saves the object workflow.

You now create the E-Mail Notification event action. Before you create this event action, you must first create an E-Mail Template that you use for sending e-mail notifications.

4. Creating an E-Mail Template and Configuring an E-Mail Notification Event Action

Continuing from the previous step, you are now on the Object Workflows page. In this step, you create an e-mail template, which you use for sending e-mail notifications.

1. In the Common Setup pane on the left, click the **E-Mail Templates** link.
You are on the E-Mail Templates page. You can use this page either to search and edit an existing template or to create a fresh template. In this activity, you create a template.
2. In the Search Results region, click the **Actions** menu.
3. Click the **Create** menu item.
The Create E-Mail Template page appears.
4. Click the **Object** list.
5. Click the **Opportunity** list item.
6. Enter Budget Available Date Update" in the **Name** field.
7. Enter "**Opportunity customer budget available date has changed**" in the **E-Mail Subject** field.
8. You now insert a field token in the **E-Mail Subject** field. Place your cursor where you want the token to appear. Click after "Opportunity" in the **E-Mail Subject** field.
9. Click the field-token list on the right of the **E-Mail Subject** field.
10. Click the **Opportunity Name** list item.
11. Click the **<<Insert** button
12. In the E-Mail Body region, enter "**- budget available date regarding opportunity has moved to . The new opportunity close date is .**" in the **E-Mail Body** field.
13. In the E-Mail Body region, use the **Fields** list item of the **Select** list to insert the following field tokens:

Field	Token
Customer	[\$PartyUniqueName\$]
Opportunity Name	[\$Name\$]
Date Budget Available	[\$BudgetAvailableDate\$]
Close Date	[\$EffectiveDate\$]

Note: Rich text formatting is available only if your e-mail account supports HTML format e-mail.

14. Click after "- " in the e-mail body.

15. In the E-Mail Body region, click the field-token list on the left of the **Insert** button.
16. Click the **Customer** list item.
17. Click the **Insert** button.
18. Click after "regarding opportunity" in the e-mail body.
19. Click the field-token list.
20. Click the **Opportunity Name** list item.
21. Click the **Insert** button.
22. Click after "has moved to" in the e-mail body.
23. Click the field-token list.
24. Click the **Date Budget Available** list item.
25. Click the **Insert** button.
26. Click after "close date is" in the e-mail body.
27. Click the field-token list.
28. Click the **Close Date** list item.
29. Click the **Insert** button.
30. Save the E-Mail Template. In the upper-right region of the page, click the **Save** button.

You're returned to the E-Mail Templates page.

31. In the Common Setup pane on the left, click the **Object Workflows** link.

You now create an E-mail Notification event action using the e-mail template you just created.

32. In the Search region, click the **Object** list.
33. Click the **Opportunity** list item.
34. Click the **Search** button.

The **Name** column in the search results lists the object workflows created for the opportunity object. From this list, you identify and select the object workflow that you are creating.

35. Click the **Budget_Date_Revised** cell.
36. Click the **Edit** button.

The Edit Object Workflow page appears. You are editing an existing object workflow to configure and add an E-Mail Notification event action.

37. In the Actions region, click the **Create** button on the right of the E-Mail Notification event action.

The Create Action: E-Mail Notification page appears.

38. Enter "**Notify sales team about budget available date change**" in the **Name** field.
39. Select the e-mail template that you created. In the E-Mail Details region, click the **E-Mail Template** list.
40. From the list of e-mail templates, identify and select the template that you created. Click the **Budget Available Date Update** template.
41. Click the **Recipient Type** list.
42. Click the **Specific e-mail addresses** list item.
43. Enter the desired information into the **E-Mail Addresses** field.
44. Save the E-Mail Notification event action. In the upper-right region of the page, click the **Save** button.

You are returned to the Edit Object Workflow page.

45. Save the object workflow. In the upper-right region of the page, click the **Save** button.

You have now configured an object workflow with three event actions. This object workflow is invoked (launched) when you update the budget available date for the opportunity that you create.

5. Creating an Opportunity Record

In this section, you trigger the object workflow that you created in the previous steps by entering the budget availability date for an opportunity. You note down the details where the changes will occur after the workflow is triggered.

1. Click the **Navigator** menu.
2. Click the **Opportunities** link under **Sales**.

The Overview page of Opportunities appears. You use this page to create an opportunity record.

3. In the Opportunities region, click the **Create** button.

The Create Opportunity page appears.

4. Verify that **Close Date** is the current date, **Sales Stage** is 01-Qualification, and **Win Probability (%)** is 0 (zero).
5. In the **Name** field, enter "**50 Solar Green Servers**".
6. In the **Sales Account** field, enter "**Solar Inc (KIRKLAND, US)**".
7. Click the **Sales Account** search button adjacent to the **Sales Account** field.

The Search and Select: Sales Account dialog appears.

8. In the dialog, the Solar Inc (Kirkland, US) search parameter appears in the **Name** field. Click the **Search** button.
9. In the Search Results region, click the **Solar Inc (Kirkland, US)** cell.

Note: Ensure that the All Sales Accounts tab is selected.

10. Click the **OK** button.
11. In the Revenue Items region, click the **Add Row** button.
12. Enter "**DG 150 Green Servers**" in the **Name** field.
13. Enter "**50**" in the **Quantity** field.
14. Enter "2000" in the **Estimated Price** field.
15. In the upper-right region of the page, click the **Save and Edit** button.

The Edit Opportunity page appears.

16. Expand the following if not already expanded:
 - o Show More area in the Summary region.
 - o Revenue Items region.
 - o Activity Center region.
17. Locate the following fields and note their current values:
 - o **Close Date** (in the Summary region).
 - o **Strategic Value** (under Show Less area in the Summary region).
 - o **Level of Risk** (under Show Less area in the Summary region).
 - o **Date Budget Available** (under Show Less area in the Summary region).

When you update the **Date Budget Available** field to trigger the workflow, the values in the **Close Date**, **Strategic Value**, and **Level of Risk** fields should change.

18. Under the Show Less area in the Summary region, click the **Level of Risk** list.
19. Click the **High** list item.
20. In the upper-right region of the page, click the **Save** button.
21. You are still on the Edit Opportunity page. In the Activity Center region, click the **Tasks** tab.

