Analytics SDK Developer Guide

Salesforce, Spring '24

@salesforcedocs

Last updated: January 12, 2024

CONTENTS

ANALYTICS SDK DEVELOPER GUIDE OVERVIEW

Use the Analytics SDK features to embed CRM Analytics functionality directly where your users work everyday, without having to transition between Lightning Experience and CRM Analytics Studio. The Analytics SDK lets you communicate and interact with CRM Analytics assets from Lightning Apps, Lightning Web Components, Apex, Visualforce, and more. You can create one cohesive experience powered by CRM Analytics features directly in Salesforce pages and apps.

Important: The CRM Analytics SDK and all its components require the purchase of CRM Analytics Platform licenses.

CRM Analytics Lightning Web Components

Are you ready to use the power of CRM Analytics directly inside Lightning Experience and create your own custom CRM Analytics functionality? By using theCRM Analytics Lightning Web Components, you can access the power of REST APIs to retrieve collections of CRM Analytics data assets, execute queries and schedule data syncs for recipes and dataflows.

CRM Analytics Assets SDK Aura Component

Use the CRM Analytics `<wave:sdk>` Aura Component to retrieve collections of CRM Analytics assets, such as dashboards, lenses, and datasets and describe the details of individual assets. Then, customize the display of the results via a Lightning Component controller. You can also create dynamic SAQL queries against your CRM Analytics data to display runtime results.

CRM Analytics Templates SDK Aura Component

You have great apps, and you're creating app templates to copy or distribute those apps. And you might even be using the REST APIs to work with templates and folders.

By using the CRM Analytics `<wave:sdk>` Aura Component template methods, your application can do many of the same things from a Lightning Component controller.

CRM Analytics Aura Events

Would you like your application to communicate with your CRM Analytics dashboards, whether your application is built with the Lightning SDK, Visualforce, or mobile? How about from an application outside of Salesforce? Wouldn't it be great if your application could apply filters or know about dashboard selections and filters made by a user?

Your application could take actions specific for your business when values fall outside of defined ranges. Or you could have an application that is a viewpoint for dashboards made available by different parts of your business application ecosystem. Imagine that: a single information source to present to your executive staff!

The CRM Analytics Aura events are the foundation for a new way of thinking about CRM Analytics applications. Coupled with the Lightning Locker, you can even code your application outside of Salesforce—you can interact with CRM Analytics from any JavaScript application.

CRM Analytics in Apex

Is your company one of the gazillion using custom code in Apex, the server-side programming language for Lightning Platform? Would you like it to be easier to query data in CRM Analytics directly from your Apex code? Say hello to the CRM Analytics in Apex features.

CRM Analytics in Apex lets developers build well-formed SAQL queries and execute them in the security context of the logged-in user, ensuring that security settings are honored. API versioning is supported to avoid breaking applications as the feature set evolves. Use `Wave.Lenses` to list and describe CRM Analytics lens assets. CRM Analytics also offers `Wave.InvalidParameterException` to help catch bad values supplied to the class methods.

CRM Analytics Lightning Web Components

Use CRM Analytics Lightning Web Components to retrieve data and metadata for CRM Analytics assets, execute queries, and schedule data syncs for recipes, dataflows, and data connectors.

CRM Analytics Assets SDK Aura Component

Discover CRM Analytics dashboards, lenses and datasets, get their details, and dig into dataset fields. Discover dashboard saved views and explore dashboard state. Create and execute queries directly on datasets

CRM Analytics Template SDK Aura Component

Discover CRM Analytics templates and apps created from them. Create, update, and delete CRM Analytics apps created from templates.

CRM Analytics Aura Events

Easily interact with the embedded CRM Analytics Dashboard component in custom applications. Use Aura Events for Lightning Apps, Visualforce, or your preferred development environment.

CRM Analytics in Apex

Build and execute queries and retrieve lens and template data using CRM Analytics Apex classes and methods.

CRM Analytics Lightning Web Components

Use CRM Analytics Lightning Web Components to retrieve data and metadata for CRM Analytics assets, execute queries, and schedule data syncs for recipes, dataflows, and data connectors.

The `lightning/analyticsWaveApi` module provides wire adapters and JavaScript functions built on top of the CRM Analytics REST API. Use these wire adapters and functions to work with CRM Analytics data and metadata.

Wire Adapters

`getActions`

Retrieves a collection of Salesforce actions available for the specified Analytics user. For more information on syntax and usage, see the getActions reference in the Lightning Web Components Developer Guide.

`getAnalyticsLimits`

Retrieves the Analytics limits for CRM Analytics. For more information on syntax and usage, see the getAnalyticsLimits reference in the Lightning Web Components Developer Guide.

`getDataConnector`

Retrieves a specific CRM Analytics data connector by ID or developer name. For more information on syntax and usage, see the getDataConnector reference in the Lightning Web Components Developer Guide.

`getDataConnectors`

Retrieves a collection of CRM Analytics data connectors. For more information on syntax and usage, see the getDataConnectors reference in the Lightning Web Components Developer Guide.

`getDataConnectorSourceFields`

Retrieves a collection of source fields for a source object used by a CRM Analytics data connector. For more information on syntax and usage, see the getDataConnectorSourceFields reference in the Lightning Web Components Developer Guide.

`getDataConnectorSourceObject`

Retrieves a source object used by a CRM Analytics data connector. For more information on syntax and usage, see the getDataConnectorSourceObject reference in the Lightning Web Components Developer Guide.

`getDataConnectorSourceObjectDataPreviewWithFields`

Retrieves the fields for a data preview for a source object used by a CRM Analytics data connector. For more information on syntax and usage, see the getDataConnectorSourceObjectDataPreviewWithFields reference in the Lightning Web Components Developer Guide.

`getDataConnectorStatus`

Retrieves the status for a specific CRM Analytics data connector by ID or developer name. For more information on syntax and usage, see the getDataConnectorStatus reference in the Lightning Web Components Developer Guide.

`getDataConnectorTypes`

Retrieves the collection of CRM Analytics data connector types. For more information on syntax and usage, see the getDataConnectorTypes reference in the Lightning Web Components Developer Guide.

`getDataflowJob`

Retrieves a specific CRM Analytics dataflow job. For more information on syntax and usage, see the getDataflowJob reference in the Lightning Web Components Developer Guide.

`getDataflowJobs`

Retrieves a collection of CRM Analytics dataflow jobs. For more information on syntax and usage, see the getDataflowJobs reference in the Lightning Web Components Developer Guide.

`getDataflowJobNode`

Retrieves a specific CRM Analytics dataflow job node for a recipe or dataflow. For more information on syntax and usage, see the getDataflowJobNode reference in the Lightning Web Components Developer Guide.

`getDataflowJobNodes`

Retrieves a collection of CRM Analytics dataflow job nodes for a recipe or dataflow. For more information on syntax and usage, see the getDataflowJobNodes reference in the Lightning Web Components Developer Guide.

`getDataflows`

Retrieves a collection of CRM Analytics dataflows. For more information on syntax and usage, see the getDataflows reference in the Lightning Web Components Developer Guide.

`getDataset`

Retrieves a specific CRM Analytics dataset by ID or developer name. For more information on syntax and usage, see the getDataset reference in the Lightning Web Components Developer Guide.

`getDatasets`

Retrieves a collection of CRM Analytics datasets. For more information on syntax and usage, see the getDatasets reference in the Lightning Web Components Developer Guide.

`getDatasetVersion`

Retrieves a specific CRM Analytics dataset version by ID or developer name. For more information on syntax and usage, see the getDatasetVersion reference in the Lightning Web Components Developer Guide.

