# TOC

# Composer 24

## Embed Composer Dashboards Into Applications Using Composer

# Content Security Policy Support for Embedded Pages

Pages that include embedded Composer content can include content security policies with some restrictions. Some guidelines for creating these policies include:

- `script-src` Allow the domain used to serve Composer. `unsafe-inline` or a `nonce` is required. `unsafe-eval` is not required.
- `style-src` Allow the domain used to serve Composer. `unsafe-inline` is required; `nonce` is not yet supported. `unsafe-eval` is not required.
- `font-src` Allow the domain used to serve Composer. Allow `data:` for embedded fonts.

# Using Nonce

If you want to omit `unsafe-inline` for `script-src`, you must use a nonce in the page. The nonce should be a unique string that changes on each new page load.

To instruct the page that the nonce is allowed, include it in the `script-src` portion of your content security policy definition as `nonce-<random string>`. Next, add the same string as the value of the `nonce` attribute on the script tag you use to import the Composer embed, as well as any scripts that call those embed functions.

When this is included, the embed system will include the nonce on any inline script tags it creates.

Example: `index.html`

```
html
<head>
	<meta
		http-equiv="Content-Security-Policy"
		content="
			font-src 'self' data:;
			style-src 'self' https://localhost:8080 'unsafe-inline';
			script-src 'self' https://localhost:8080 'nonce-abc123';
		"
	/>
</head>
<body>
	<script
		data-name="composer-embed-manager"
		src="https://localhost:8080/composer/embed/embed.js"
		nonce="abc123"
	></script>
	<script
```

```
            data-name="main-application-script"
            src="https://localhost/myApp/main.js"
            nonce="abc123"
        ></script>
</body>
```

# Embed Components Into Your Application

You can embed Composer components into your own applications.

The following components can be embedded:

- dashboards, lite dashboards
- visual authoring experience
- source editor

When components are embedded, the way in which users can interact with them is determined by settings established in their Composer user account. See Embedded Composer Component Controls.

**Note:** insightsoftware recommends using Trusted Access for all embed-related workflows.

The list of options available on an embedded visual's drop-down menu depends on:

- The mode setting for the dashboard. If the embed mode is `readonly` or **Read Only**, no menu is available at all.
- The visual interactivity settings specified in the visual definition.
- The dashboard interactivity settings specified for the dashboard, if those settings include override interactivity settings for all visuals in the dashboard.

When options are shown for a visual in an embedded dashboard, they may be shown in the visual sidebar menu or within the visual drop-down menu itself, depending on the setting of the `editor.placement` property.

- If `editor.placement` is set to `dockRight`, the options appear in sidebars using the visual sidebar menu and the resulting sidebar editing panels.

  If `editor.placement` is set to `modals`, the options appear in the visual drop-down menu itself and the resulting floating dialogs.

**Note:** In environments where you use Typescript for your client side code, you can use Embed Manager as an npm package. See https://www.npmjs.com/package/logi-embed.

See the following topics for embedding components using JavaScript:

- Embedded Component Prerequisites
- Token-Related Configuration Properties
- Embed Composer Components Using JavaScript And Trusted Access
- EmbedManager Methods
- Embedded Dashboard Properties And Objects
- Embedded Visual Authoring Properties And Objects
- Embedded Events
- Interactivity Properties
- Embedded Composer Component Controls

See the following topics for embedding a dashboard using an embeddable dashboard snippet:

- Embedded Component Prerequisites
- Generate An Embeddable Dashboard HTML Snippet
- Generate A Visual Gallery HTML Snippet
- Embed A Dashboard Using A Generated Snippet And Trusted Access Tokens
- Embedded Composer Component Controls
- Embed Multiple Composer Dashboards On A Single Page

See the following topics for embedding the source editor:

- Embedded Source Editor
- Embed Source Editor Properties

# Embedded Source Editor

Use Composer to embed a source editor for your users, enabling them to view, edit, or configure data sources directly in your application. Define the editor for users as needed for your custom solution.

You can embed the complete source editor, or limit what your users can access and use.



# Step 1. Define a Connection to Create a Source

Define a connection users can access to create a source. See Embed Source Editor Properties for more information on the properties for creating a source.

```
{
        create: {
                visible: true,
                connections: {
                        ids: ["6347eca23d231b5c3c2a2f79", "633d6ef33d27115c3c2a2e45"],
                        defaultId: "6347eca22d271b5c3c2a2f79"
                }
        },
        fields: {
                visible: true
        },
        caching: {
                visible: true
        },
        settings: {
                visible: true
        }
}
```

# Step 2. Create a Component for the Source Editor

```
// First initialize embed manager
const sourceEditor = await embedManager.createComponent('source-editor', {
       sourceId: "633167d188f857721d2f70dd", // existing Source ID
       activeTab: "fields",
       theme: "composer",
       tabs: { ... },
       breadcrumbs: { ... },
       interactivityOverrides: { ... },
       notificationSettings: { ... },
});
```

# Step 3. Define Interactivity Settings

```
interactivityOverrides: {
       create: {
              ADD_FROM_CONNECTION: false,
              UPLOAD_NEW_FILE: 'false' // value can be string as well
       },
       fields: {
              ADD_DERIVED_FIELD: false,
              ADD_HIERARCHY_FIELD: 'false' // value can be string as well
       },
       caching: {
              STATISTICS_CACHE: false,
              SCHEDULE_REFRESH: 'false' // value can be string as well
       }
}
```

# Step 4. Define Breadcrumbs Configuration

```
{
       title: "Back to Sources",
       href: "https://company.com/sources",
       target: "_blank",
       onClick: () => {
```

```
                console.log("do action");
        }
}
```

# Embed Source Editor Properties

Use the following properties when creating an embedded source editor.