Currently there are no Tasks with your sign-in initials.

22. In the Additional Details region, click the **Opportunity Team** tab.
23. In the Team Members table, locate your sign-in username. You will now add another team member: **Mateo Lopez**.
24. In the Opportunity Team tab, click the **Add Team Members** button.
The Select and Add: Team Members dialog appears.
25. Find a team member and click the **Done** button.
26. Search and add a contact to the opportunity. This will be the primary contact for the opportunity.
27. In the Additional Details region, click the **Contacts** tab.
28. In the Contacts tab, click the **Add Row** button.
29. Enter the desired information into the **Name** field. As you type in the contact name, the contact name and e-mail ID appear.
30. Click the desired contact.
31. In the upper-right region of the page, click the **Save and Close** button.
32. You're back to the Overview page of Opportunities. You will now edit the opportunity record you just created and update the budget availability date to trigger the event actions.

6. Editing the Opportunity Record to Trigger the Workflow and Verifying the Invoked Event Actions

Continuing from the previous step, you are on the Overview page of Opportunities. You will trigger (launch) the object workflow that you created in the previous steps by entering the budget availability date for an opportunity.

You will also verify the following after the workflow is invoked:

- Field Updates: Verify that the opportunity Close Date is set to 7 days from the new Budget Available Date. Verify that the Level of Risk is lowered for the opportunity and the Strategic Value is set to Medium
 - Task Creation: Verify that a task is created for the opportunity owner to follow up with the customer.
 - E-mail notification: Verify that an e-mail notification is sent to the entire opportunity team about the change in the opportunity's Budget Available Date.
1. From the table in the Opportunities region, identify and select the opportunity record that you created. Click the **50 Solar Green Servers** link.
 2. You are on the Edit Opportunity page. Before you proceed, expand the Show More area in the Summary region if not already expanded.
 3. Under the Show Less area in the Summary region, click the **Date Budget Available** (calendar) button.
 4. Click **9** as the date, or click any future date.
 5. In the upper-right region of the page, click the **Save and Close** button.

Note: You have just changed the budget availability date and saved the record, saving the changes to the database. The object workflow will be evaluated and triggered at this time, and all the event actions will be executed. The Field Updates event action always happens first followed by other event actions in no particular order. It might take a few seconds for the updates to occur.

6. In the table under the Opportunities region, identify and select the opportunity record that you created. Click the **50 Solar Green Servers** link.

You are on the Edit Opportunity page. The workflow is now triggered and the configured event actions have been invoked. You will now verify the invocation of Field Updates event action.

7. In the Summary region, **Close Date** is now set to **7** days after the **Date Budget Available** value.
8. Under the Show Less area in the Summary region, **Strategic Value** is now set to **Medium**.
9. Under the Show Less area in the Summary region, **Level of Risk** has now been populated with the next value in the list, from **High** to **Low**.

You have verified the invocation of the Field Updates event action. Now, you verify the invocation of the Task Creation event action

In the Activity Center region, the **Due Date** filter under the Tasks tab may prevent your task from appearing in the invoked tasks list even after it is invoked. For example, a task having a due date after 10 days will not appear if the Due Date filter is set to Next 7 Days. In this activity, you use the **All Tasks** filter option to search for your invoked task.

10. In the Activity Center region, click the **Due Date** list under the Tasks tab.
11. Click the **All Tasks** list item.
12. In the Tasks tab, click the arrow button adjacent to the **Category** list.
13. In the Search results, locate the **Subject** that has your sign-in initials. Click the **Follow up with customer Solar Inc on budget available date** link.

You are on the Edit Task page. Note these details:

- o The **Owner** of the opportunity is the owner of this task.
- o The task **Assignees** are persons in the opportunity sales team.
- o In the **Description** field, the field tokens that you had inserted when creating the Task Creation event action have been replaced with values at runtime.

14. Click the **Done** button.
15. In the upper-right region of the page, click the **Save and Close** button.

You have verified the invocation of the Task Creation event action. Next, verify the invocation of the E-Mail Notification event action.

16. In your e-mail client, locate the **Subject** of your e-mail. Click the **Subject** link.
17. Notice that the field tokens that you inserted while configuring the e-mail notification event action have been replaced with values at runtime. You have now verified the invocation of the E-Mail Notification event action.

You have now verified the invocation of all three event actions.

Related Topics

Configure Notification Delivery Methods from Object Workflow

You can send notifications from an object workflow using the notification dispatcher. Create a Groovy function in the context of an object workflow, where you can define the notification recipients and delivery channels.

You can configure these notifications for multiple business objects, all delivery channels, and multiple recipients.

- When you configure an object workflow within Application Composer, navigate to **Object Functions** and create a function.
- Set the **Visibility** as **Callable by External Systems**.

- Within the body of the function, call a new global method, `sendNotificationSimple()`.

This Groovy function calls in to the notification framework, and eventually its dispatcher sends the notification.

Here's a sample Groovy script:

```
try {
  def recipientPartyId = AssigneeResourceId
  def messageText = 'An SR notification for status change.'

  if (recipientPartyId) {
    // Call to send notification
    adf.util.sendNotificationSimple(adf, messageText, recipientPartyId)
  }
} catch (e) {
  // Log the failure in groovy logging. Logs can be viewed in 'Runtime Messages'.
  println("Failure to trigger notification from Groovy Script " + e.getMessage());
}
```

Delete Unpublished Object Workflows

You can use Application Composer to delete any unpublished object workflows and their associated actions from the current sandbox. Upon deletion, the corresponding create or update events won't be raised and none of the actions will be triggered.

However, deleting an unpublished object workflow won't delete the email template or object function referenced in the object workflow actions. Also, if an action has been scheduled before the object workflow is deleted, then the action will continue to be executed at the scheduled time. If your sandbox was already published, then you can't delete the object workflow but you can inactivate it.

To delete an unpublished object workflow:

1. In Application Composer, click **Object Workflows** under Common Setup.

The Object Workflows page appears.

2. Click the **Object** list and select the object whose workflow you want to delete.
3. Optionally, if you know the name of the workflow, enter it in the **Name** field and specify if it's active or not using the **Active** list.
4. Click **Search**.