`getDatasetVersions`

Retrieves a collection of versions for a CRM Analytics dataset. For more information on syntax and usage, see the getDatasetVersions reference in the Lightning Web Components Developer Guide.

`getDependencies`

Retrieves a collection of dependencies for a CRM Analytics asset. For more information on syntax and usage, see the getDependencies reference in the Lightning Web Components Developer Guide.

`getRecipe`

Retrieves a specific CRM Analytics recipe by ID. For more information on syntax and usage, see the getRecipe reference in the Lightning Web Components Developer Guide.

`getRecipes`

Retrieves a collection of CRM Analytics recipes. For more information on syntax and usage, see the getRecipes reference in the Lightning Web Components Developer Guide.

`getReplicatedDataset`

Retrieves a specific CRM Analytics replicated dataset by ID, also known as a connected object. For more information on syntax and usage, see the getReplicatedDataset reference in the Lightning Web Components Developer Guide.

`getReplicatedDatasets`

Retrieves a collection of CRM Analytics replicated datasets, also known as connected objects. For more information on syntax and usage, see the getReplicatedDatasets reference in the Lightning Web Components Developer Guide.

`getReplicatedFields`

Retrieves a collection of fields belonging to a CRM Analytics replicated dataset, also known as connected object. For more information on syntax and usage, see the getReplicatedFields reference in the Lightning Web Components Developer Guide.

`getSchedule`

Retrieves a schedule for a CRM Analytics recipe, dataflow, or data sync. For more information on syntax and usage, see the getSchedule reference in the Lightning Web Components Developer Guide.

`getSecurityCoverageDatasetVersion`

Retrieves the security coverage for a specific CRM Analytics dataset version by ID or developer name. For more information on syntax and usage, see the getSecurityCoverageDatasetVersion reference in the Lightning Web Components Developer Guide.

`getStories`

Retrieves a collection of Einstein Discovery stories. For more information on syntax and usage, see the getStories reference in the Lightning Web Components Developer Guide.

`getWaveFolders`

Retrieves a collection of CRM Analytics apps or folders. For more information on syntax and usage, see the getWaveFolders reference in the Lightning Web Components Developer Guide.

`getWaveTemplate`

Retrieves a CRM Analytics template by ID or API name. For more information on syntax and usage, see the getWaveTemplate reference in the Lightning Web Components Developer Guide.

`getWaveTemplateConfig`

Retrieves the configuration for a CRM Analytics template by ID or API name. For more information on syntax and usage, see the getWaveTemplateConfig reference in the Lightning Web Components Developer Guide.

`getWaveTemplateReleaseNotes`

Retrieves the release notes for a CRM Analytics template by ID or API name. For more information on syntax and usage, see the getWaveTemplateReleaseNotes reference in the Lightning Web Components Developer Guide.

`getWaveTemplates`

Retrieves a collection of CRM Analytics templates. For more information on syntax and usage, see the getWaveTemplates reference in the Lightning Web Components Developer Guide.

`getXmd`

Retrieves a specific CRM Analytics extended metadata type (Xmd) for a version of a dataset. For more information on syntax and usage, see the getXmd reference in the Lightning Web Components Developer Guide.

Functions

`createDataConnector`

Creates an instance of a CRM Analytics connector to connect to data in your Salesforce orgs, apps, data warehouses, and database services. For more information on syntax and usage, see the createDataConnector reference in the Lightning Web Components Developer Guide.

`createDataflowJob`

Creates a CRM Analytics dataflow job, which is the equivalent of clicking **Run Now** for a data prep recipe, a data sync, or a dataflow in the CRM Analytics Data Manager UI. For more information on syntax and usage, see the createDataflowJob reference in the Lightning Web Components Developer Guide.

`createDataset`

Creates a dataset. For more information on syntax and usage, see the createDataset reference in the Lightning Web Components Developer Guide.

`createDatasetVersion`

Creates a version for a specific CRM Analytics dataset by ID or developer name. For more information on syntax and usage, see the createDatasetVersion reference in the Lightning Web Components Developer Guide.

`createReplicatedDataset`

Creates a CRM Analytics replicated dataset, also known as a connected object. For more information on syntax and usage, see the createReplicatedDataset reference in the Lightning Web Components Developer Guide.

`deleteDataConnector`

Deletes a specific CRM Analytics connector by ID or developer name. For more information on syntax and usage, see the deleteDataConnector reference in the Lightning Web Components Developer Guide.

`deleteDataset`

Deletes a specific CRM Analytics dataset by ID or developer name. For more information on syntax and usage, see the deleteDataset reference in the Lightning Web Components Developer Guide.

`deleteRecipe`

Deletes a specific CRM Analytics recipe by ID. For more information on syntax and usage, see the deleteRecipe reference in the Lightning Web Components Developer Guide.

`deleteReplicatedDataset`

Deletes a specific CRM Analytics replicated dataset by ID. For more information on syntax and usage, see the deleteReplicatedDataset reference in the Lightning Web Components Developer Guide.

`executeQuery`

Executes a CRM Analytics query written in Salesforce Analytics Query Language (SAQL) or standard SQL. For more information on syntax and usage, see the executeQuery reference in the Lightning Web Components Developer Guide.

`ingestDataConnector`

Triggers the CRM Analytics to run a data sync. For more information on syntax and usage, see the ingestDataConnector reference in the Lightning Web Components Developer Guide.

`updateDataConnector`

Updates a CRM Analytics data connector. For more information on syntax and usage, see the updateDataConnector reference in the Lightning Web Components Developer Guide.

`updateDataflowJob`

Updates a CRM Analytics dataflow job, which is the equivalent of clicking **Stop** for a data prep recipe, a data sync, or a dataflow in the CRM Analytics Data Manager UI. For more information on syntax and usage, see the updateDataflowJob reference in the Lightning Web Components Developer Guide.

`updateDataset`

Updates a specific CRM Analytics dataset by ID or developer name. For more information on syntax and usage, see the updateDataset reference in the Lightning Web Components Developer Guide.

`updateDatasetVersion`

Updates a specific CRM Analytics dataset version by ID or developer name. For more information on syntax and usage, see the updateDatasetVersion reference in the Lightning Web Components Developer Guide.

`updateRecipe`

Updates a CRM Analytics recipe. For more information on syntax and usage, see the updateRecipe reference in the Lightning Web Components Developer Guide.

`updateReplicatedDataset`

Updates a CRM Analytics replicated dataset by ID. For more information on syntax and usage, see the updateReplicatedDataset reference in the Lightning Web Components Developer Guide.

`updateReplicatedFields`

Updates the collection of fields for a CRM Analytics replicated dataset by ID. For more information on syntax and usage, see the updateReplicatedFields reference in the Lightning Web Components Developer Guide.

`updateSchedule`

Updates the schedule for a CRM Analytics data prep recipe, data sync, or dataflow. For more information on syntax and usage, see the updateSchedule reference in the Lightning Web Components Developer Guide.

`updateXmd`

Updates the user Xmd for a CRM Analytics dataset. For more information on syntax and usage, see the updateXmd reference in the Lightning Web Components Developer Guide.

`validateWaveTemplate`

Validates a CRM Analytics template for org readiness. For more information on syntax and usage, see the valildateWaveTemplate reference in the Lightning Web Components Developer Guide.

