Oracle Fusion Cloud Sales Automation

How do I configure notifications for Sales?

Oracle Fusion Cloud Sales Automation
How do I configure notifications for Sales?

G30289-03

Author: Carmen Myrick

Contents

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 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.

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 How do I configure notifications for Sales?

Overview of Configuring Notifications for Sales

A notification is an alert for users, such as salespeople and managers, to notify them about an event on a business object, such as an account, and let them take necessary actions. Enabling notifications is a global setup.

You can generate, configure, and enable notifications for Sales objects such as accounts, contacts, leads, and opportunities.

Depending on your configuration, the application sends notifications to users when there are updates to records. When configuring notifications, you determine who receives the notifications and when they receive them.

> **Note:** All of these types of notifications appear at the top of the page under Notifications (the bell icon). When users click the notification icon, they're directed to the appropriate page for your enabled UI.

Use this playbook to learn to do these tasks:

- Enable notifications for Sales
- Define notification scripts for Sales
- Set notification preferences for Sales
- Debug Groovy prompts
- Configure push notifications

This table lists the setup steps and where you can get more information.

Setup Steps for Notifications

Step	Where to Get More Details
Step 1 Enable notifications for sales objects.	*Enable Notifications for Sales*
Step 2 Get an overview of the tasks to configure notifications.	*Overview of Setting Up Notification Scripts and Preferences*
Step 3 Define Groovy scripts using Application Composer.	*Define Notification Scripts for Sales*
Step 4 Configure notification rules.	*Configure Delivery Preferences and Text for a Notification*

Step	Where to Get More Details
Step 5 View the runtime log and debug your Groovy script.	*Debug Groovy Prompts*
Step 6 Push notifications to mobile devices.	*Configure Push Notifications*

Related Topics

- View and Act On Notifications

Enable Notifications for Sales

You define and generate notifications with a Groovy script in Application Composer, using conditions that must be met for each notification. When a notification is created on an account that salespeople own, you can specify the notification recipients.

For example, you can reference sample Groovy scripts to define and activate notifications for salespeople when:

- They're added to the account team.
- They're designated as owner of an account.
- An account they own has been deleted.
- A Sales object is assigned to an account.

> **Note:** Before you enable notifications for Sales, ensure that you've enabled notifications generally. For steps, see the topic about enabling and disabling notifications, linked in the Related Topics section. If you update the applications, be sure to go back and check these settings again.

Notifications are lined up in a queue, and you can manually add a record to a queue by changing the owner. Or, you can add a notification to a queue based on specific criteria. You can use a rule or condition to add leads to a queue based on specific record criteria, for example. Records remain in a queue until they're assigned an owner.

Here's how you enable notifications:

1. Sign in as a setup user.
2. In the Setup and Maintenance go to the Sales offering and select the **Sales Foundation functional area**.
3. Click the drop-down button in the Sales Foundation row, then click **Change Feature Selection** in the menu. The Edit Features: Sales Foundation page appears.
4. Select the **Enable** checkbox for the notification-related features you want to set up. This enables notifications for the Sales objects.

   > **Note:** This opt-in is for your chosen channel or communication path, such as bell notifications, and not for specific objects.

5. Click **Done**.

Related Topics

- Disable or Enable Workflow Notifications
- About Object Workflows
- Best Practices for Content and Layout in Workflow Notifications

Overview of Setting Up Notification Scripts and Preferences

After you enable the notifications feature for sales, you then configure the notifications.

Here are the high-level steps:

1. Define notification prompts as Groovy scripts in Application Composer. The scripts contain the conditions that must be met for each notification. Notifications are sent when the defined conditions are met. For example, you can send a notification when a sales object is assigned to an account.

   For more information about defining notification prompts, see *Define Notification Scripts for Sales*.
2. Configure notification preferences using the Notification Preferences page. For example, configure the notification message using SmartText and specify the notification recipient.

   For more information about notification preferences, see *Configure Delivery Preferences and Text for a Notification*.

Define Notification Scripts for Sales

Here's how you define a Groovy script for a notification prompt:

1. Ensure you're working in an active sandbox.
2. Click **Navigator** > **Configuration** > **Application Composer**.
3. In the navigation tree, expand Standard Objects, expand the object you want, and then click **Server Scripts**. For example, **Standard Objects** > **Account** > **Server Scripts**.
4. Click the **Triggers** tab on the Server Scripts page.
5. Click the add icon.
6. In the Create Trigger Object page, create the Groovy trigger:
   a. In the **Trigger** field, select the trigger type.

      Oracle recommends that you use the trigger type **After Changes Posted to Database**. This trigger type lets you to stop potential issues if the Groovy script is accidentally written to run indefinitely. If the trigger type is set to Before Update to Database, with a bad script, you may receive errors.