## Top Level Properties

| Property | Default Value | Description |
|---|---|---|
| `sourceId` | `undefined` | The ID of the source for the editor to connect to. If an ID is not passed, an empty editor is embedded.<br>Type: string |
| `activeTab` | `'create'` | The Active Tab presented to the user when the editor is loaded. Options include `'create'`, `'fields'`, `'caching'`, and `'settings'`. |
| `theme` | `undefined` | The theme to use for the source editor. If a theme name is not passed, the default theme is used. Options include `modern`, `dark`, and `<custom>`.<br>Type: string |
| `header` | | Type: HeaderProps<br>See Header Properties. |
| `breadcrumbs` | | Type: BreadcrumbsProps<br>See Breadcrumb Properties. |
| `tabs` | | Type: TabsProps<br>See Tab Properties. |
| `interactivityOverrides` | | Type: InteractivityOverrides<br>See Interactivity Overrides. |
| `notificationSettings` | `{ enabled: true, position: 'top-right' }` | Use to show or hide the notifications and define their position.<br>Type: NotificatonSettings |

## Header Properties

| Property | Default Value | Description |
|---|---|---|
| `visible` | true | Defines the visibility of the header. |

| Property | Default Value | Description |
|---|---|---|
| | | Type: boolean |
| showTitle | true | Defines the visibility of the source name.<br>Type: boolean |

# Breadcrumb Properties

| Property | Default Value | Description |
|---|---|---|
| title | "Sources" | The title of the first item in the breadcrumbs path.<br>Type: string |
| href | "" | The link address for the first item in the breadcrumbs path.<br>Type: string |
| target | "" | The link target parameter. Use "_blank" to open the editor in a new tab.<br>Type: string |
| onClick | undefined | The action handler for selection of the breadcrumb.<br>Type: string |

# Tab Properties

| Property | Default Value | Description |
|---|---|---|
| create.visible | true | Define visibility of the Source Creation tab.<br>Type: boolean |
| create.connections | undefined | Configure the Connections list as a data entity.<br>Type: ConnectionsProps |
| fields.visible | true | Define visibility of the Fields tab.<br>Type: boolean |
| caching.visible | true | Define visibility of the Cache tab.<br>Type: boolean |

| Property | Default Value | Description |
|---|---|---|
| `settings.visible` | `true` | Define visibility of the Global Settings tab.<br>Type: boolean |

# Connection Properties

| Property | Default Value | Description |
|---|---|---|
| `ids` | `undefined` | The Connection IDs to use in the source editor for Data Entity creation.<br>■ If no values are defined, all connections are visible.<br>■ If the value is an empty array, users can only connect to the connection defined in `defaultId`.<br>■ If a selected Data Entity is not in the list, the connection for that entity is added to the list of available connections for the data entity.<br>Type: string array |
| `defaultId` | `undefined` | Define the default connection ID. If this value is not defined, no default connection selection is available.<br>Type: string |

# Interactivity Overrides

| Property | Default Value | Description |
|---|---|---|
| `create` | `undefined` | The overrides for the interactivity settings related to the Source Creation tab.<br>Type: CreationInteractivitySettings |
| `fields` | `undefined` | The overrides for the interactivity settings related to the Fields tab.<br>Type: FieldsInteractivitySettings |
| `caching` | `undefined` | The overrides for the interactivity settings related to the Fields tab.<br>Type: CachingInteractivitySettings |

# Creation Interactivity Settings

| Property | Description |
|---|---|
| `"ADD_FROM_CONNECTION" : true` | When set to `true`, users can see and use the **Add From Connection** menu item on the Source Creation tab. When set to `false`, it is hidden.<br>Type: boolean |
| `"ADD_FROM_FILE_UPLOAD" : true` | When set to `true`, users can see and use the **Add From File** menu item on the Source Creation tab. When set to `false`, it is hidden.<br>Type: boolean |
| `"SELECT_FILE_UPLOAD" : true` | When set to `true`, users can see and use the **Select File** control on the Source Creation tab. When set to `false`, it is hidden.<br>Type: boolean |
| `"UPLOAD_NEW_FILE" : true` | When set to `true`, users can see and use the **Upload New File** button on the Source Creation tab. When set to `false`, it is hidden.<br>Type: boolean |
| `"CREATE_JOINS" : true` | When set to `true`, users can see and use the **Add** control. When set to `false`, it is hidden.<br>Type: boolean |
| `"FILTER_VALUES_ENTITIES" : true` | When set to `true`, users can see and use the **Filter Values Entity** option on the Source Creation tab. When set to `false`, it is hidden.<br>Type: boolean |