CRM Analytics Assets SDK Aura Component

Discover CRM Analytics dashboards, lenses and datasets, get their details, and dig into dataset fields. Discover dashboard saved views and explore dashboard state. Create and execute queries directly on datasets

Call the SDK

To call the SDK, declare the `wave:sdk` component in your component or app.

```
<wave:sdk aura:id="sdk"/>
```

Use `sdk.invokeMethod` to specify the method and any parameters.

```
sdk.invokeMethod(context, methodName, methodParameters, callback)
```

For example, here's a call that uses `listDashboards` as *`methodName`*.

```
var context = {};
   var methodName = 'listDashboards';
   var methodParameters = {
     'pageSize' : 200,
     'sort' : 'Name'
   };

   sdk.invokeMethod(context, methodName, methodParameters,
   $A.getCallback(function (err, data) {
   if (err !== null) {
     //DO THIS IF THE METHOD FAILS
     console.error("SDK error", err);
   } else {
     //DO THIS IF THE METHOD SUCCEEDS
     component.set('v.dashboards', data.dashboards);
   }
}))
```

See the wave:sdk Component Reference for a full working example.

Each method has its own set of parameters (*`methodParameters`*).

Methods and Parameters

`listDashboards`

Retrieves a list of all dashboards. Can be filtered by specifying parameters

Parameter Name	Description	Required	Type
`folderId`	Filters the results to include only the contents of a specific folder.	FALSE	String, base platform object ID format
`page`	A generated token that indicates the view of the dashboards to be returned	FALSE	String

Parameter Name	Description	Required	Type
`pageSize`	Number of items to be returned in a single page. Default is 25 and maximum is 200.	FALSE	Integer
`q`	Search terms. Individual terms are separated by spaces. Wild cards aren't supported.	FALSE	String
`sort`	Sort order of the results. Enum values are `LastModified`, `MRU`, and `Name`. Default value is `MRU`.	FALSE	String
`scope`	Type of scope to be applied to the returned items (`CreatedByMe` or `SharedWithMe`)	FALSE	String
`type`	Asset type	FALSE	String
`templateApiName`	Filter collection by `templateApiName`.	FALSE	String
`mobileOnly`	For mobile dashboards only.	FALSE	String

`listLenses`

Retrieves a list of all lenses. Can be filtered by specifying parameters

Parameter Name	Description	Required	Type
`folderId`	Filters the results to include only the contents of a specific folder.	FALSE	String, base platform object ID format
`page`	A generated token that indicates the view of the lenses to be returned	FALSE	String
`pageSize`	Number of items to be returned in a single page. Default is 25 and maximum is 200.	FALSE	Integer
`q`	Search terms. Individual terms are separated by spaces. Wild cards aren't supported.	FALSE	String
`sort`	Sort order of the results. Enum values are `LastModified`, `MRU`, and `Name`. Default value is `MRU`.	FALSE	String

Parameter Name	Description	Required	Type
`scope`	Type of scope to be applied to the returned items (`CreatedByMe` or `SharedWithMe`)	FALSE	String

`listDatasets`

Retrieves a list of all datasets. Can be filtered by specifying parameters

Parameter Name	Description	Required	Type
`folderId`	Filters the results to include only the contents of a specific folder.	FALSE	String, base platform object ID format
`hasCurrentOnly`	Filters the list of datasets to include only those datasets that have a current version. The default is `false`.	FALSE	String, base platform object ID format
`page`	A generated token that indicates the view of the dashboards to be returned	FALSE	String
`pageSize`	Number of items to be returned in a single page. Default is 25 and maximum is 200.	FALSE	Integer
`q`	Search terms. Individual terms are separated by spaces. Wild cards aren't supported.	FALSE	String
`sort`	Sort order of the results. Enum values are `LastModified`, `MRU`, and `Name`. Default value is `MRU`.	FALSE	String
`scope`	Type of scope to be applied to the returned items (`CreatedByMe` or `SharedWithMe`)	FALSE	String

`describeDashboard`

Retrieves the details of a single dashboard.

Parameter Name	Description	Required	Type
`dashboardId`	15 or 18-digit id of the dashboard.	TRUE	String, base platform object ID format

`describeLens`

Retrieves the details of a single lens.

Parameter Name	Description	Required	Type
lensId	15 or 18-digit id of the lens.	TRUE	String, base platform object ID format

`describeDataset`

Retrieves the details of a single dataset.

Parameter Name	Description	Required	Type
datasetId	15 or 18-digit id of the dataset.	TRUE	String, base platform object ID format

`getDatasetFields`

Retrieves a list of all the fields for a single dataset.

Parameter Name	Description	Required	Type
`datasetId`	15 or 18-digit id of the dataset.	TRUE	String, base platform object ID format
`versionId`	15 or 18-digit version id of the dataset.	TRUE	String, base platform object ID format

`executeQuery`

Executes a CRM Analytics SAQL query.

Parameter Name	Description	Required	Type
`query`	The SAQL query to execute, in JSON format.	TRUE	String, base platform object ID format

`listDashboardSavedViews`

Retrieves a list of all saved views for a dashboard.

Parameter Name	Description	Required	Type
`dashboardIdOrApiName`	The 15 or 18-digit id or the fully qualified name of the dashboard.	TRUE	String, base platform object ID format

`getDashboardSavedView`

Retrieves the detail of one dashboard saved view.

Parameter Name	Description	Required	Type
`dashboardIdOrApiName`	The 15 or 18-digit id or the fully qualified name of the dashboard.	TRUE	String, base platform object ID format
`viewId`	The 15 or 18-digit id of the saved view.	TRUE	String, base platform object ID format

`getDashboardInitialSavedView`

Retrieves the initial view information for a dashboard saved view.

Parameter Name	Description	Required	Type
`dashboardIdOrApiName`	The 15 or 18-digit id or the fully qualified name of the dashboard.	TRUE	String, base platform object ID format
`viewId`	The 15 or 18-digit id of the initial saved view.	TRUE	String, base platform object ID format

CRM Analytics Template SDK Aura Component

Discover CRM Analytics templates and apps created from them. Create, update, and delete CRM Analytics apps created from templates.

Call the SDK

To call the SDK, declare the `wave:sdk` component in your component or app.

```
<wave:sdk aura:id="sdk"/>
```

Use `sdk.invokeMethod` to specify the method and any parameters.

```
sdk.invokeMethod(context, methodName, methodParameters, callback)
```

For example, here's a call that uses `listFolders` as *`methodName`*.

```
var context = {};
var methodName = 'listFolders';
var methodParameters = {
    'templateSourceId' : templateSourceId,
    'pageSize' : pageSize,
    'q' : q,
    'sort' : sort,
    'scope' : scope,
    'page' : page,
    'isPinned' : isPinned,
    'mobileOnly' : false
};
```

```
sdk.invokeMethod(context, methodName, methodParameters,
                 $A.getCallback(function (err, data) {
    if (err !== null) {
        //DO THIS IF THE METHOD FAILS
        console.error("SDK error", err);
    } else {
        //DO THIS IF THE METHOD SUCCEEDS
        component.set('v.folders', data.folders);
    }
}))
```

Each method has its own set of parameters (*methodParameters*)—except listTemplates, which doesn't need any.

Methods and Parameters

`listTemplates`

Retrieves a list of all templates that the user has access to. Has no parameters.

`getTemplate`

Retrieves information for the specified template.

Parameter Name	Description	Required	Type
`templateId`	ID of the template.	TRUE	String, in either the internal `'sfdc_internal__'` or base platform object ID format

`getTemplateConfig`

Retrieves variable and UI information for the template.

Parameter Name	Description	Required	Type
`templateId`	ID of the template.	TRUE	String, in either the internal `'sfdc_internal__'` or base platform object ID format

`validateWaveTemplate`

Runs validation on a CRM Analytics template to check for org readiness.

