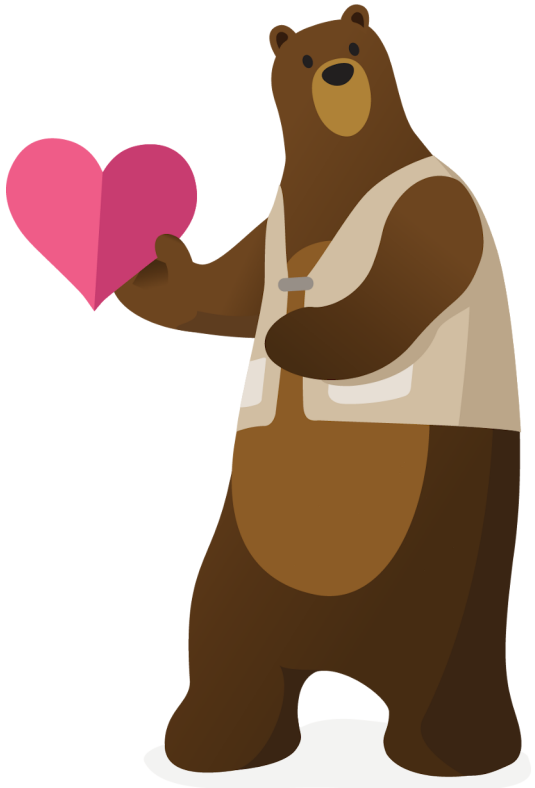
Analytics SDK Developer Guide

Salesforce, Spring '21



'21

@salesforcedocs

Last updated: January 5, 2021

CONTENTS

Tableau CRM SDK Developer Guide 1
Tableau CRM Assets SDK 2
Tableau CRM Template SDK 6
Tableau CRM Web SDK Events 9
Tableau CRM Apex SDK 18

TABLEAU CRM SDK DEVELOPER GUIDE

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

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

Tableau CRM Assets SDK

Are you ready to use the power of Tableau CRM directly inside Lightning Experience and create your own custom Tableau CRM functionality? By using the Tableau CRM Assets SDK, you can access the power of REST APIs to retrieve collections of Tableau CRM 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 Tableau CRM data to display runtime results.

Tableau CRM Templates SDK

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 Tableau CRM Template SDK, your application can do many of the same things from a Lightning Component controller.

Tableau CRM Web SDK Events

Would you like your application to communicate with your Tableau CRM 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 if 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 Tableau CRM Web SDK events we're making available are the foundation for a new way of thinking about Tableau CRM applications. Coupled with the Lightning Locker, you can even code your application outside of Salesforce—you can interact with Tableau CRM from any JavaScript application.

Tableau CRM Apex SDK

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 Tableau CRM directly from your Apex code? Say hello to the Tableau CRM Apex SDK and send well-formed SAQL queries to Tableau CRM.

This first phase of the Tableau CRM Apex SDK lets developers build 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 SDK evolves. The SDK also offers `Wave.InvalidParameterException` to help catch bad values supplied to the class methods.

Tableau CRM Assets SDK

Discover Tableau CRM 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

Tableau CRM Template SDK

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

Tableau CRM Web SDK Events

Easily interact with Tableau CRM in custom applications. Use Lightning Components for Visualforce, or your preferred development environment.

Tableau CRM Apex SDK

Query your data in Tableau CRM from any Apex class. Construct well-formed queries using the query builder. The Tableau CRM Apex SDK makes using Tableau CRM from Apex much easier.

Tableau CRM Assets SDK

Discover Tableau CRM 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 = {apiVersion: "47"};
   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

Gets 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
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 are not 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

Gets 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 are not supported.	FALSE	String

Parameter Name	Description	Required	Type
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

listDatasets

Gets 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 are not 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

Gets 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

Gets 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

Gets 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

Gets 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 Tableau CRM SAQL query.

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

listDashboardSavedViews

Gets 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

Gets 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

Gets 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

Tableau CRM Template SDK

Discover Tableau CRM templates and apps created from them. Create, update, and delete Tableau CRM 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 = {apiVersion: "44"};
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

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

getTemplate

Gets information for the specified template.

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

getTemplateConfig

Gets variable and UI information for the template.

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

createFolder

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

Parameter Name	Description	Type
name	Dev name of the folder.	String
label	Display label of the folder.	String
description	Description of the folder.	String
templateSourceId	Required; ID of the template used to create the folder.	String, in either the internal `'sfdc_internal__'` or base platform object ID format

Parameter Name	Description	Type
templateValues	Variable values for the template.	Object; map of name-value pairs, such as `{'Can_Continue' : true}`

listFolders

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

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

updateFolder

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

Parameter Name	Description	Type
folderId	ID of the folder.	String
label	New display label for the folder.	String
description	New description for the folder.	String
applicationStatus	When specified, cancels an app that is in progress.	`'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 template values that differ from the values used when the app was originally created.