# Fields Interactivity Settings

| Property | Default Value | Description |
|---|---|---|
| `ADD_DERIVED_FIELD` | `true` | Defines the visibility of the Add Derived Field button on the Fields tab.<br>Type: boolean |
| `ADD_HIERARCHY_FIELD` | `true` | Defines the visibility of the Add Hierarchy Field button on the Fields tab.<br>Type: boolean |
| `ADD_CUSTOM_METRIC` | `true` | Defines the visibility of the Add Custom Metric button on the Fields tab. |

| Property | Default Value | Description |
|---|---|---|
| | | Type: boolean |
| VISIBILITY | true | Defines the visibility of the Visibility column on the Fields tab.<br>Type: boolean |
| SETTINGS | true | Defines the visibility of the Settings menu on the Fields tab.<br>Type: boolean |
| FILTER_VALUES | true | Defines the visibility of the Filter Values menu on the Fields tab.<br>Type: boolean |
| INFO | true | Defines the visibility of the Info menu on the Fields tab.<br>Type: boolean |
| DELETE | true | Defines the visibility of the Delete button on the Fields tab.<br>Type: boolean |
| FILTER_VALUE_OVERRIDES | true | Defines the visibility of the Filter Values tab on the Fields tab.<br>Type: boolean |
| FIELD_CAPABILITIES | true | Defines the visibility of the Field Capabilities button on the Fields tab.<br>Type: boolean |
| MANAGE_TRANSLATIONS | true | Defines the visibility of the Upload Translation File button on the Fields tab.<br>Type: boolean |

# Caching Interactivity Settings

| Property | Default Value | Description |
|---|---|---|
| DATA_CACHE | true | Defines the visibility of the Data Cache control on the Cache tab.<br>Type: boolean |
| STATISTICS_CACHE | true | Defines the visibility of the Statistics Cache control on the Cache tab.<br>Type: boolean |
| SCHEDULE_REFRESH | true | Defines the visibility of the Schedule Refresh control on the Cache tab.<br>Type: boolean |

| Property | Default Value | Description |
| --- | --- | --- |
| STATISTICS_CACHE | true | Defines the visibility of the Statistics Cache control on the Cache tab.<br>Type: boolean |

# Embedded Dashboard Cross-Visual Publish and Subscribe Example

The following JavaScript example uses methods, properties, and embedded events of the `Zoomdata` and `EmbedManager` classes to publish and subscribe cross-visual links and filters in an embedded dashboard.

```
// EmbedManager is a singleton and `initComposerEmbedManager` can create a new
// manager or return an existing one. See other samples for howto properly
// initialize the embed manager.
const getEmbedManager = async () => window.initComposerEmbedManager();

// An embedding application should listen for the 'composer-dashboard-loaded'
// event before attempting to publish or subscribe. If multiple dashboards are
// added to the page, listeners can be added the object returned upon embedding
// to specifically target an individual dashboard
const DASHBOARD_LOADED_EVENT = 'composer-dashboard-loaded';
const pubSubReady = new Promise((resolve) => {
       function resolvePubSubReady() {
              resolve();
              document.removeEventListener(DASHBOARD_LOADED_EVENT, resolvePubSubReady);
       }
       document.addEventListener(DASHBOARD_LOADED_EVENT, resolvePubSubReady);
});

// `countrySelect` represents a drop-down field on the embedding application's
// page that lists a set of countries the user can filter by. After selecting a
// value, the embedding application calls its 'publishValue' function, defined
// below with the link name 'country' and the selected country value. The
// link name is defined in the dashboard. To clear the value filter, make sure
// you pass a null value as shown below.
const countrySelect = document.getElementById('country-select');
countrySelect.addEventListener('change', (event) => {
       const value = event.target.value;
       publishValue('country', value);
});

// The sample function publishValue below takes a link name and single string
// value and publishes the value on the link name. This causes all visuals on
// any loaded dashboards that are subscribed to the given link to filter
// themselves by the selected value. A visual can be set to subscribe to an
// arbitrary link name by adding a cross-source link using that link name and
// assigning any relevant field to that link.

// PUBLISHER_ID is an arbitrary string that subscribers can use to associate a
// message with who sent it.
```

```
const PUBLISHER_ID = 'Embedding Application';
async function publishValue(linkName, value) {
	const embedManager = await getEmbedManager();
	const message = {
		type: 'selection',
		valueType: 'ATTRIBUTE',
		ranges: [
			{
				operation: 'IN',
				value: value
			}
		]
	};
	const options = {
		publisherId: 'PUBLISHER_ID'
	};

	await pubSubReady;
	embedManager.publish(
		linkName,
		message,
		options
	);
}

// This sample function reates a subscription to the linkname provided using the
// handler provided. The returned unsubscribe function should be stored and
// called when the subscriber should stop receiving messages.
async function subscribe(linkname, handler) {
	const embedManager = await getEmbedManager();
	await pubSubReady;
	return embedManager.subscribe(linkname, handler);
}

// The following code provides an example of how to subscribe to a linkname on
// page load and receive the latest value published. It creates a simple alert
// popup on receipt of each message.
function subscriptionHandler(message, publisherId) {
	if (message === null) {
		alert(`${publisherId} cleared its publication`);
	} else {
		const value = message.ranges[0].value;
		alert(`${publisherId} published ${value}`);
	}
```

```
}
const countryUnsubscribe = subscribe('country', subscriptionHandler);
```

# Embed Multiple Composer Dashboards On a Single Page

You can embed multiple Composer dashboards on a single page of your application and in a single `<div>` in your application, including embedding one or more empty dashboards. You can also embed the same dashboard multiple times on a page and each embedded instance can use its own property settings (theme, mode, header, title). Finally, the dashboards can be embedded using different methods: you can embed one dashboard using generated embed code and another dashboard using JavaScript.