Parameter Name	Description	Required	Type
`templateId`	ID of the template.	TRUE	String, in either the internal `'sfdc_internal__'` or base platform object ID format
`validationInput`	The input to use during the validation.	TRUE	A map of template variable values to validate the template with. These values override any template defaults.

`createFolder`

Creates a folder from the specified template. Not used for apps that aren't created from templates.

Parameter Name	Description	Required	Type
`name`	Developer name of the folder.	TRUE	String
`label`	Display label of the folder.	TRUE	String
`description`	Description of the folder.	FALSE	String
`dynamicOptions`	Map of configuration options for folder creation.	FALSE	Object; map of name-value pairs, such as `{'runtimeLogEntryLevel' : 'Warning'}`. For supported names and values, see Template Options Input.
`templateSourceId`	ID of the template used to create the folder.	TRUE	String, in either the internal `'sfdc_internal__'` or base platform object ID format
`templateValues`	Variable values for the template.	FALSE	Object; map of name-value pairs, such as `{'Can_Continue' : true}`

`listFolders`

Retrieves a list of folders that the user has access to. Can be filtered by specifying parameters.

Parameter Name	Description	Required	Type
`templateSourceId`	Limit the results to folders created from the specified template.	FALSE	String, in either the internal `'sfdc_internal__'` or base platform object ID format
`pageSize`	Limit the results by page.	FALSE	Number
`q`	Limit the results by query string.	FALSE	String
`sort`	Sort the results alphabetically or by most recently used.	FALSE	Enum: `'alpha'` or `'mru'`
`scope`	Return only folders within the specified scope.	FALSE	Enum: `'createdByMe'` or `'sharedWithMe'`
`page`	Return the specified page of results.	FALSE	Number
`isPinned`	Limit the results to pinned or unpinned folders.	FALSE	Boolean
`mobileOnly`	Limit the results to mobileOnly folders.	FALSE	Boolean

`updateFolder`

Updates a folder's metadata. Can cancel an in-progress app.

Parameter Name	Description	Required	Type
`folderId`	ID of the folder.	TRUE	String
`label`	New display label for the folder.	FALSE	String
`description`	New description for the folder.	FALSE	String
`applicationStatus`	When specified, cancels an app that is in progress.	FALSE	`'cancelledstatus'` is the only value

`upgradeFolder`

Resets or upgrades the specified folder. If the version of the template matches the version used to create the app, resets the app using the template. If the versions don't match, upgrades the app using the new template version. Can specify dynamic options or template values that differ from the values used when the app was originally created.

Parameter Name	Description	Required	Type
`dynamicOptions`	Map of configuration options for folder upgrade.	FALSE	Object; map of name-value pairs, such as `{'runtimeLogEntryLevel' : 'Warning'}`. For supported names and values, see Template Options Input.
`folderId`	ID of the folder.	TRUE	String
`templateValues`	Variable values for the template.	FALSE	Object; map of name-value pairs, such as `{'Can_Continue' : true}`

`deleteFolder`

Deletes the specified folder.

Parameter Name	Description	Required	Type
`folderId`	ID of the folder.	TRUE	String

CRM Analytics Aura Events

Easily interact with the embedded CRM Analytics Dashboard component in custom applications. Use Aura Events for Lightning Apps, Visualforce, or your preferred development environment.

CRM Analytics Aura Events

`wave:assetLoaded`

A CRM Analytics asset fires this event when the asset is finished loading. The payload contains the asset type and the asset id. For a CRM Analytics dashboard asset, the event is fired: on the initial load of a dashboard, when a user resets to dashboard to the initial view, and when the user selects a dashboard view. After this event is received, you can safely reapply mandatory filters or resync the dashboard state.

`wave:update`

Use this event to dynamically set the filter on a CRM Analytics dashboard or interact with the dashboard by dynamically changing the selection. It has four attributes: the unique ID of the CRM Analytics asset on which to apply the filter, the payload, the asset type (currently only dashboard), and the fully qualified developer name of the CRM Analytics dashboard. The payload is a JSON string that identifies the datasets and any dimensions and field values.

`wave:selectionChanged`

A CRM Analytics dashboard fires this event for consumption by custom Aura components. It provides the following attributes: the ID of the dashboard that fired the event and the payload. The payload object contains the selection information—the name of the step involved when changing the selection and an array of objects representing the current selection. Each object in the array contains one or more attributes based on the selection.

`wave:discover`

This event sends a global request to identify CRM Analytics dashboard assets. The response is a `wave:discoverResponse` event. You can include your own parameter in this event that is included in the response payload.

`wave:discoverResponse`

This event is fired by listening to CRM Analytics dashboard assets in response to the `wave:discover` event. The payload includes the dashboard identifier, the type of component, the dashboard title, whether the dashboard is still loading, and any optional parameter sent with the request.

`wave:pageChange`

Use this event to update the CRM Analytics dashboard page that is displayed. It has two attributes: the unique ID of the page to display and the fully qualified developer name of the CRM Analytics dashboard

CRM Analytics Aura Events - Update Event

Create a custom component to dynamically set filters in a CRM Analytics dashboard embedded in a Lightning page.

CRM Analytics Aura Events - SelectionChanged Event

React to selections in your dashboard and get the row data for the selection.

CRM Analytics Aura Events - Discover Event

This event sends a request to CRM Analytics dashboards to identify their assets.

CRM Analytics Aura Events - Discover Response Event

This event provides the response following a request for CRM Analytics dashboards to identify their assets.

CRM Analytics Aura Events - Page Change Event

This event sends a request to CRM Analytics dashboards to change the displayed page.

CRM Analytics Aura Events - Asset Loaded Event

React to the CRM Analytics asset rendering completion event.

CRM Analytics Aura Events - Update Event

Create a custom component to dynamically set filters in a CRM Analytics dashboard embedded in a Lightning page.

Example - Setting a Filter with the Update Event

This event works with embedded dashboard components. Embed your CRM Analytics dashboard in a Lightning page (see the Embed CRM Analytics Dashboards in Lightning Pages help topic for more information). Be sure to save and activate your page.

The CRM Analytics Aura events allow CRM Analytics to interact with the UI container. In this example, we create a custom component to interact with the embedded CRM Analytics dashboard, so you need some familiarity with the Developer Console. See the Lightning Aura Components Developer Guide for more information.

In the Developer Console, create an Aura Component named filterTest, and copy the following into the component markup definition (`filterTest.cmp`):

```
<aura:component implements="force:appHostable,
                            flexipage:availableForAllPageTypes,
                            flexipage:availableForRecordHome,
                            force:hasRecordId,
                            forceCommunity:availableForAllPageTypes,
                            force:lightningQuickAction"
                            access="global" >
  <aura:attribute name="filter" type="String" access="GLOBAL"/>
  <aura:attribute name="developerName" type="String" access="GLOBAL" default="XXXXXXXXXXX"/>

  <aura:handler name="init" value="{!this}" action="{!c.doInit}"/>
  <aura:registerEvent name="update" type="wave:update"/>
  <div class="container">
    <ui:inputText value="{!v.filter}" label="Filter: " size="200"></ui:inputText>
    <ui:button press="{!c.handleSendFilter}" label="Fire"/>
  </div>
</aura:component>
```

Replace the `XXXXXXXXXXX` with the fully qualified developer name of your dashboard. To find the name, use uCRL or Postman to execute the API call */services/data/v47.0/wave/dashboards*. The developer name is required and it must be the fully qualified name - `<namespace>__<devName>`.