The page displays the object workflow based on the specified search criteria.

5. Select the object workflow and click the delete icon.
6. Click **OK** on the confirmation dialog.

Inactivate an Object Workflow

Instead of deleting an object workflow, you can inactivate it. Do this if you think you might need the object workflow later. Also, if your sandbox was already published, then your only choice is to inactivate it. If an action has been scheduled before the object workflow is inactivated, then the action will continue to be executed at the scheduled time.

An object workflow is automatically activated once you save it.

To inactivate an object workflow:

1. In Application Composer, open the existing object workflow that you want to inactivate.

2. Deselect the Active check box.
3. Click **Save**.

FAQs for Creating Object Workflows

Why are some email templates invalid?

Email templates can become invalid if the corresponding template cannot be found in the file repository. This can happen if the template file was migrated elsewhere or was accidentally deleted. Contact your application administrator to attempt recovering the template (file) from the file repository.

Can I specify multiple approvers when configuring a business process flow event action?

Yes. You can modify the default template to design either a parallel or serial business process flow and save it as a new project. A parallel approval is where any one approval is sufficient from the approvers specified in the process flow.

A serial approval is where all the approvals are required sequentially from the specified approvers.

Related Topics

- [Set Up Serial Group Approval](#)
- [Set Up Parallel Approval](#)

7 Custom Subject Areas

About Custom Subject Areas

With your applications, you get prebuilt analytics that answer typical business questions you might have. But in case your questions aren't answered, you can create your own analytics using prebuilt subject areas.

If the prebuilt subject areas don't cover what you need, then you can create your own custom subject areas to use to create analytics that do answer your questions. This feature is especially useful as you work through the process of configuring your applications. For example, if you create custom objects and want to report on them, then you will need to create custom subject areas.

Subject Areas and Analytics

What's the relationship between subject areas and analytics?

A subject area is the starting point for building analytics. It's basically a collection of related information from your database. Analytics, meanwhile, are like snapshots of a subject area. They give you a focused view of that collection of information, so you can get more meaningful insights from your data.

You can look at a subject area like a painter's palette and the resulting analytics are the completed paintings that you create. The subject area contains all the colors you need to paint one or more pieces of artwork. And your finished artwork gives you the intelligence you need to better understand and operate your business.

When to Create Custom Subject Areas

Creating custom subject areas isn't required, and you do get prebuilt subject areas that contain a lot of information from which you can build your own custom analytics. You will need to create custom subject areas, however, for two reasons:

- If you create custom objects and want to report on them
- If the existing prebuilt subject areas don't contain the measures (facts) you need for reporting

How to Create and Use Custom Subject Areas

To create a custom subject area, you're guided through the process step by step, but here's a quick overview:

1. Navigate to Application Composer.
2. Click **Custom Subject Areas** on the Overview page of Application Composer.
3. Select **Create** from the Actions menu.
4. Complete the guided process.

Next, to actually build analytics from that subject area, use BI Composer. Using a custom subject area in BI Composer is the same as using a prebuilt subject area.

1. From the Navigator menu, select Tools > Reports and Analytics.
2. In the Contents pane, click **Create**.
3. Select the published custom subject area and start creating your analytic, such as a report or an online analysis.

Things to Remember

Before you start creating custom subject areas, you should know the following:

- **Tools:**
 - Use Application Composer to create custom subject areas. Then use BI Composer to create analytics based on those custom subject areas.
 - If you're using unified migration to migrate your configurations, then always create custom subject areas in the source environment, not the target environment. Otherwise your custom subject areas will be overwritten when you do the migration. And, when creating the migration set on the Migration page, remember to include Business Intelligence content so that your custom subject areas will be visible in both Analytics Answers and BI Composer.
- **Objects:** A subject area is always built around a business object. That object is the focus of any reporting created using that subject area. You can add related objects to a subject area, but there is always a primary object and you can't change it.
- **Dates and numbers:**
 - The real meaningful substance of any reporting is typically "how much" or "how many." You're measuring things to get insights into how well your business is doing. When creating a custom subject area, you can tag attributes of the custom subject area object as measures, which means you specify which date and number fields are measurable. You will use these measures when you later use this subject area to create analytics.
 - You can also enable date leveling, which means that you can group the data in your analytics by different calendar attributes. You do this by mapping a date attribute to a calendar object, like Week Of the Year, Month of the Year, or Enterprise Quarter. This mapping lets you view data in your analytics by a specific quarter or year, for example.
- **Publishing:** Once you create a custom subject area, you can't actually use it in BI Composer until you publish it.

In the next few topics, we will review some of these key concepts. We will then talk in more detail about how to create custom subject areas.

Related Topics

- [Custom Subject Area Objects](#)
- [Measures](#)
- [Custom Subject Area Fields](#)
- [Custom Subject Areas Dates](#)

Custom Subject Area Objects

When you create a custom subject area, one of the first things you do is pick the primary object. The primary object is the focus of any reporting created using that subject area. Does your subject area need more data to be helpful?

If so, then just add child objects to the subject area. A subject area always has one primary object and, depending on your reporting needs, could have one or more child objects. Still need more data? You can further expand the scope of information by adding related objects as well. Let's look at these objects in more detail.

Primary Object

You pick a primary object during the first train stop when creating a custom subject area. The list of available objects includes all standard objects as well as top-level custom objects. A top-level object is simply an object without a parent.

After you save your custom subject area, the primary object is frozen and can't be changed.

Note: You can't include Notes and Tasks in a custom subject area.