**Note:** When more than one dashboard is embedded in application, the cross-visual filters that are published for a visual on one of the dashboards can be subscribed to by any visual on any of the embedded dashboards. However, this is not true unless the dashboards are embedded in the same application. Visuals in a dashboard open in one window or tab cannot subscribe to cross-visual filters published by visuals in a different window or tab.

# Embedded Library Events

Events can be used in your JavaScript to control an embedded Composer component when specific events occur.

**Note:** The ability for your end-users to perform some of the events listed here is controlled by the permissions granted to them with their Composer credentials. See Embedded Composer Component Controls.

You can subscribe using `.addEventListener()` to embedded inventory component events so that you can execute your own logic when an event occurs.

**Note:** Composer provides a `componentInstanceId` provides a as part of event details for dashboard, source, and visual events.

The following events are supported:

| Event | Data Passed | Example |
|---|---|---|
| composer-inventory-ready | `undefined` | `inventory.addEventListener("composer-inventory-ready", (e) => { console.log(e); });` |
| composer-inventory-loaded | Inventory items `e.detail.inventoryItems` | `inventory.addEventListener("composer-inventory-loaded", (e) => { console.log(e.detail.inventoryItems); );` |
| composer-inventory-failed | Failed reason `e.detail.failedReason` | `inventory.addEventListener("composer-inventory-failed", (e) => { console.log(e.detail.failedReason); });` |
| composer-inventory-item-deleted | Inventory item data `e.detail.inventoryItem` | `inventory.addEventListener("composer-inventory-item-deleted", (e) => { console.log(e.detail.inventoryItem); }};` |

# Embedded Component Prerequisites

To embed a Composer component into your own application, the following prerequisites must be met:

- You must have created a host application into which you want to embed the Composer component.
- Trusted Access, enabled by default, must be enabled and configured to authenticate users of your embedded components. See Trusted Access. Be sure that your application is registered as a Composer client and that you have met the prerequisites of Trusted Access.

  **Note:** insightsoftware recommends using Trusted Access for all embed-related workflows.

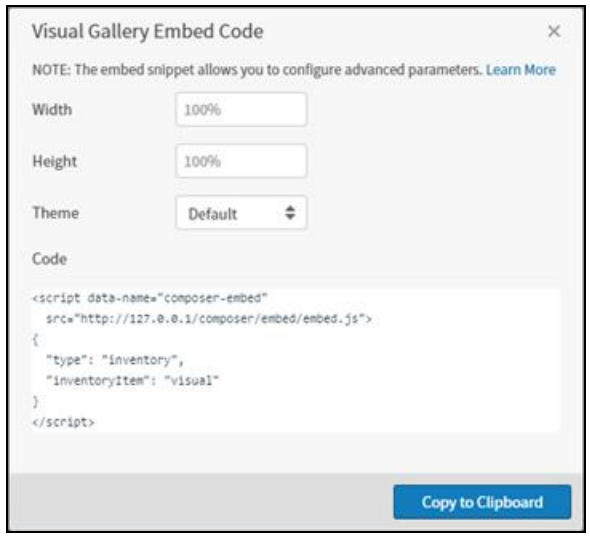
- Cross-origin sharing (CORS) must be enabled for your Composer instance. See Enable Composer Visual Data Discovery Component Access From Other Sites Using Cross-Origin Resource Sharing (CORS).
- You will need to be a Composer administrator or a user assigned to a group with the **Generate Embed Code** privilege so you can generate library, visual gallery, or sources inventory snippets used for embedding.

# Generate a Visual Gallery HTML Snippet

You can generate an embeddable HTML snippet for the visual gallery using the UI.

**Generate an embeddable HTML snippet**

1. Log into the UI as an administrator or as a user assigned to a group with the **Generate Embed Code** privilege.
2. Select **Visual Gallery** on the top-level navigation banner or select **Visual Gallery** on the Home page. The visual gallery appears.
3. Select the embed library icon ( ) above the list of visuals. The Embed Code dialog appears.




The Code section of this dialog shows the embeddable snippet. If you do not need to change any of the default settings on this page, simply select **Copy to Clipboard** and you can skip the rest of these steps and embed the copied snippet in your application.

If, however, you want to alter the default settings on this dialog, continue with the rest of these steps. Note as you change settings that the embeddable snippet is updated automatically. All settings are optional.

4. The default width setting (100%) is shown in the **Width** box. Select the entry field, then enter the width value you want in CSS units. For example, `800px`, `75%`, `500em` and `80vw` are all valid settings.
5. The default height setting (100%) is shown in the **Height** box. Select the entry field, then enter the height value you want in CSS units. For example, `800px`, `75%`, `500em` and `80vh` are all valid settings.