For most filters, you need the fully qualified name of the dataset the dashboard is using. To find the name, log in to Workbench, click **Utilities** | **REST Explorer**. In the text box, type */services/data/v47.0/wave/datasets* and then click **Execute**.

Add a controller (`filterTestController.js`) to the bundle, then copy the following JavaScript into it. This example JavaScript shows how to construct the payload for the update event—in this case, setting `StageName` to `Closed Won` for the dataset used by the dashboard. Replace these names with valid names from your dashboard and dataset, for the filter you wish to set. For more information about creating the filter, see Filter and Selection Syntax for Embedded Dashboards.

```
({
  doInit: function(component, event, helper) {
    component.set('v.filter', '{"datasets" : "<namespace>__<datasetName>": [{"fields":
["StageName"], "selection": ["Closed Won"]}]}}');
  },
  handleSendFilter: function(component, event, helper) {
    var filter = component.get('v.filter');
    var developerName = component.get('v.developerName');
    var evt = $A.get('e.wave:update');
```

```
    evt.setParams({
      value: filter,
      devName: developerName,
      type: "dashboard"
    });
    evt.fire();
  }
})
```

Add a Style (`filterTest.css`) to the bundle and copy the following CSS into it.

```
.THIS.container {
  border: 1px solid #A0A0A0;
  padding: 5px;
  width: 100%;
  margin: 5px auto;
  background: white;
}

.THIS .uiInputText {
  display: inline-block;
  margin-right: 5px;
}
```

Finally, add a design (`filterTest.design`) to the bundle and copy the following into it.

```
<design:component label="Filter Test">
  <design:attribute name="filter" label="Filter" description="The initial filter"/>
  <design:attribute name="developerName"
     label="Dashboard Name"
     description="The CRM Analytics Dashboard to send the filter to"/>
</design:component>
```

That's it. You can use your custom component to interact with CRM Analytics. Add your custom component to the Lightning Page with your embedded dashboard. Make sure that the developer name in the filter matches that of the dataset in the dashboard you embedded. Applying your filter by clicking the "Fire" button causes the dashboard to be updated.

Resources

For more information about Aura events and other Lightning development features, see the Lightning Aura Components Developer Guide.

CRM Analytics Aura Events - SelectionChanged Event

React to selections in your dashboard and get the row data for the selection.

Example - Reacting to a Selection with the selectionChanged Event

The dashboard component also generates Lightning events when the user changes a selection. The payload for these events is effectively the row data for the current selection. Datasets can be from many sources, so the actual payload may only be meaningful when the user has knowledge of the datasets being used. This example shows how to receive and iterate through the payload, which is an array of objects representing the current selection.

This example uses the same dashboard used for the update event example, so be sure to follow those steps first.

Using the Developer Console, create an Aura component named recordView. Copy the following markup into the component.

```
<aura:component implements="force:appHostable,
                flexipage:availableForAllPageTypes,
                flexipage:availableForRecordHome,
                force:hasRecordId,
                forceCommunity:availableForAllPageTypes,
                force:lightningQuickAction"
                access="global" >
  <aura:handler event="wave:selectionChanged" action="{!c.handleSelectionChanged}"/>
  <aura:attribute name="msg" type="String" default="Please make a selection in CRM Analytics
 that contains a record ID" access="GLOBAL"/>
  <aura:attribute name="recordId" type="String" default="" access="GLOBAL"/>
  <aura:dependency resource="markup://force:navigateToSObject" type="EVENT"/>
  <div class="container">
    <aura:if isTrue="{!v.recordId == ''}">
      <div class="msg">
        {!v.msg}
      </div>
      <aura:set attribute="else">
        <force:recordView recordId="{!v.recordId}"/>
      </aura:set>
    </aura:if>
  </div>
</aura:component>
```

Add a Controller (recordViewController.js) to the bundle and copy the following JavaScript into it:

```
({
  handleSelectionChanged: function(component, event, helper) {
    var params = event.getParams();
    var payload = params.payload;
    if (payload) {
      var step = payload.step;
      var data = payload.data;
      data.forEach(function(obj) {
        for (var k in obj) {
          if (k === 'Id') {
            component.set("v.recordId", obj[k]);
          }
        }
      });
    }
  }
})
```

This example references recordId, an Opportunity record identifier. If you don't use this in your dashboard, substitute a different field.

Payload data can contain other objects, each in turn containing key-value pairs. For example, aside from the Id, you can also get the noun (for example, "dashboard") and the verb (for example, "selection").

Add a Style (recordView.css) to the bundle and copy the following CSS into it:

```
.THIS.container {
  min-height: 650px;
  min-width: 200px;
  height: 100%;
```

```
  width: 100%;
  border: 1px solid #A0A0A0;
  margin: 5px auto;
}

.THIS .msg {
  vertical-align: middle;
  text-align: center;
  margin: 2em auto;
}
```

For a better experience in Lightning App Builder, add a Design (recordView.design) to the bundle and paste in the following:

```
<design:component label="Record View">
  <design:attribute name="recordId" label="Record ID" description="ID of the record"/>
  <design:attribute name="msg" label="Message" description="Message to display"/>
</design:component>
```

The page you created using Lightning App Builder should now show the Record View component in the palette. Drag this component onto the page, then save the page.

Go back to Lightning Experience, and make a selection in your CRM Analytics Dashboard component. The corresponding Salesforce Opportunity record (or the record type you specified in recordViewController.js) will be displayed in the newly added component.

Resources

For more information about Aura events and other Lightning development features, see the Lightning Aura Components Developer Guide.

CRM Analytics Aura Events - Discover Event

This event sends a request to CRM Analytics dashboards to identify their assets.

The *wave:discover* event sends a global request to listening CRM Analytics dashboard assets to respond with their identifying information (via the *wave:discoverResponse* event). You can include your own parameter in the response.

The *wave:discover* and *wave:discoverResponse* events work hand-in-hand. They're particularly useful for discovering when a dashboard is being added dynamically to the page, or whether the page has multiple dashboards.

Example - Setting Up Your Request and Receiving a Response

Using the Developer Console, create an Aura component and copy the following markup into the component. The markup sets up the handlers for the events, and adds buttons for adding a dashboard and for discovering dashboards.

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <aura:handler event="wave:discoverResponse" action="{!c.handleDiscoverResponse}"/>
    <aura:registerEvent name="discoverEvent" type="wave:discover"/>

    <ui:inputText label="Dashboard Name" aura:id="idTextBox"/>
    <ui:button label="Add Dashboard" press="{!c.addDashboard}"/>
    <ui:button label="Are you there?" press="{!c.discoverDashboard}"/>
    {!v.body}
    <ui:outputText aura:id="outName" value="" class="text"/>
</aura:component>
```

Add a controller to the bundle, then copy the following JavaScript into it. This code shows how to fire the *discover* event, and how to use the result when the *discoverResponse* event is fired. The code also shows how to create dashboard components.