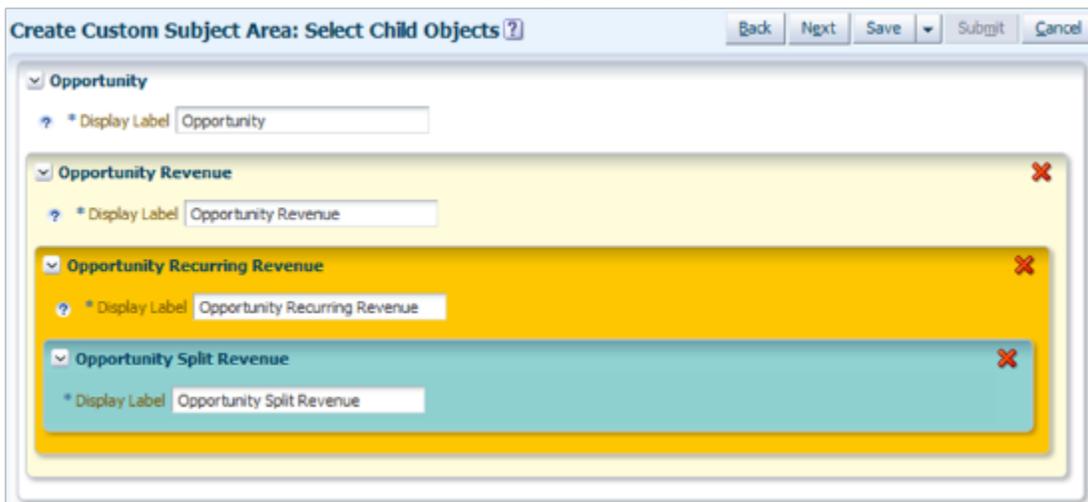
Child Objects

To expand the scope of information that you can report on, you can add one or more child objects during the second train stop when creating a custom subject area. After adding child objects, you can include data from both the primary object and its children in the analytics you create from this subject area.

In a family, one parent can have many children; we call this a one-to-many relationship. In Application Composer, you create this type of relationship by creating a parent object with one or more children. So, if a parent object is the primary object of a custom subject area, then you can add its children to the subject area, too. Note that in Application Composer, you can't create a child object of a child object (grandchildren).

When you create custom subject areas, however, it's a little different. Yes, you can add child objects to a subject area. But, you add child objects in sequential order. It's a single hierarchy of objects: parent-child-grandchild-great-grandchild.

Here's a screenshot of what the parent-child-grandchild-great-grandchild hierarchy looks like.



The parent-child-grandchild-great-grandchild hierarchy supports adding up to three levels of child objects with one child object at each level; for example, parent-child1-child1.1-child1.1.1. If no child objects exist for the selected primary object, then you can't add child objects to a custom subject area.

After you publish a custom subject area, you can't add or remove child objects.

Related Objects

If you need still more data in your custom subject area, above and beyond what you can get from the primary object and its child objects, then you can also include attributes (fields) from related objects. You do this during the third train stop when creating a custom subject area.

Think of related objects like friends of the primary object. They aren't quite children, but they still have a relationship with the primary object. A related object doesn't have a parent and can exist on its own. In contrast, child objects can't exist without a parent. So, instead of the parent-child one-to-many relationship, related objects have a many-to-one relationship with the primary object. You can think of that like many people can be friends with the same parent. In Application Composer, you create this type of relationship by creating a many-to one relationship on the Relationships page.

After you publish a custom subject area, you can't remove related objects, but you can always add new related objects and republish your subject area.

How Custom Subject Areas Support Bidirectional Reporting

Custom subject areas can support reporting in both directions for any custom one-to-many reference relationships and dynamic choice list relationships that were created in Application Composer. You can see this support when selecting the primary and child objects for your custom subject area.

1. When selecting the primary and child objects, you can select either the source or target object in the object hierarchy.
2. Depending on the type of relationship, the other side of the relationship is available for selection as either a child or related object.

This support for bidirectional reporting directly in your custom subject area reduces the need for multiple custom subject areas and joins.

Related Topics

- [About Custom Subject Areas](#)
- [Create Custom Subject Areas](#)

Custom Subject Area Fields

After you pick the primary object and add child objects to a custom subject area, the next train stop in the process is to pick the fields (also called object attributes or just attributes).

If you recall, a subject area is a collection of information from your database, and analytics are snapshots and measurements of that information. Picking the fields is the part of the process where you build that collection of information in your subject area. When you're later using this subject area to design an analytic, you will be able to choose from the set of fields that you included in this step.

Where Fields Come From

The fields that you can pick belong to the primary object and any child objects that you added in the previous train stop. You can also add fields from objects that are related to the primary object.

Tip: Fields represent a column of information contained in an object. When you build your analytics in BI Composer, you build them by adding columns. Columns and fields in relational database terms are the same thing.

Types of Fields You Can Add

There are different types of fields in your applications. You can add most of these types to custom subject areas. Here's what you can add:

- Text
- Number
- Date

Keep an eye out for date fields that are canonical date candidates, which are predefined for some standard objects and subject areas, and display with a read-only check mark. Optionally add canonical date fields from the lowest standard object in your custom subject area. This means that, if you later join your custom subject area to a standard subject area in BI Composer, you can create analytics that can drill up and down the date hierarchy.

For example, maybe users want to see sales commission amounts on their closed opportunities for the quarter, and then drill down to view amounts by week. And maybe they also want to drill up to view amounts by year. This drill down is enabled because the canonical date field in your custom subject area is automatically joined to the standard Time dimension in the standard subject area.

- Percentage
- Date time
- Currency
- Check box
- Fixed choice list

In your analytics, only the actual user selections are displayed, as text strings.

It's a good idea to avoid adding multi-select fixed choice list fields to your custom subject areas, since only the first selected values are displayed in analytics, not all selected values.

- Dynamic choice list

In your analytics, only the actual user selections are displayed, as text strings.

You can add custom dynamic choice list fields only to custom subject areas, not prebuilt subject areas.

- Long text

Types of Fields That Analytics Answers and BI Composer Supports

Meanwhile, Analytics Answers and BI Composer supports a corresponding set of data types when creating analytics. It's basically the same as the types of fields you can add to a custom subject area in Application Composer. But, there are slight differences in terms of how check box fields and choice list fields in a custom subject area are interpreted.

- Boolean

Note: Check box fields from your objects are interpreted as Boolean fields on the BI side. If you're using the Boolean data type for fields other than check boxes, then those fields are displayed as either 0 or 1 in your analytics.

- Number
- Currency
- Date
- String

As mentioned above, fixed choice list and dynamic choice list fields are interpreted as string fields on the BI side.