6. Select a theme in the **Theme** box. By default, three possible themes are available: **Logi-Composer**, **Logi-Modern**, and **Logi-Dark**. However, if you add your own themes to the application, more options are available in this list. The default is **Logi-Composer**, which is the same as the **Logi-Modern** theme. For information on adding your own UI themes, see Manage UI Themes.

7. When all the optional settings are defined as you need, select **Copy to Clipboard** to copy the HTML snippet to the clipboard. You can then paste the this snippet into your application code.

8. Close the Embed Code dialog by selecting the **x** in the upper right corner of the dialog.

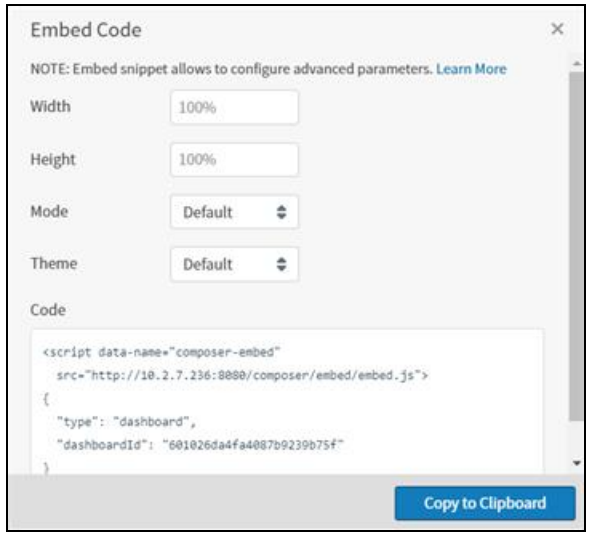
# Generate an Embeddable Dashboard HTML Snippet

You can generate an embeddable HTML snippet for a dashboard using the UI.

For even more control, use JavaScript to embed the dashboard. See Embed Composer Components Using JavaScript And Trusted Access.

**Generate an embeddable HTML snippet for a dashboard**

1. Log into the UI as an administrator or as a user assigned to a group with the **Generate Embed Code** privilege.
2. Select **Library** on the top-level navigation banner or select **Dashboard** on the Home page. The library opens, displaying dashboards in a table (list) format.
3. Locate the dashboard in the library list for which you want to generate an embeddable snippet.
4. Select  in the associated **Actions** column. The Embed Code dialog appears.




The **Code** section of this dialog shows the embeddable snippet. If you do not need to change any of the default settings on this page, simply select **Copy to Clipboard** and you can skip the rest of these steps and embed the copied snippet in your application.

If, however, you want to alter the default settings on this dialog, continue with the rest of these steps. Note as you change settings that the embeddable snippet is updated automatically. All settings are optional.

5. The default width setting (100%) is shown in the **Width** box. Click in the box and enter the width value you want in CSS units. For example, `800px`, `75%`, `500em` and `80vw` are all valid settings.

6. The default height setting (100%) is shown in the **Height** box. Click in the box and enter the height value you want in CSS units. For example, `800px`, `75%`, `500em` and `80vh` are all valid settings.

7. Select a mode in the **Mode** box. The mode setting determines the way in which your users will be able to work with the embedded dashboard. If you do not want the user in your application to change anything and only be able to view the dashboard, select **Read Only**. If you want your users to be able to make changes to the dashboard, select **Interactive**. The default is **Interactive**.

   When the mode is **Read Only**, the dashboard cannot be changed.

   > **Note:** The level of interactivity a user has with an embedded dashboard is determined by the interactivity settings of each visual in the dashboard. See Control How Users Interact With A Visual.

8. Select a theme in the **Theme** box. By default, three possible themes are available: **Logi-Composer**, **Logi-Modern**, and **Logi-Dark**. However, if you add your own themes to the application, more options are available in this list. The default is **Logi-Composer**, which is the same as the **Logi-Modern** theme. For information on adding your own UI themes, see Manage UI Themes.

9. When all the optional settings are specified as you need, select **Copy to Clipboard** to copy the embeddable dashboard snippet to the clipboard. You can then paste the embeddable HTML snippet into your application code.

10. Close the Embed Code dialog by selecting the **x** in the upper right corner of the dialog.

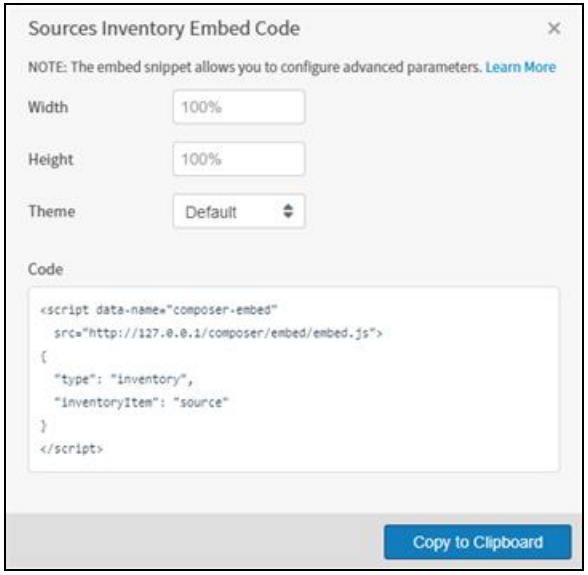
11. If you want to specify additional properties for your embedded dashboard, use Javascript. The supported dashboard properties are described in Embedded Dashboard Properties And Objects.