```
({
    addDashboard: function(component, event, helper) {
        var selectCmp = component.find("idTextBox");
        var config = {
            "developerName": selectCmp.get("v.value"),
            "showHeader": false,
            "height": 400
        };
        $A.createComponent("wave:waveDashboard", config,
            function(dashboard, status, err) {
                if (status === "SUCCESS") {
                    dashboard.set("v.rendered", true);
                    dashboard.set("v.showHeader", false);
                    component.set("v.body", dashboard);
                } else if (status === "INCOMPLETE") {
                    console.log("No response from server or client is offline.")
                } else if (status === "ERROR") {
                    console.log("Error: " + err);
                }
            }
        );
    },
    discoverDashboard: function(component, event, helper) {
        $A.get("e.wave:discover").fire();
    },
    handleDiscoverResponse: function(cmp, event, helper) {
        var myText = cmp.find("outName");
        myText.set("v.value", event.getParam("developerName"));
    },
})
```

That's it! You can use these events to get some context about available dashboard components, and then interact with them via the *Update* and *selectionChanged* events.

Resources

For more information about Aura events and other Lightning development features, see the Lightning Aura Components Developer Guide.

CRM Analytics Aura Events - Discover Response Event

This event provides the response following a request for CRM Analytics dashboards to identify their assets.

Example - Setting Up Your Request and Receiving a Response

Refer to the *wave:discover* on page 19 event for details and an example using *wave:discoverResponse*.

Resources

For more information about Aura events and other Lightning development features, see the Lightning Aura Components Developer Guide.

CRM Analytics Aura Events - Page Change Event

This event sends a request to CRM Analytics dashboards to change the displayed page.

The *wave:pageChange* event sends a global request to a listening CRM Analytics dashboard to update the page that is displayed.

Example - Updating the Displayed Dashboard Page

This event works with embedded dashboard components. Embed your CRM Analytics dashboard in a Lightning page (see the Embed CRM Analytics Dashboards in Lightning Pages help topic for more information). Be sure to save and activate your page.

The CRM Analytics Aura events allow CRM Analytics to interact with the UI container. In this example, we create a custom component to interact with the embedded CRM Analytics dashboard, so you need some familiarity with the Developer Console. See the Lightning Aura Components Developer Guide for more information.

Using the Developer Console, create an Aura component and copy the following markup into the component. The markup sets up the handler for the event, input fields for the event parameters, and adds a button for firing the *pageChange* event.

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <aura:attribute name="pageId" type="String" access="GLOBAL" default="page_one"/>
    <aura:attribute name="developerName" type="String" access="GLOBAL"
default="XXXXXXXXXXXXXX"/>
    <aura:registerEvent name="pageChange" type="wave:pageChange"/>

    <div class="container">
        <div class="slds-form-element">
            <label class="slds-form_element__label" for="developerName">Developer Name:
</label>
            <lightning:textarea name="developerName" value="{!v.developerName}"/>
        </div>
        <div class="slds-form-element">
            <label class="slds-form_element__label" for="pageId">Page Id: </label>
            <lightning:textarea name="pageId" value="{!v.pageId}"/>
        </div>
        <div class="slds-form-element">
            <lightning:button onclick="{!c.handleSendPageChange}" label="Fire"/>
        </div>
    </div/>
</aura:component>
```

Replace the `XXXXXXXXXX` with the fully qualified developer name of your dashboard. To find the name, use cURL or Postman to execute the REST API call */services/data/v47.0/wave/dashboards*. The developer name is required and it must be the fully qualified name - `<namespace>__<devName>`.

Add a controller to the bundle, then copy the following JavaScript into it. This code shows how to fire the *pageChange* event.

```
({
    handleSendPageChange : function(component, event, helper) {
        var pageId = component.get('v.pageId');
        var developerName = component.get('v.developerName');
```

```
        var evt = $A.get('e.wave:pageChange')'
        var params = {
            devName: developerName,
            pageid: pageId
        };
        evt.setParams(params);
        evt.fire();
    }
})
```

That's it. You can use your custom component to interact with a CRM Analytics dashboard. Add your custom component to the Lightning Page with your embedded dashboard. Make sure that the *pageId* matches that of a *pageId* in the dashboard you embedded. Change the dashboard pages by clicking the "Fire" button and watch the dashboard update.

Resources

For more information about Aura events and other Lightning development features, see the Lightning Aura Components Developer Guide.

CRM Analytics Aura Events - Asset Loaded Event

React to the CRM Analytics asset rendering completion event.

Example - Reacting to Dashboard Rendering Completion with the Asset Loaded Event

For this example, the CRM Analytics asset used is a dashboard. The dashboard component generates a Lightning event when it has fully loaded and rendered its state (widgets, steps, and queries). This event is sent: on the initial load of a dashboard, when a user resets to dashboard to the initial view, and when the user selects a dashboard view. The payload for this completion event is the asset type and asset id. This example shows how to receive the payload and display the results.

This event works with embedded dashboard components. Embed your CRM Analytics dashboard in a Lightning page (see the Embed CRM Analytics Dashboards in Lightning Pages help topic for more information). Be sure to save and activate your page.

Using the Developer Console, create an Aura component named assetLoaded. Copy the following markup into the component.

```
<aura:component
     implements="force:appHostable,flexipage:availableForAllPageTypes,
     flexipage:availableForRecordHome,force:hasRecordId"
     access="global" >
  <aura:attribute name="dashboardStatus" type="String" access="GLOBAL"
      default="Loading dashboard..."/>
  <aura:attribute name="assetType" type="String" default=""/>
  <aura:attribute name="assetId" type="String" default=""/>

  <aura:handler event="wave:assetLoaded" action="{!c.handleAssetLoaded}"/>

  <div class="container">
    <div class="slds-form-element">
      <label class="slds-form_element__label" for="filter">Dashboard Status:</label>
      <ui:inputText value="{!v.dashboardStatus}" />
    </div>
    <div class="slds-form-element">
```

```
      <label class="slds-form_element__label" for="assetType">Loaded Asset Type:</label>
      <ui:inputText value="{!v.assetType}" />
    </div>
    <div class="slds-form-element">
      <label class="slds-form_element__label" for="assetId">Loaded Asset Id:</label>
      <ui:inputText value="{!v.assetId}" />
    </div>
  </div>
</aura:component>
```

Add a Controller (assetLoaded.js) to the bundle and copy the following JavaScript into it:

```
({
  handleAssetLoaded: function(component, event, helper){
    component.set("v.dashboardStatus", "Dashboard is loaded");
    component.set("v.assetType", event.getParam("type"));
    component.set("v.assetId", event.getParam("id"));
  }
})
```

This example listens for any asset to be loaded and then displays the type and id of the asset. For greater functionality, combine this code with the `wave:selectionChanged` event example code to build a component that listens for asset loaded events and informs the user of the dashboard status so the user knows when to safely make updates to the dashboard.

You can now use your custom component to interact with a CRM Analytics dashboard component. In the Lightning App Builder, add your custom component to the Lightning Page with your embedded dashboard. That's it. Save and view the page to see the `wave:assetLoaded` event in action.

Resources

For more information about Aura events and other Lightning development features, see the Lightning Aura Components Developer Guide.

CRM Analytics in Apex

Build and execute queries and retrieve lens and template data using CRM Analytics Apex classes and methods.

CRM Analytics in Apex currently consists of the `executeQuery` method, the `Wave.QueryBuilder` class, and the `Wave.Lenses` class.

`ConnectApi.Wave.executeQuery`

Use the `executeQuery` function (exposed through the ConnectApi namespace) to pass a SAQL query from an Apex page to CRM Analytics, and get a response in the form of JSON.

`Wave.QueryBuilder`

The `QueryBuilder` class is the most convenient, preferred, and safest way to construct a SAQL query string for execution.

`Wave.Lenses`

Retrieve a collection of CRM Analytics lens assets and describe a single lens asset.

`Wave.Templates`

Retrieve a collection of CRM Analytics templates, describe a single template and template configuration.

Resources

For more information on using Apex, see the Apex Developer Guide.

CRM Analytics Apex Lens

Use the `Lenses` class to retrieve a collection of CRM Analytics lens assets and to describe a single lens asset.