- Percentage
- Date time
- Long text

Related Topics

- [About Custom Subject Areas](#)
- [Create Custom Subject Areas](#)

Measures

When you want to see how your business is doing, you typically review analytics to find out "how much" or "how many." Keeping on top of this kind of summary data helps you to evaluate your company's performance.

To get the exact summaries you need, you specify the date and number fields in your custom subject areas that you want to summarize, and you also specify how you want to summarize them. These fields are called measures.

What's a Measure?

Measures are a set of functions applied to a date or number field so that your users can see summaries in their analytics. Examples of typical measures include a SUM of the revenue in Euros, or a COUNT of the number of opportunities worth over \$500,000.

When you create a custom subject area, you pick the aggregation function (SUM, COUNT, and so on) for measures. You can apply these functions on fields of type Date, Numeric, or Currency. After you define the measures for the required fields and publish the custom subject area, you can select these fields and the applied measures when creating analytics in BI Composer.

Tip: The measures you define in Application Composer display in the Facts folder in BI Composer.

Things to Remember

You can specify measures only when creating a custom subject area. You can't delete measures in an already published custom subject area, but you can add new measures. You can't edit or add measures to a standard, prebuilt subject area.

If you don't define a measure, then one will be automatically selected from the lowest object in the custom subject area when you submit the subject area for publication.

Measures and Field Types

Select measures based on your reporting needs. For example, you can use measures to view product sales per store, state, or country. Or, use measures to view the number of support tickets opened or closed per day, week, or month, and so on. But keep in mind, the measures available for selection might differ depending on the field type.

Here are some measures you can apply to fields of type Numeric, Currency, or Date.

- For Numeric and Currency fields, a measure can be:

- All

Note: All isn't a measure, but an option in the UI that selects all of the measures.

- Sum: Calculates the sum of the values.
- Average: Calculates the mean value.
- Count: Calculates the number of rows that aren't null.
- Count Distinct: Calculates the number of rows that aren't null. Each distinct occurrence of a row is counted only once.

Note: Although Count Distinct is usually used in cases requiring a count on a foreign key (because a count of distinct rows is what's wanted), it's not required. If your requirements allow multiple instances of the same foreign key value to be counted multiple times, you can use Count rather than Count Distinct.

- Maximum: Calculates the highest numeric value.
- Minimum: Calculates the lowest numeric value.
- First: Selects the first occurrence of the item.
- Last: Selects the last occurrence of the item.
- Median: Calculates the middle value.
- Standard Deviation: Calculates the standard deviation to show the level of variation from the average.
- Standard Deviation Population: Calculates the standard deviation using the formula for population variance and standard deviation.

- For Date fields, a measure can be:
 - All
 - Maximum
 - Minimum

Related Topics

- [About Custom Subject Areas](#)
- [Custom Subject Areas Dates](#)
- [Custom Subject Area Fields](#)
- [Create Custom Subject Areas](#)

Custom Subject Areas Dates

Once you pick the fields for your custom subject area, you can then review just the date fields for two additional configuration steps: allowing date leveling and picking one canonical date field.

First, you can enable one or more fields for date leveling which enables the aggregation of report data over a period of time. You can also enable one field to be the canonical date which lets you join this custom subject area to the standard Time dimension. Using this common date, your users can filter data across multiple subject areas in a single report, and drill up and down the date hierarchy. Let's look at date leveling and the concept of canonical date in more detail.

Why Date Leveling is Useful

Date leveling gives you additional date context about your transaction data, and enables the filtering of data by calendar attributes. Date leveling means that you're connecting (mapping) a date field in your custom subject area to a very large calendar object in BI Composer. Later, in BI Composer, that date field will display with a series of calendar attributes. (The calendar object supports both the enterprise and fiscal calendar.) When you design your analytics, you pick that date field plus the calendar attributes you're interested in, such as Day Name, Week of the Year, Month of the Year, or Enterprise Quarter. In your resulting analytic, transaction data displays along with the helpful date context that you have added.

For example, a report can display a set of opportunity records along with the enterprise quarter they were created in: Q1, Q2, and so on. That's date leveling. And, maybe you need to know which day the opportunity was created on, such as on a Monday or Wednesday. That's also date leveling.

Maybe you want to see a report of all orders entered, but you would like to know what enterprise quarters they were entered in. If you enable date leveling on the creation date for your sales order object, then you can create analytics that show orders per enterprise quarter.

Let's refine that report a little more. Maybe you want to see only orders created in 2019 Q1. When you design that report, add a filter on the enterprise quarter to display only records where the quarter is equal to 2019 Q1.

If you don't enable date leveling for a date field, then later in BI Composer, that date field will still display for inclusion in an analytic, but without the added calendar attributes that are so helpful.

Configure Date Leveling

To configure date leveling, use the **Configure Dates** step of the guided train process to either allow or disallow leveling for a date field. You might need to expand the field list in the **Date Field** column to view the Allow Leveling check box.

Enable date leveling only for those dates that you want to report on. And, even after you publish the custom subject area, you can always come back and select additional dates for date leveling, and then republish.

Why the Canonical Date is Useful

A canonical date in a custom subject area is useful because, when you join your custom subject area to a standard subject area in BI Composer, the canonical date automatically joins to the standard Time dimension. The Time dimension includes the date hierarchy, so this means that the analytics you create can drill up and down the date hierarchy.

But first, let's understand where the canonical date comes from. Canonical dates are already defined for some standard objects. You can see these fields in the Fields step when adding fields from standard objects to your custom subject area. (A date field that's a canonical date will have a check mark in the Canonical Date Candidate column.) If you plan to design a report that lets users drill up and down the date hierarchy, then include a canonical date field in your custom subject area. But that's only the first step.

In the next Configure Dates step, you must select the radio button for that field in the Canonical Date column. Do this for one field only, and only for the lowest standard object in your custom subject area. Canonical dates are available only for standard objects because custom objects don't have corresponding subject areas in BI Composer with the standard Time dimension to join to.