Parameter Name	Description	Type
folderId	ID of the folder.	String
templateValues	Variable values for the template.	Object; map of name-value pairs, such as `{'Can_Continue' : true}`

deleteFolder

Deletes the specified folder.

Parameter Name	Description	Type
folderId	ID of the folder.	String

Tableau CRM Web SDK Events

Easily interact with Tableau CRM in custom applications. Use Lightning Components for Visualforce, or your preferred development environment.

Web SDK Events

wave:assetLoaded

A Tableau CRM asset fires this event when the asset is finished loading. The payload contains the asset type and the asset id. For a Tableau CRM 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 an Tableau CRM dashboard or interact with the dashboard by dynamically changing the selection. It has four attributes: the unique ID of the Tableau CRM asset on which to apply the filter, the payload, the asset type (currently only dashboard), and the fully qualified developer name of the Tableau CRM dashboard. The payload is a JSON string that identifies the datasets and any dimensions and field values.

wave:selectionChanged

An Tableau CRM 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 Tableau CRM 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 Tableau CRM 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 Tableau CRM 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 Tableau CRM dashboard

Note: To use these events, you need the Tableau CRM platform license Insights Builder PSL.

Tableau CRM Web SDK Events - Update Event
Create a custom component to dynamically set filters in an Tableau CRM dashboard embedded in a Lightning page.

Tableau CRM Web SDK Events - SelectionChanged Event
React to selections in your dashboard and get the row data for the selection.

Tableau CRM Web SDK Events - discover Event
This event sends a request to Tableau CRM dashboards to identify their assets.

Tableau CRM Web SDK Events - discoverResponse Event
This event provides the response following a request for Tableau CRM dashboards to identify their assets.

Tableau CRM Web SDK Events - Page Change Event
This event sends a request to Tableau CRM dashboards to change the displayed page.

Web SDK Events - Asset Loaded Event
React to the Tableau CRM asset rendering completion event.

Tableau CRM Web SDK Events - Update Event

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

Example - Setting a Filter with the Update Event

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

The Tableau CRM Web SDK events allow Tableau CRM to interact with the UI container. In this example, we create a custom component to interact with the embedded Tableau CRM 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, log in to Workbench, click **Utilities** | **REST Explorer**. In the text box, type *`/services/data/v47.0/wave/dashboards`* and then, click **Execute**. 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 the filter syntax Filter and Selection Syntax for Embedded Dashboards help topic.

```
({
  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 Tableau CRM Dashboard to send the filter to"/>
</design:component>
```

That's it. You can use your custom component to interact with Tableau CRM. 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 the new Web SDK events and other Lightning development features, see the Lightning Aura Components Developer Guide.

Tableau CRM Web SDK 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 Tableau CRM
 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 Tableau CRM 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 the new Web SDK events and other Lightning development features, see the Lightning Aura Components Developer Guide.

Tableau CRM Web SDK Events - discover Event

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

The `wave:discover` event sends a global request to listening Tableau CRM 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 the new Web SDK events and other Lightning development features, see the Lightning Aura Components Developer Guide.

Tableau CRM Web SDK Events - discoverResponse Event

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

Example - Setting Up Your Request and Receiving a Response

Refer to the *wave:discover* on page 14 event for details and an example using *wave:discoverResponse*.

Resources

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

Tableau CRM Web SDK Events - Page Change Event

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

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

Example - Updating the Displayed Dashboard Page

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

The Tableau CRM Web SDK events allow Tableau CRM to interact with the UI container. In this example, we create a custom component to interact with the embedded Tableau CRM 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="XXXXXXXXXXXXXXX"/>
    <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, log in to Workbench, click **Utilities** | **REST Explorer**. In the text box, type `/services/data/v47.0/wave/dashboards` and then, click **Execute**. 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 Tableau CRM 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 the new Web SDK events and other Lightning development features, see the Lightning Aura Components Developer Guide.

Web SDK Events - Asset Loaded Event

React to the Tableau CRM asset rendering completion event.

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

For this example, the Analytic 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 Tableau CRM dashboard in a Lightning page (see the Embed Tableau CRM 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 Tableau CRM 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 the new Web SDK events and other Lightning development features, see the Lightning Aura Components Developer Guide.

Tableau CRM Apex SDK

Query your data in Tableau CRM from any Apex class. Construct well-formed queries using the query builder. The Tableau CRM Apex SDK makes using Tableau CRM from Apex much easier.

The Apex SDK currently consists of the executeQuery method and the 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 Tableau CRM, and get a response in the form of JSON. For example, this sample sends 'your SAQL query' to Tableau CRM.

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

Sending queries like this is very 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

Exposed through the Wave namespace, the 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 need to 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 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;
```

Resources

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

Tableau CRM Apex SDK QueryBuilder Examples
Build simple or complex SAQL queries using QueryBuilder.

Tableau CRM Apex SDK QueryBuilder Examples

Build simple or complex SAQL queries using QueryBuilder.

QueryBuilder is the core of this first phase of the Tableau CRM Apex SDK, 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;
```