# Generate a Sources Inventory HTML Snippet

You can generate an embeddable HTML snippet for the sources inventory using the UI.

**Generate an embeddable HTML snippet**

1. Log into the UI as an administrator or as a user assigned to a group with the **Generate Embed Code** privilege.
2. Select **Sources** on the top-level navigation banner or select **Sources** on the Home page. The Sources page appears.
3. Select the embed sources inventory icon ( ) above the list of sources. The Embed Code dialog appears.




The Code section of this dialog shows the embeddable snippet. If you do not need to change any of the default settings on this page, simply select **Copy to Clipboard** and you can skip the rest of these steps and embed the copied snippet in your application.

If, however, you want to alter the default settings on this dialog, continue with the rest of these steps. Note as you change settings that the embeddable snippet is updated automatically. All settings are optional.

4. The default width setting (100%) is shown in the **Width** box. Select the entry field, then enter the width value you want in CSS units. For example, `800px`, `75%`, `500em` and `80vw` are all valid settings.

5. The default height setting (100%) is shown in the **Height** box. Select the entry field, then enter the height value you want in CSS units. For example, `800px`, `75%`, `500em` and `80vh` are all valid settings.

6. Select a theme in the **Theme** box. By default, three possible themes are available: **Logi-Composer**, **Logi-Modern**, and **Logi-Dark**. However, if you add your own themes to the application, more options are available in this list. The default is **Logi-Composer**, which is the same as the **Logi-Modern** theme. For information on adding your own UI themes, see Manage UI Themes.

7. When all the optional settings are defined as you need, select **Copy to Clipboard** to copy the HTML snippet to the clipboard. You can then paste the this snippet into your application code.

8. Close the Embed Code dialog by selecting the **x** in the upper right corner of the dialog.

# Embed a Dashboard Using a Generated Snippet and Trusted Access Tokens

**Note:** insightsoftware recommends using Trusted Access for all embed-related workflows.

**Embed a dashboard into your application using Trusted Access Tokens**

1. Verify that the embedded dashboard prerequisites have been met. See Embedded Component Prerequisites.
2. Use the Composer API to register your host application as a Trusted Access client of your Composer instance and obtain a client ID and client secret. See Trusted Access - Register a Client.
3. Add a `window.composerGetToken` function to your host application to obtain an access token and expiration for a user session when a user uses the embedded dashboard.

   The `composerGetToken` function must communicate with a server you have created (see Step 4) that takes information about the active user session (the user name) and contacts Composer to generate an access token for the user session. This is an `async` function that returns a promise. For example:

   ```
   window.composerGetToken = async function composerGetToken() {
          return { access_token: "<access_token>", expires_in: <seconds> };
   };
   ```

   Where `<access-token>` is the Trusted Access token you obtain from your server and `<seconds>` is the number of seconds until the token expires.
4. Implement your own server-side code that accepts requests generated by the `window.composerGetToken` function and, in turn, makes requests to Composer to generate a token for the specific user session of a user using the embedded dashboard. Your server-side code should respond to these requests with a token and expiration information.

   To generate an access token for the user, call the Trusted Access push/tokens API to generate tokens for new Composer users, or the pull/tokens API to retrieve tokens for existing users. See Trusted Access API Endpoints and Trusted Access
5. Generate and copy the embeddable snippet for your dashboard. See Generate An Embeddable Dashboard HTML Snippet.
6. Insert the generated snippet into your host application using the appropriate `script` tag, configured to show the embedded dashboard in the appropriate HTML element.

# Embed Composer Components Using JavaScript and Trusted Access

You can embed a Composer component using JavaScript.

The following components can be embedded:

- dashboards, lite dashboards
- visual authoring experience
- source editor

When components are embedded, the way in which users can interact with them is determined by settings established in their Composer user account. See Embedded Composer Component Controls.

Complete examples of JavaScript code that initializes and authorizes the Composer `EmbedManager` class are provided in Embedded JavaScript Examples.

Follow these steps:

- Step 1. Verify The Prerequisites Have Been Met
- Step 2. Make The Embed Manager Available To Your Application
- Step 3. Initialize And Authorize The Embed Manager
- Step 4. Create And Render A Composer Component In Your Application

## Step 1. Verify the Prerequisites Have Been Met

Verify that the embedded dashboard prerequisites have been met. See Embedded Component Prerequisites.

## Step 2. Make the Embed Manager Available to Your Application

To make the Composer's embed manager available to your application, include this script in your HTML:

```
<script data-name="composer-embed-manager" src="https://<sampleurl>/embed/embed.js"></script>
```

where `<samplecomposerurl>` is the URL of your Composer instance.

After this script is run, the `initComposerEmbedManager` function is available globally in `window`.