Let's look at an example of drilling up and down the date hierarchy. Maybe users want to see sales commission amounts on their closed opportunities for the quarter, and then drill down to view amounts by week. Maybe they want to drill up to view amounts by year. This is drilling up and down. To achieve this, you must do the following:

1. In Application Composer, create a custom subject area for opportunity and opportunity revenue, with the Line Close Date as the canonical date from the opportunity revenue object.
2. In BI Composer:
 - a. Create an analysis that includes both the custom subject area and a standard subject area, such as Sales - CRM Pipeline.
 - b. In addition to including at least the Sales Commission field, include the date hierarchy in your analysis, such as Enterprise Time, from the Sales - CRM Pipeline's Time dimension folder.

Configure the Canonical Date

To indicate which date field should be the canonical date, use the **Configure Dates** step of the guided train process. To enable a date field as the one canonical date for a custom subject area, view the date fields for the lowest standard object in the custom subject area, and select the radio button for the desired date field in the Canonical Date column. You might need to expand the field list in the **Date Field** column to view the Canonical Date radio button.

You can add a canonical date to both new and existing custom subject areas.

If you publish a custom subject area without a canonical date, then you can always come back later and select a canonical date, and then republish.

Related Topics

- [About Custom Subject Areas](#)
- [Create Custom Subject Areas](#)

Secure Custom Subject Areas

You can secure a custom subject area by granting or revoking access rights from the role names that access the custom subject area. You can add or delete role names, or grant or revoke access rights from those role names.

You can also add role names from a predefined list and assign or revoke permissions.

Manage Role Names and Access Rights

While defining a custom subject area using the train stops, you can use the Actions list in the Configure Security step to manage role names and access rights as follows:

- Select and add role names for a custom subject area from a predefined list of role names. This predefined list also provides the description for each role name. You can also select and add multiple role names from this predefined list using either the `shift` or `ctrl` keys. After you add a new role name, you can select appropriate access for that role name.
- Select and delete role names listed for a custom subject area. You can also select and delete multiple role names using either the `shift` or `ctrl` keys.

Note: You can't delete the role name listed as **Everyone**.

- Read access is granted by default to each role name you add. To revoke Read access from a listed role name, select **No access** for that role name.

Note: You can create custom subject areas even for the objects whose data you don't have access to. This way you build custom subject areas without compromising data security.

Related Topics

- [About Custom Subject Areas](#)
- [Create Custom Subject Areas](#)
- [Publication Statuses of Custom Subject Areas](#)

Create Custom Subject Areas

Creating a subject area is simple. In Application Composer, just follow the train steps.

Tip: Before you create a custom subject area, review all the prebuilt subject areas to see if the one you want is already available.

Creating a Custom Subject Area

To create a custom subject area, you can't be in a sandbox.

1. Navigate to Application Composer.
2. Click **Custom Subject Areas** on the Overview page of Application Composer.
3. Select **Create** from the **Actions** menu.

Here are the steps in the train stops that you can use for configuring your custom subject area:

1. Define Custom Subject Area

In this step, you provide the name for your subject area and select the primary object that is the basis for the reports you create later using the custom subject area. Subject areas usually have names or labels that correspond to the type of information they contain, such as service requests and orders. Display labels have the Custom: prefix added automatically.

2. Select Child Objects

In this step, you select the child objects whose data you want to use in your reports. You can add child objects only if the primary object has child objects. Otherwise, the add icon is disabled. You can only add one child object per level. The parent-child-grandchild-grand grandchild hierarchy supports adding up to three levels of child objects with one child object at each level, for example, parent-child1-child1.1-child1.1.1.

3. Configure Fields

In this step, you select the fields that you want to display on your reports. You typically add at least one field from each of the objects that you have selected for your custom subject area.

Select the desired measures to generate for number, date, or currency fields from all the available objects so that the subject area includes only those measures that you want to analyze. Also, define at least one measure.

In the **Measure Aggregations** column, select an option from the list of predefined formulas that you can apply to the Measure field. When you select the formula, the application applies the selected formulas to the selected field as measures.

Note: You can choose measures only for the lowest child. For example, if only a primary object exists with no children, you can select measures for the primary object. Otherwise, if any child objects exist, you can select measures only at the lowest child object level, not for the parent object.

In the **Canonical Date Candidate** column, notice which standard object date fields are already defined as canonical dates. Canonical dates are already defined for some standard objects that support the Time dimension in standard subject areas, and display with a read-only check mark. Optionally add this field from

the lowest object in your custom subject area. This means that you can later join your custom subject area to a standard subject area in BI Composer, and create analytics that can drill up and down the date hierarchy.

You can change the display labels of the fields that you select in this step. Additionally, you can use the Select Fields dialog to remove fields that belong to the primary object, or add fields from the related objects. The Select Fields dialog appears when you click Select Fields when configuring fields for your custom subject area.

After you publish your custom subject area, the fields you have selected for your subject area are automatically added to their owning object's folder. If you have also defined measures, those fields are automatically added to the Facts folder. If you didn't define a measure, then one is automatically created for the custom subject area.

4. Configure Dates

If required, select one or more Date columns for date leveling. Date leveling lets you add calendar attributes to your analytics, which provide better context to your transaction data.

You can also optionally select one date field to be the canonical date for the custom subject area. Do this if you want to create analytics that can drill up and down the date hierarchy.

For more information on configuring date fields in a custom subject area, see [Custom Subject Area Dates](#).

5. Configure Security

Select the required security level for the **Everyone** Role Name, which is added by default, or add additional Role Names by clicking in the + icon and define the security level for each one of them.