      If you're creating a new object, and you want to trigger a notification when the object is created, Oracle recommends that you use the trigger type **Before Insert to Database**. However, some Before trigger types don't have all attributes exposed yet, resulting in some fields being blank. To debug your triggers if you're not getting the expected results, follow the steps in *Debug Your Groovy Scripts*.
   b. Enter a name for the trigger.

c. Enter the trigger Definition details. When you create a Groovy script, you need the API names of the fields you're trying to access. Create your triggers based on parent/child fields, with these steps:
   i. In the Trigger Definition section, click **Show/Hide Expression Palette**.
   ii. Click the **Fields** tab.
   iii. Select an **Object**.
   iv. Click the **Maximize Edit Script** arrow. The fields for the selected object are displayed.
   v. Select the API you want, and click Insert.
   vi. To close the expression palette, click the **Restore Edit Script** arrow, and then click **Show/Hide Expression Palette**.

   The `isAttributeChanged()` function works only for the Before trigger types. The workaround for the After trigger types involves retrieving the old value before the database is updated, then retrieving the new value after the update, and then comparing the two values to see whether the attribute is changed. However, this workaround works only for high-level attributes, such as CriticalFlag and Status, and not for the ViewRow attribute.

7. Add a Groovy script. For example, if you're defining the notification for accounts, add the `Account Push notification received` Groovy script.
8. Click **Save and Close**.
9. Navigate to the Notification Preferences page to configure your preferences for the notifications. See *Configure Delivery Preferences and Text for a Notification* for more information.

Note: You can reuse a Groovy notification prompt if no other notification uses it.

Related Topics

- Overview of Sandboxes
- Create and Activate Sandboxes
- Publish Sandboxes

Configure Delivery Preferences and Text for a Notification

Here's how you specify the recipient types, delivery methods, and notification text for a specific trigger event:

1. Sign in as a user with the Sales Administrator job role.
2. In the Navigator, click **Tools** > **Notification Preferences**.
3. In the Notification Preferences page, select an object from the **Object** list.
4. Click **Add** to add a blank row to the table.
5. Select **Yes** from the Enabled drop-down list
6. In the **Override** field, select Yes or No:

   Override lets you specify whether a user has the ability to opt-out of the notification. If you set it to no, users can't opt out of the notification.
7. Select a Groovy notification prompt from the **Triggering Event** list.
8. Enter the notification name and description. The text you enter into the Notification Name field becomes the first part of the notification subject.

9. In the Recipients column, click the edit icon.
10. In the Configuration for Trigger Name window, select the notification delivery options for recipients and then click **Done**.
    The application sends notifications to the specified recipients only if you select at least one delivery method, so be sure to select at least one method. In addition to getting communication through the selected delivery method, the recipient also receives a bell notification. Select the **Bell Notifications** option to enable the notifications that appear in the global header.
    See the tables later in this topic for more information about the supplied recipients and delivery methods.
11. Click the **New SmartText** link, enter the notification text for the selected object, and then click **Publish**.
    SmartText also lets you add variables to the notification message.
12. To delete a notification preference, select the row and click **Delete**. This action deletes any associated notification text, too.

    > **Note:** If you delete a notification that's generated using a Groovy script, you can create a new notification using the same Groovy script, if it isn't used by another notification.

13. To modify an existing notification text, click the **Update SmartText** icon for the selected row.
14. Click **Save**.

Here are the predefined notification recipients:

Recipient	Description
Manager of Owner ID	Manager of the notification owner
Owner ID	The one who sends or assigns the notification
Resource team	Additional resources added to an object in the Team subtab (Fusion Service only) **Note:** Resource teams aren't applicable for Digital Sales.

Here are the predefined delivery methods:

Delivery Method	Description
Bell Notifications	Enables the bell notification for the web application
Mobile Notifications	Enables notifications on the mobile application
Browser Notifications	Enables notifications to users in the Omnichannel toolbar (Fusion Service only)
Email Notifications	Enables email notifications

For more information about using SmartText, see topics on using smart texts in the Using Fusion Service guide.

Note: In the sales applications, activities (tasks and appointments) use a different model to set up notifications. For more information on how to set up notifications for activities, see the "Set Up Activity Notifications" topic in the Activities chapter in the Implementing Sales guide.

Related Topics

- Set Up Activity Notifications
- How do I use SmartText in service requests?
- What are the SmartText naming conventions?

Debug Your Groovy Scripts

Use the Groovy script 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 Groovy Script Debugger

Access the Groovy script debugger from either the Custom Objects or Standard Objects page in Application Composer.

To access the debugger:

1. In Application Composer, in 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.

Use the Groovy Script Debugger

The Groovy script debugger contains multiple regions, described in the following table, which you can use to debug your scripts for an object:

Groovy Script Debugger Regions

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.