# Step 3. Initialize and Authorize the Embed Manager

Use the `window.initComposerEmbedManager` function to initialize and authorize the embed manager. This can be done using the following configuration properties. Note that the token you supply must be obtained beforehand using the Trusted Access API, usually in backend code that supports your HTML.

The `getToken` property is a method that returns a token from Trusted Access prior to token expiration. Here is an example:

```
const getEmbedManagerPromise =
window.initComposerEmbedManager({
       getToken: function () {
           // transform for the embed syntax
           return getToken().then((result) => { // getToken function uses the Trusted Access API
               return {
                   access_token: result.token,
                   expires_in: result.expiresIn,
               };
            });
        },
});
```

After the `initComposerEmbedManager` function is called, it returns a promise that resolves an instance of the `EmbedManager` class. The objects, methods, properties, and embedded events provided with the Composer `EmbedManager` class can be used in your HTML.

# Step 4. Create and Render a Composer Component in Your Application

After the Composer `EmbedManager` class has been initialized and authorized, you can use its methods to embed a Composer component in your application and set various properties for that component. In addition, you can use Composer embedded events to control component behavior when specific events occur.

Other `EmbedManager` methods can be used to modify, refresh, and remove Composer components in your application.

Complete descriptions of the supported `EmbedManager` methods and properties are provided in EmbedManager Methods, Embedded Dashboard Properties And Objects, and Embedded Visual Authoring Properties And Objects. Supported Composer embedded events are described in Embedded Events.

The following simple example creates and renders an embedded dashboard:

```
getEmbedManagerPromise.then((embedManager) => {
     embedManager.createComponent('dashboard',
          {
               dashboardId: <id>, // dashboard id
               <property>,... // set of properties
```

```
          }
     ).then(dashboard => dashboard.render(
          htmlElement, // htmlElement
          {width: '100%', height: '100%'}
     ))
});

// Example with async await
async function createComponent() {
     const embedManager = await getEmbedManagerPromise;
     const newDashboard = await embedManager.createComponent('dashboard',
          {
               dashboardId: <id>, // dashboard id
               <property>,... // set of properties
          });
     newDashboard.render(
          htmlElement, // htmlElement or selector
          {width: '100%', height: '100%'}
     ));
}
```

The following simple example creates and renders an embedded lite dashboard:

```
embedManager.createComponent('lite-dashboard',
       const componentConfig = {
               "dashboardId": "<dashboard-ID>",
               "componentInstanceId":"<component-instance-ID>",
               "type":"lite-dashboard",
               "application":{
                       "banner":false,
                       "logo":true
               },
               "interactivityProfileName":"lite",
               "theme":"modern",
               "editor":{
                       "placement": "docRight"
               },
               "header": {
                       "title": "My Lite Dashboard"
                       "showActions": false,
                       "showTitle": false,
                       "visible": false
               },
```

```
                "availableVisTypes" : [`HISTOGRAM`,`BOX_PLOT`,`HEATMAP`]
        }
);
```

# Token-Related Configuration Properties

The `initComposerEmbedManager` window function supports the following token-related configuration properties.

| Property | Description |
|---|---|
| `getToken` | This method returns a token using Trusted Access. This is the same as the the `window.composerGetToken` method. There is no default.<br><br>`getToken: () => {`<br>`  return Promise.resolve({ // this is an example function. server side should be called in production`<br>`  "access_token": "60a26b4f-c5d7-49af-a90b-1da4e6eb12b8",`<br>`  "expires_in": "6000"`<br>`});`<br>`}` |

# EmbedManager Methods

The following `EmbedManager` methods can be used to manage embedded Composer components. Sample code showing the use of these methods in an application are provided. See Embedded JavaScript Examples for a complete example of how these methods might be coded in JavaScript.

| Method | Description |
|---|---|
| `createComponent()` | Creates a Composer component in your application. This method is asynchronous and returns a promise that resolves the created component.<br><br>The parameters input to this method include:<br><br>■ a string that identifies the type of component you want created (the component types currently supported are `dashboard` and `visual-builder`). An error occurs if an invalid component type is specified.<br><br>■ the configuration of the component, which includes the properties described in Embedded Dashboard Properties And Objects and Embedded Visual Authoring Properties And Objects.<br><br>Use a standard `render` method (for example, `dashboard.render` or `visualBuilder.render`) to render the Composer component in your application after it has been created. Examples are provided in Embedded JavaScript Examples.<br><br>The following example embeds a dashboard:<br><br>`const dashboard = await embedManager.createComponent('dashboard', {`<br>`dashboardId: <id>,`<br>`theme: 'composer',`<br>`interactivityProfileName: '<interactivity-profile-name>',`<br>`interactivity {`<br>`settings:{`<br>`CHANGE_LAYOUT: "true",`<br>`},`<br>`visualSettings:{`<br>`FILTER: "false",`<br>`overrideVisualInteractivity: "true",`<br>`},`<br>`}`<br>`editor: {`<br>`placement: 'dockRight' // use eve sidepanel`<br>`},`<br>`header: {`<br>`visible: true,`<br>`showTitle: true,` |