CRM Analytics Apex Query

Query your data in CRM Analytics from any Apex class. Construct well-formed queries using the query builder.

CRM Analytics Apex QueryBuilder Examples

Build simple or complex SAQL queries using QueryBuilder.

CRM Analytics Apex Templates

Use the `Templates` class to retrieve a collection of CRM Analytics templates, describe a single template and template configuration.

CRM Analytics Apex Lens

Use the `Lenses` class to retrieve a collection of CRM Analytics lens assets and to describe a single lens asset.

The CRM Analytics `Wave.Lenses` class provides access from Apex to the CRM Analytics lens assets.

`Wave.Lenses` collection methods

`Map<String, Object> getLenses()`

Retrieves a collection of lenses.

`Map<String, Object> getLenses(String searchOptions)`

Retrieves a collection of lenses using search options.

Example: Apex class example

```
public with sharing class LensController {
  public LensController() {
  }

  @AuraEnabled(cachable=true)
  public static Map<String, Object> getLenses() {
    // This has all fields available, plus filterGroup; all fields are optional/nullable

    Wave.LensesSearchOptions options = new Wave.LensesSearchOptions();
    options.q = 'widget';
    options.filterGroup = 'supplemental';
    options.scope = 'CreatedByMe';
    options.page = null;
    options.sortParam = 'Name';

    // Pass null to get the default search options or leave it off completely to return

    // a collection with no search options
    Map<String, Object> lensesJson = Wave.Lenses.getLenses(options);

    // lensesJson is the JSON response as an Apex Map (from JSON.deserializedUntyped),
```

```
 which
    // you can pull fields from
    return lensesJson;
  }
}
```

Example: LWC example

```
import {LightningElement, wire} from "lwc";
import getLenses from "@salesforce/apex/Wave.Lenses.getLenses";

export default class Lenses extends LightningElement {
  results;

  @wire(getLenses, {
    options: {
      // All are optional
      filterGroup = "Supplemental",
      sortParam = "Name"
    }
  })
  // can also use these
  // @wire(getLenses, { options: {} })
  // @wire(getLenses, {})
  // @wire(getLenses)
  // @wire(getLenses, { options: {'$options'} }) // with a binding
  onLenses({data, error}) {
    if (error) {
      this.results = "Error:\n" + JSON.stringify(error, undefined, 2);
    } else if (data) {
      // data is the LensCollectionRepresentation JSON object
      this.results = "Lenses: " + data.lenses.map(l => `${l.name} {$l.id}`).join(",
");
    } else {
      this.results = "No data";
    }
  }
}
```

`Wave.Lenses` describe methods

`Map<String, Object> getLens(String lensIdOrApiName)`

Retrieves a lens by ID or the API name.

`Map<String, Object> getLens(String lensIdOrApiName, String filterGroup)`

Retrieves a lens by ID or the API name and a `filterGroup` parameter.

Example: Apex class example

```
public with sharing class LensController {
  public LensController() {
  }
```

```
  @AuraEnabled(cacheable=true)
  public static Map<String, Object> getLens(String idOrName) {
    Map<String, Object> lens = Wave.Lenses.getLens(idOrName);
    return lens;
  }
}
```

Example: LWC example

```
import {LightningElement, wire} from "lwc";
import getLenses from "@salesforce/apex/Wave.Lenses.getLens";

export default class Lens extends LightningElement {
  lensIdOrApiName; // set this to the ID or name you want to retrieve

  results;

  @wire(getLens, {
    lensIdOrApiName: '$lensIdOrApiName'
  })
  onLens({data, error}) {
    if (error) {
      this.results = "Error:\n" + JSON.stringify(error, undefined, 2);
    } else if (data) {
      // data is the LensRepresentation JSON object
      this.results = `Lens: ${data.name} ${data.id})`;
    } else {
      this.results = "No data";
    }
  }
}
```

For information on request parameters and the Lens JSON responses, see the Analytics REST API Developer Guide: Lenses List Resource.

CRM Analytics Apex Query

Query your data in CRM Analytics from any Apex class. Construct well-formed queries using the query builder.

The Apex SDK query features include the `executeQuery` method and the `Wave.QueryBuilder` class.

`ConnectApi.Wave.executeQuery`

At its simplest, use the `executeQuery` function (exposed through the ConnectApi namespace) to pass a SAQL query from an Apex page to CRM Analytics, and get a response in the form of JSON. For example, this sample sends 'your SAQL query' to CRM Analytics.

```
String query = '[your SAQL query]';
   ConnectApi.LiteralJson result =  ConnectApi.Wave.executeQuery(query);
   String response = result.json;
```

Sending queries like this is useful, but relies on the developer coding well-formed SAQL queries. Wouldn't it be great if a class constructed the queries for you?

`Wave.QueryBuilder`

The `Wave.QueryBuilder` class is the most convenient, preferred, and safest way to construct a SAQL query string for execution. It's not an exhaustive implementation of all possible SAQL queries—so sometimes you must write your own—but it does cover the vast majority of use cases, including:

- load dataset statement
- foreach statement
- group statement
- order statement
- limit statement
- filter statement
- functions such as min, max, count, avg, unique, as, sum

Use `Wave.QueryBuilder` and its associated classes, `Wave.ProjectionNode` and `Wave.QueryNode`, to incrementally build your SAQL statement. For example:

```
Wave.ProjectionNode[] projs = new Wave.ProjectionNode[]{
   Wave.QueryBuilder.get('State').alias('State'),
   Wave.QueryBuilder.get('City').alias('City'),
   Wave.QueryBuilder.get('Revenue').avg().alias('avg_Revenue'),
   Wave.QueryBuilder.get('Revenue').sum().alias('sum_Revenue'),

   Wave.QueryBuilder.count().alias('count')};
   ConnectApi.LiteralJson result = Wave.QueryBuilder.load('0FbD00000004DSzKAM',
'0FcD00000004FEZKA2')
   .group(new String[]{'State', 'City'})
   .foreach(projs)
   .execute('q');
   String response = result.json;
```

CRM Analytics Apex QueryBuilder Examples

Build simple or complex SAQL queries using QueryBuilder.

QueryBuilder is the core of the CRM Analytics Apex feature set, so let's take a closer look.

1. Here's a simple count query:

```
Wave.ProjectionNode[] projs = new
Wave.ProjectionNode[]{Wave.QueryBuilder.count().alias('c')};
String query = Wave.QueryBuilder.load('datasetId',
'datasetVersionId').group().foreach(projs).build('q');
```

Output:

```
q = load "datasetId/datasetVersionId";
q = group q by all;
q = foreach q generate count as c;
```

2. Query selecting specific attributes and using aliases.

```
Wave.ProjectionNode[] projs = new Wave.ProjectionNode[]{Wave.QueryBuilder.get('Name'),
Wave.QueryBuilder.get('AnnualRevenue').alias('Revenue')};
```

```
String query =
Wave.QueryBuilder.load('datasetId','datasetVersionId').foreach(projs).build('q');
```

Output:

```
q = load "datasetId/datasetVersionId";
q = foreach q generate Name,AnnualRevenue as Revenue;
```

3. Query using a filter condition.

```
Wave.ProjectionNode[] projs = new Wave.ProjectionNode[]{Wave.QueryBuilder.get('Name'),
Wave.QueryBuilder.get('AnnualRevenue').alias('Revenue')};
String query =
Wave.QueryBuilder.load('datasetId','datasetVersionId').foreach(projs).filter('Name != \'My
 Name\'').build('q');
```