Debugger Region	Description
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 `println()` statements, then those values are captured on this tab.

To use the debugger:

1. In Application Composer, in 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:
   - Step Over

     Review one statement in the selected script at a time.
   - 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**.
   - Step Out

     If you're debugging inside a child function and you want to move the control back to the parent function, then click **Step Out**.
   - 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.

Enable or Disable the Groovy Script Debugger

The Groovy script debugger is enabled by default. However, 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, go to: **Sales offering** > **SalesFoundation functional area** > **Manage Administer Profile Values task**.
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`.
   - TRUE displays the debugger.
   - FALSE hides the debugger.

Sample Groovy Scripts for Notifications

Use Groovy scripts to define the start of notifications. Here's sample code to generate notifications to the recipients specified in the Notification Preferences page for sales objects. You can change the scripts to fit your requirements.

Scripts for Standard Objects

Here are sample scripts for standard sales objects:

Sample Groovy Scripts for Notifications

Object	Scenario	Create and Assign Scripts
Accounts	Push notification received	Account Create `// Send notification to Owner when Account is created // Use trigger type BEFORE INSERT TO DATABASE def map = new HashMap(); // Specify one or more channels map.put("Channels", ["ORA_SVC_BELL"]); // Specify default MessageText def messageText = "Account created with name: " + OrganizationName; map.put("MessageText", messageText); // The following can be used to pass a Long PartyId map.put("RecipientPartyId", OwnerPartyId); adf.util.sendNotification(adf, map)`
Accounts	Drill down to details page	Account Assign `// Send notification to Owner when Account is assigned to them // Use trigger type BEFORE UPDATE TO DATABASE if (isAttributeChanged('OwnerPartyId')) { try { def map = new HashMap(); def messageText = "This Account has been assigned to you" def recipientPartyId = OwnerPartyId // Specify one or more channels map.put("Channels",["ORA_SVC_ BELL"]); // Specify default MessageText map.put("MessageText", messageText); // The following can be used to pass a Long PartyId map.put("RecipientPartyId", recipientPartyId); if (recipientPartyId) { adf.util.sendNotification(adf, map) } else { println("No Owner associated with this Account") } } catch (e) { throw new oracle.jbo.ValidationException('Failure: ' + e.getMessage()) } }`

Object	Scenario	Create and Assign Scripts
Contacts	Push notification received	Contact Create `// Send notification to Owner when Contact is created // Use trigger type BEFORE INSERT TO DATABASE def map = new HashMap(); // Specify one or more channels map.put("Channels", ["ORA_SVC_BELL"]); // Specify default MessageText def messageText = "Contact created with name: " + PersonName; map.put("MessageText", messageText); // The following can be used to pass a Long PartyId map.put("RecipientPartyId", OwnerPartyId); adf.util.sendNotification(adf, map)`
Contacts	Drill down to details page	Contact Assign `// Send notification to Owner when Contact is assigned to them // Use trigger type BEFORE UPDATE TO DATABASE if (isAttributeChanged('OwnerPartyId')) { try { def map = new HashMap(); def messageText = "This Contact has been assigned to you" def recipientPartyId = OwnerPartyId // Specify one or more channels map.put("Channels",["ORA_SVC_BELL"]); // Specify default MessageText map.put("MessageText", messageText); // The following can be used to pass a Long PartyId map.put("RecipientPartyId", recipientPartyId); if (recipientPartyId) { adf.util.sendNotification(adf, map) } else { println("No Owner associated with this Contact") } } catch (e) { throw new oracle.jbo.ValidationException('Failure: ' + e.getMessage()) } }`
Leads	Push notification received	Lead Create `// Send notification to Owner when Lead is created // Use trigger type BEFORE INSERT TO DATABASE def map = new HashMap(); //give lead object API name def objectCode = 'MklLeadVO' map.put("ObjectCode", objectCode) // Specify one or more channels map.put("Channels", ["ORA_SVC_BELL"]); // Specify default MessageText def messageText = "Lead created with name: " + Name; map.put("MessageText", messageText); // The following can be used to pass a Long PartyId map.put("RecipientPartyId",`