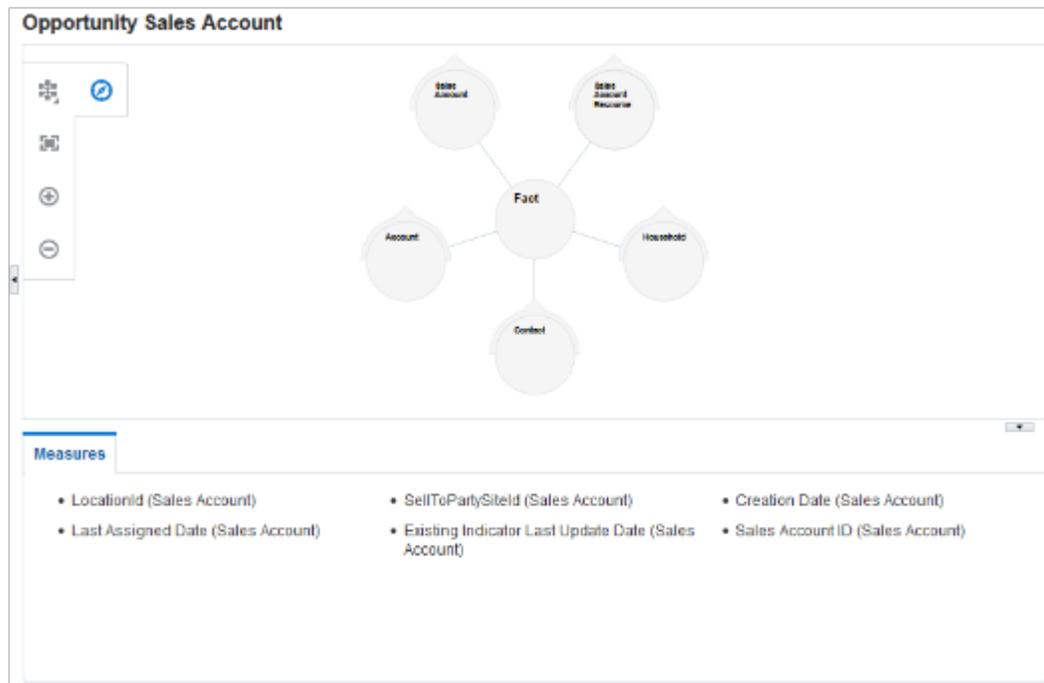
The security definition here only controls who can access the custom subject area definition to create reports. It doesn't control data visibility which is automatically controlled based on the user running the reports.

For more information on securing custom subject areas, see [Secure Custom Subject Areas](#).

6. Review and Submit

Review the custom subject area configuration for all added objects, attributes, and measures, and if satisfied, click **Submit**. If changes are required click **Back** to navigate back to the required screen.

This figure shows the custom subject area configuration.



After you submit, the custom subject area configuration is prepared for publishing. You can create and submit a custom subject area either immediately or save and close the custom subject area at any point and submit it later. You must first submit a custom subject area for publishing before you can select it from within Oracle BI Composer. After you save or submit a custom subject area, you can't modify its primary object.

To access the published custom subject area in BI:

- From the Navigator menu, select **Tools > Reports and Analytics**.
- In the Contents pane, click **Create**.
- Select the published custom subject area and start creating your report.

Editing Custom Subject Areas

You can edit a published or saved custom subject area and then republish it when your changes are done. Modifying a custom subject area doesn't affect the reports that you had created using that custom subject area before making the changes. You can use the modified custom subject area if you need to enhance existing reports.

To edit a custom subject area:

1. On the Overview page of the Application Composer, click **Custom Subject Areas**.

2. Locate the custom subject area that you want to edit, and click the Edit icon.

You can filter out inactive custom subject areas in Application Composer by viewing custom subject areas in Active status. This is safer than deleting them, because the inactive subject areas are still available and can be found by searching.

3. Make the desired changes and then click **Submit** to republish the custom subject area.

While you can edit a custom subject area in any status, there are considerations on what you can or can't do when editing. When editing a published custom subject area, it's not possible to:

- Change the primary object.
- Add or remove child objects.
- Remove related objects.
- Remove fields.
- Remove previously added measures, date levels, and canonical dates.
- Add more aggregation types for measures that are already published.

Note: You can't modify a predefined, standard subject area. Instead, you must create separate custom subject areas to meet your reporting needs. Before you create a custom subject area, be sure to review all the included subject areas to see if the one you want is already available.

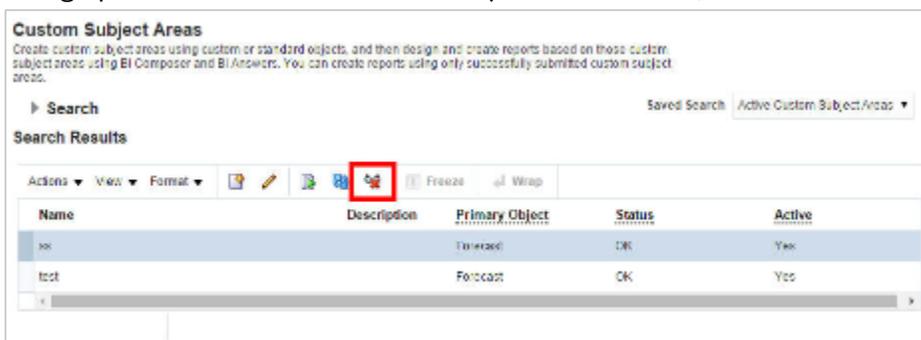
Activating or Inactivating Custom Subject Areas

When editing custom subject areas, you can activate or inactivate custom subject areas when your reporting or business requirements change. This step enables you to control what information is displayed on the reports that use the information from custom subject areas.

You can inactivate only those custom subject areas that are published and have a status of *OK*, and can activate only previously inactivated custom subject areas.

To inactivate a custom subject area, select it in the list and then click the **Inactivate** button. To activate an inactive custom subject area, select it and click **Activate**. Note that if no custom subject area is selected in the list, the button doesn't appear.

This graphic shows an active custom subject area selected, and the Inactivate button.



When searching for custom subject areas, you can filter out inactive custom subject areas in Application Composer by viewing only those in Active status. Inactivating a custom subject area is safer than deleting it, because the inactive subject areas are still available and can be found by searching.

Related Topics

- [About Custom Subject Areas](#)
- [Custom Subject Areas Dates](#)
- [Measures](#)
- [Custom Subject Area Objects](#)

Publication Statuses of Custom Subject Areas

After you create a custom subject area, you must publish it so that it's available for use in BI Composer. A custom subject area can have a few different publication statuses. Let's look at each one.

What Happens When You Submit for Publishing

But first, let's understand the publication process. When you submit a custom subject area for publishing, two processes occur in the background. The first process is synchronous and creates Oracle Applications Development Framework (Oracle ADF) artifacts. You must wait until this first process is over. The second process is asynchronous and creates centralized metadata repository (RPD) fragments and submits them to the Oracle BI server.