Output:

```
q = load "datasetId/datasetVersionId";
q = foreach q generate Name,AnnualRevenue as Revenue;
q = filter q by Name != 'My Name';
```

4. Query with a limit statement.

```
Wave.ProjectionNode[] projs = new Wave.ProjectionNode[]{Wave.QueryBuilder.get('Name'),
Wave.QueryBuilder.get('AnnualRevenue').alias('Revenue')};
String query =
Wave.QueryBuilder.load('datasetId','datasetVersionId').foreach(projs).cap(10).build('q');
```

Output:

```
q = load "datasetId/datasetVersionId";
q = foreach q generate Name,AnnualRevenue as Revenue;
q = limit q 10;
```

5. Query with an order statement.

```
Wave.ProjectionNode[] projs = new Wave.ProjectionNode[]{Wave.QueryBuilder.get('Name'),
Wave.QueryBuilder.get('AnnualRevenue').alias('Revenue')};
List<List<String>> orders = new List<List<String>>{new List<String>{'Name', 'asc'}, new
List<String>{'Revenue', 'desc'}};
String query =
Wave.QueryBuilder.load('datasetId','datasetVersionId').foreach(projs).order(orders).cap(10).build('q');
```

Output:

```
q = load "datasetId/datasetVersionId";
q = foreach q generate Name,AnnualRevenue as Revenue;
q = order q by (Name asc, Revenue desc);
q = limit q 10;
```

6. Query with a union statement.

```
Wave.ProjectionNode[] projs = new Wave.ProjectionNode[]{Wave.QueryBuilder.get('Name'),
Wave.QueryBuilder.get('AnnualRevenue').alias('Revenue')};
Wave.QueryNode nodeOne =
Wave.QueryBuilder.load('dataseOne','datasetVersionOne').foreach(projs);
Wave.QueryNode nodeTwo = Wave.QueryBuilder.load('datasetTwo',
'datasetVersionTwo').foreach(projs);
```

```
String query = Wave.QueryBuilder.union(new List<Wave.QueryNode>{nodeOne,
nodeTwo}).build('q');
```

Output:

```
qa = load "datasetOne/datasetVersionOne";
qa = foreach q generate Name,AnnualRevenue as Revenue;
qb = load "datasetTwo/datasetVersionTwo";
qb = foreach q generate Name,AnnualRevenue as Revenue;
q = union qa, qb;
```

7. Executing the query to get the result set via Query Builder.

```
Wave.ProjectionNode[] projs = new
Wave.ProjectionNode[]{Wave.QueryBuilder.count().alias('c')};
ConnectApi.LiteralJson result = Wave.QueryBuilder.load('datasetId',
'datasetVersionId').group().foreach(projs).execute('q');
```

8. Example of grouping by a specific dataset attribute.

```
Wave.ProjectionNode[] projs = new Wave.ProjectionNode[]{Wave.QueryBuilder.get('Name'),
Wave.QueryBuilder.get('Revenue').sum().alias('REVENUE_SUM')};
ConnectApi.LiteralJson result = Wave.QueryBuilder.load('datasetId',
'datasetVersionId').group(new String[]{'Name'}).foreach(projs).build('q');
```

Output:

```
q = load "datasetId/datasetVersionId";
q = group q by (Name);
q = foreach q generate Name,sum(Revenue) as REVENUE_SUM;
```

CRM Analytics Apex Templates

Use the `Templates` class to retrieve a collection of CRM Analytics templates, describe a single template and template configuration.

The CRM Analytics `Wave.Templates` class provides access from Apex to CRM Analytics templates. The methods are annotated with `@AuraEnabled` for use in Lightning Web Components (LWC).

`Wave.Templates` collection methods

`Map<String, Object> getTemplates()`

Retrieves a collection of templates.

`Map<String, Object> getTemplates(Wave.TemplatesSearchOptions searchOptions)`

Retrieves a collection of templates using search options.

Example: Apex class example

```
public with sharing class TemplatesController {
    public TemplatesController() {
    }

    @AuraEnabled(cachable=true)
    public static List<String, Object> getTemplateNames() {
      // This has filterGroup, type, and options; all fields are optional/nullable
```

```
      Wave.TemplatesSearchOptions options = new Wave.TemplatesSearchOptions();
      options.type = 'app';
      options.filterGroup = 'small';
      options.options = 'ViewOnly';

      // Pass null to get the default search options or leave it off completely to
return
      // a collection with no search options
      Map<String, Object> templates = Wave.Templates.getTemplates(options);

     // templates is the JSON response as an Apex Map (from JSON.deserializedUntyped),
 which
      // you can pull fields from
      List<Object> templateList = (List<Object>) templates.get('templates');
      List<String> names = new List<String>();
      for (Object templateObj : templateList) {
        names.add((String) ((Map<String, Object>) templateObj).get('name'));
      }
      return names;
    }
}
```

Example: LWC example

```
import {LightningElement, wire} from "lwc";
import getTemplates from "@salesforce/apex/Wave.Templates.getTemplates";

export default class Templates extends LightningElement {
  results;

  @wire(getTemplates, {
    options: {
      // All are optional
      type = 'app'
    }
  })

  // can also use these
  // @wire(getTemplates, { options: {} })
  // @wire(getTemplates, {})
  // @wire(getTemplates)
  // @wire(getTemplates, { options: {'$options'} }) // with a binding
  onTemplates({ data, error }) {
    if (error) {
      this.results = "Error:\n" + JSON.stringify(error, undefined, 2);
    } else if (data) {
      // data is the TemplateCollectionRepresentation JSON object
      this.results = 'Template names: ' + data.templates.map(l => ${l.name}).join(',
');
    } else {
       this.results = "No data";
    }
  }
}
```

`Wave.Templates` describe methods

`Map<String, Object> getTemplate(String templateIdOrApiName)`

Retrieves a template by ID or the API name.

`Map<String, Object> getTemplateConfig(String templateIdOrApiName)`

Retrieves a template configuration by ID or the API name.

Example: Apex class example

```
public with sharing class TemplateController {
    public TemplateController() {
    }

    @AuraEnabled(cacheable=true)
    public static Map<String, Object> getTemplate(String idOrName) {
      Map<String, Object> template = Wave.Templates.getTemplate(idOrName);
      return template;
    }
}
```

Example: LWC example

```
import {LightningElement, wire} from "lwc";
import getTemplate from "@salesforce/apex/Wave.Templates.getTemplate";

export default class Lens extends LightningElement {
    templateIdOrApiName; // set this to the ID or name you want to retrieve

    results;

    @wire(getTemplate, {
      templateIdOrApiName: '$templateIdOrApiName'
    })
    onTemplate({data, error}) {
      if (error) {
        this.results = "Error:\n" + JSON.stringify(error, undefined, 2);
      } else if (data) {
        // data is the TemplateRepresentation JSON object
        this.results = 'Template: ${data.name} ${data.id}';
      } else {
        this.results = "No data";
      }
    }
}
```

For information on request parameters and the Template JSON responses, see the Analytics REST API Developer Guide: Templates List Resource. To see the full reference for the Templates Apex class, see the Apex Reference Guide: wave Namespace.

ANALYTICS SDK RELEASE NOTES

Use the Salesforce Release Notes to learn about the most recent updates and changes to the Analytics SDK.

The Analytics SDK covers different tool sets, including Lightning Web Components, Aura components, Aura events, and Apex classes.

For a list of all current developer changes, see CRM Analytics in the Salesforce Release Notes.

Note: If the Analytics Development section in the Salesforce Release Notes isn't present, there aren't any updates for that release.