Object	Scenario	Create and Assign Scripts
		OwnerId); adf.util.sendNotification(adf, map);
Leads	Drill down to details page	Lead Assign Lead Assign // Send notification to Owner when Lead is assigned to them // Use trigger type BEFORE UPDATE TO DATABASE if (isAttributeChanged('OwnerId')) { try { def map = new HashMap(); def messageText = "This Lead has been assigned to you" def recipientPartyId = OwnerId // Specify one or more channels map.put("Channels",["ORA_SVC_BELL"]); // Specify default MessageText map.put("MessageText", messageText); // The following can be used to pass a Long PartyId map.put("RecipientPartyId", recipientPartyId); if (recipientPartyId) { adf.util.sendNotification(adf, map) } else { println("No Owner associated with this Lead") } } catch (e) { throw new oracle.jbo.ValidationException('Failure: ' + e.getMessage()) } }
Opportunities	Push notification received	Opportunity Create // Send notification to Owner when Opportunity is created // Use trigger type BEFORE INSERT TO DATABASE def map = new HashMap(); // Specify one or more channels map.put("Channels",["ORA_SVC_BELL"]); // Specify default MessageText def messageText = "Opportunity created with name: " + Name; map.put("MessageText", messageText); // The following can be used to pass a Long PartyId map.put("RecipientPartyId", OwnerResourcePartyId); adf.util.sendNotification(adf, map)
Opportunities	Drill down to details page	Opportunity Assign // Send notification to Owner when Opportunity is assigned to them // Use trigger type BEFORE UPDATE TO DATABASE if (isAttributeChanged('OwnerResourcePartyId')) { try { def map = new HashMap(); def messageText = "This Opportunity has been assigned to you" def recipientPartyId = OwnerResourcePartyId // Specify one or more channels map.put("Channels",["ORA_SVC_BELL"]); // Specify default

Object	Scenario	Create and Assign Scripts
		`MessageText map.put("MessageText", messageText); // The following can be used to pass a Long PartyId map.put("RecipientPartyId", recipientPartyId); if (recipientPartyId) { adf.util.sendNotification(adf, map) } else { println("No Owner associated with this Opportunity") } } catch (e) { throw new oracle.jbo.ValidationException('Failure: ' + e.getMessage()) } }`

Tip: In these scripts, you're only enabling the bell icon in the global header. To enable notifications for other channels (for example, browser and mobile), use this syntax: `// Specify one or more channels map.put("Channels", ["ORA_SVC_BELL", "ORA_SVC_OMNI","ORA_SVC_MOBILE"]););`

- ORA_SVC_BELL: Notifications icon in the global header
- ORA_SVC_OMNI: Notifications icon in the browser
- ORA_SVC_MOBILE: Mobile notifications

Groovy Script for Custom Objects

You can use a Groovy script to send notifications for custom objects.

Prerequisites:

- Use the global function, `adf.util.sendNotification` (adf, map).
- The `objectCode` needs to be explicitly passed as part of the request parameter hash map in the Groovy trigger.
- Your custom object needs to be published.

Sample Groovy Script for Custom Object Notifications

Object	Scenario	Create and Assign Scripts
Custom object	Send bell notification to the user based on the user's party ID	`try { Long recipientPartyId = 100010037456865 def messageText = 'A custom object notification (default message).' def objectCode='CustomObject_SR_c' def map = new HashMap(); map.put("Channels", ["ORA_SVC_BELL"]); map.put("MessageText", messageText); map.put("RecipientPartyId", recipientPartyId); map.put("ObjectCode", objectCode); adf.util.sendNotification(adf, map) println("Triggered a notification"); } catch (e) { // Log the failure in groovy logging. Logs can be viewed in 'Runtime Messages'. println("Failure to`

Object	Scenario	Create and Assign Scripts
		`trigger notification from Groovy Script " + e.getMessage()); }`

To find the Object Code for your custom object, navigate to the overview page for the object. The API name is listed in the Object Information region of that page.

Configure Push Notifications

With push notifications, users get notifications on their mobile devices. You define notifications in Application Composer with your unique, conditional logic. Devices such as iOS and Android support push notifications.

For example, you can define alerts for upcoming milestones, reassignments, or escalations. Define the starting point to send a notification when you assign a high-severity request to a user.

After they sign in to the mobile application, users can view a list of their notifications by clicking the Notifications icon on the global header. They can dismiss notifications after viewing or dismiss them in bulk.

As you implement push notifications, you decide when to raise each notification. You specify whether to enable mobile push notification alerts, or to deliver the alerts only to the bell notifications on the desktop application.

> **Note:** If the Notifications feature is set up, it automatically sets up bell notifications. Delivery of bell notifications happens even if mobile notifications are disabled. Clicking a notification and navigating to the record details marks the notification as read, and it's cleared from the notification list.

Here's how you enable push notifications:

1. Ensure you're working in an active sandbox.
2. In the Setup and Maintenance go to the Sales offering and select the Sales Foundation functional area.
3. Click the drop-down button in the Sales Foundation row.
4. Click **View Feature Selection** in the menu. The Edit Features: Sales Foundation page appears.
5. Select the **Enable** checkbox for the notification-related features you want to set up. This enables notifications for the Sales objects.

   > **Note:** This opt-in is for your chosen channel or communication path, such as bell notifications, and not for specific objects.

6. Select the **Enable** checkbox for Mobile Notifications.
7. Publish the sandbox.