If you're working on multiple custom subject areas at once, then be sure to publish them one at a time. The first custom subject area might publish successfully, but subsequent custom subject areas could result in publish failures.

Publication Statuses

A custom subject area can have one of the following statuses:

- Pending: This status indicates either of the following:
 - You saved and closed the configuration process for a custom subject area before submitting it for publishing.
 - A failure occurred in the background processes when creating Oracle ADF and RPD artifacts.
- In Process: This status indicates that the data is in the process of being published to Oracle BI.

Note: If the in-process status doesn't change to **OK**, even after multiple refresh attempts, then there could be an error in publishing. If an error occurs, then the details are displayed as well as information about how to fix problems, where applicable. Use these error status details to pinpoint and fix problems quickly.

- OK: This status indicates that the custom subject area has been published successfully. You can use BI Composer to create reports using the objects, attributes, and measures that you have configured in the subject area.

Note: You must refresh the status to know whether the custom subject area was submitted successfully. You might have to refresh the status multiple times because it could take a while to create the Oracle ADF and RPD artifacts.

Related Topics

- [About Custom Subject Areas](#)
- [Create Custom Subject Areas](#)
- [Secure Custom Subject Areas](#)

How to Report on Custom Fields

If you add custom fields to certain key standard objects, then you don't have to create a custom subject area just to report on those new fields. Actually, key objects come with predefined extension dimensions.

Notes:

- To report on custom dynamic choice list fields added to standard objects, you must create a custom subject area because you can't add custom dynamic choice list fields to prebuilt subject areas. You can then create a union report in Analytics Answers that joins the desired prebuilt subject area with your new custom subject area.
- The formula field type isn't supported by subject areas. You can't add formula fields to a custom report.

Extension dimensions exist in certain standard subject areas in BI Composer and conveniently hold all your custom fields. This means that if you create a custom field for Opportunity, for example, then you don't have to do anything to report on it. Simply go to your favorite standard subject area for Opportunity; your custom field automatically appears in the extension dimension, ready for inclusion in analytics.

Create Analytics Using Extension Fields

Here's how to create analytics using extension dimension fields:

1. In Application Composer, create custom fields for standard objects, and add those custom fields to the user interface.
2. Publish the sandbox.
3. From the navigator, select **Reports and Analytics** under Tools to navigate to BI Composer.
4. Select a standard subject area whose primary object is the object that you created custom fields for.
5. Create your analytic.

When you specify the columns to include, you can select the extension fields from the extension dimension folder, which appears as the <Object Name> Extension folder. For example, Opportunity Extension.

The extension fields available for reporting vary by object type.

Hierarchies in Analytics

Some prebuilt, standard subject areas include hierarchies, such as a customer or territory hierarchy. When you design analytics with a hierarchy, users can drill up and down that hierarchy to view data grouped at different levels.

For example, with a territory hierarchy, users can drill up and down their data to look at information grouped by region, state, and city. Hierarchies are delivered with standard subject areas only; you can't add them to custom subject areas.

What you can do, however, is join your custom subject area with a standard subject area that has the hierarchy you need. This lets you create analytics with a hierarchy using data pulled from your custom subject area.

Supported Hierarchies

The following hierarchies are supported in some standard subject areas, such as Sales - CRM Pipeline:

- Resource
- Territory
- Customer
- Partner

Example of Adding a Hierarchy to Analytics

If you want to create analytics that include a hierarchy, and also pull data from a custom subject area, then here's how you do it.

Let's say you have a custom object named Ticket that you want to report on. The Ticket object includes the Account field as a dynamic choice list. This means at runtime, your users can create tickets and assign accounts (customers) to them.

Now you have 1,000 tickets in your database and it's time to report on them. Your users would like to view total number of tickets up and down their customer hierarchy.

Here's how you can create a report to meet this need:

1. Create a custom subject area for Ticket and publish it.
2. Create a report using a standard subject area, such as Sales - CRM Customer or Contacts Real Time. These subject areas include the customer hierarchy.
3. Add the account hierarchy from the subject area to your report.
4. Next, add your Ticket custom subject area to the report. This "joins" the two subject areas.
5. Finally, add the metric, such as Number of Tickets, from the custom subject area to your report.

Related Topics

- [Reporting Using Data from More than One Subject Area](#)

Custom Subject Areas: Frequently Asked Questions

Frequently Asked Questions (FAQs) for Custom Subject Areas

This section contains some frequently asked questions (FAQs) for custom subject areas. Scroll or search to find answers to common questions.

How can I report on custom objects?

To report on custom objects, you must create and publish a custom subject area in Application Composer. Once you have done that, then creating analytics in BI using a custom subject area is the same as using a standard subject area.

Can I change a custom subject area's primary object?

No. Once you save a custom subject area, you can't change its primary object. You can, however, create a new custom subject area with a different primary object.

What happens if I change a custom subject area after it's published?

You can edit a published custom subject area and then republish it after your changes are done. Modifying a custom subject area doesn't affect the reports that you created using that custom subject area before making the changes.

You can use the modified custom subject area should you need to enhance existing reports.

Note: You can't edit a primary object when you modify a custom subject area. Should you need to do so, create a new custom subject area using a different (new) primary object.

Related Topics

- [Publication Statuses of Custom Subject Areas](#)
- [About Custom Subject Areas](#)

How do I migrate my custom subject areas to a production environment?

To migrate your custom subject areas to a production environment, use unified migration. And, when creating your migration set on the Migration page, check the Business Intelligence content check box.

This ensures that your custom subject areas will be accessible from either Analytics Answers or BI Composer.

Note that the custom subject area's automatically generated sequence, available from the session log, doesn't change during migration.

Related Topics

- [Overview of Migration](#)

Why can't I see my custom subject areas in Oracle Analytics Answers or BI Composer after migrating them?

If you didn't include Business Intelligence content in your migration set during migration, then you won't see your custom subject areas in either Analytics Answers or BI Composer in the target environment after migration.

To see your custom subject areas, you must resubmit them for publication in Application Composer.

Note: To preserve the integrity between your source and target environments, resubmit only those custom subject areas in your target environment that were already published in the source environment.

Related Topics

- [Overview of Migration](#)