| Method | Description |
|---|---|
| | <pre>showActions: false, // will hide actions bar<br>title: "Static custom title"<br>}<br>});</pre> |
| `getComponentById()` | Returns an instance of a Composer component, based on its component instance ID. The parameter input to this method is a string representing the component instance ID.<br><pre>async function getComponent(<componentInstanceId>) {<br>const embedManager = await createOrGetEmbedManager();<br>return embedManager.getComponentById(<componentInstanceId>);<br>}</pre> |
| `refresh()` | Refreshes all embedded components. This is useful when the token is expired. No parameters are passed to this method.<br><pre>async function refreshComponent(<newToken>) {<br>const embedManager = await createOrGetEmbedManager();<br>embedManager.updateToken(<newToken>).then(() => {<br>embedManager.refresh();<br>});<br>}</pre> |
| `refreshComponent()` | Refreshes a particular component. The parameter used with this method is a string representing the component instance ID.<br>The following example refreshes a specific dashboard:<br><pre>async function refreshDashboard(<newToken>, <componentInstanceId>) {<br>const embedManager = await createOrGetEmbedManager();<br>embedManager.updateToken(<newToken>).then(() => {<br>embedManager.refreshComponent(<componentInstanceId>);<br>});<br>}</pre> |
| `refreshWithToken()` | Combines an update token and refresh request for all components. The parameter input to this method is a string representing the new token. |

| Method | Description |
| --- | --- |
| | ```<br>async function refreshComponent(<newToken>) {<br>  const embedManager = await createOrGetEmbedManager();<br>  embedManager.refreshWithToken(<newToken>);<br>}<br>``` |
| `removeComponent()` | Removes an embedded Composer component from your application. This method returns the value `true` when the removal is successful and `false` when it fails. When a component is removed, its watchers are also removed.<br><br>The parameter input to this method is a string representing the component instance ID.<br><br>```<br>async function clearComponent(id) {<br>  const embedManager = await createOrGetEmbedManager();<br>  embedManager.removeComponent(<componentInstanceId>);<br>}<br>``` |
| `updateToken()` | Refreshes the authorization token for an embedded Composer component. This method is asynchronous and should be called prior to a refresh.<br><br>The parameter input to this method is either a string representing the new token or a function requesting a new token.<br><br>```<br>async function refreshDashboard(<newToken>) {<br>  const embedManager = await createOrGetEmbedManager();<br>  embedManager.updateToken(<newToken>).then(() => {<br>  embedManager.refresh();<br>  });<br>}<br>``` |

# Embedded Dashboard Properties and Objects

Properties can be passed as parameters to the `createComponent` method when embedding Composer dashboards.

Here is a sample embedding `"componentInstanceId":"<id>"`:

```
embedManager.createComponent('dashboard',
       const componentConfig = {
              "dashboardId": "<dashboard-ID>",
              "componentInstanceId":"<component-instance-ID>",
       }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"dashboardId":"<dashboard-ID>"` | none | The dashboard ID. If no dashboard ID is specified, an empty dashboard is embedded.<br>Type: string |
| `"componentInstanceId":"<component-instance-ID>"` | none | The instance ID for the dashboard is generated when the dashboard is created.<br>Type: string |

Here is a sample embedding `"type"`:

```
embedManager.createComponent('dashboard',
       const componentConfig = {
              "dashboardId": "<dashboard-ID>",
              "componentInstanceId":"<component-instance-ID>",
              "type":"dashboard",
       }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"type":"dashboard"` | | The type of embedded Composer component. |

Here is a sample embedding `"application"`:

```
embedManager.createComponent('dashboard',
       const componentConfig = {
              "dashboardId": "<dashboard-ID>",
              "componentInstanceId":"<component-instance-ID>",
              "type":"dashboard",
              "application":{
                     "banner":false,
                     "logo":true
              }
       }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"application.banner": true` | `false` | Indicates whether the Composer top-level navigation banner should be shown with the embedded dashboard. Valid values are `true` or `false`.<br>Type: boolean<br>This property is deprecated and will be removed in a future release. |
| `"application.logo": true` | `false` | Indicates whether the Composer logo should be shown with the embedded dashboard. Valid values are `true` or `false`.<br>Type: boolean<br>This property is deprecated and will be removed in a future release. |

Here is a sample embedding `"interactivityProfileName"`:

```
embedManager.createComponent('dashboard',
       const componentConfig = {
              "dashboardId": "<dashboard-ID>",
              "componentInstanceId":"<component-instance-ID>",
              "type":"dashboard",
              "application":{
                     "banner":false,
                     "logo":true
              }
```

```
                "interactivityProfileName":"interactive",
        }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"interactivityProfileName" :"interactive"` | `interactive` | Determines the way in which your users will be able to work with the embedded dashboard. Valid values are `readonly` and `interactive`. If you do not want the user of your application to change anything and only be able to view the dashboard, specify `readonly`. If you want your users to be able to make changes to the dashboard, specify `interactive`.<br><br>When the mode is `readonly`, the dashboard cannot be changed.<br><br>**Note:** This setting is backward-compatible with the `interactive` option for public shared link embeds. If you are migrating to Composer's JavaScript embed technology, specify `interactive` for this setting to ease the migration.<br><br>Type: string |

Here is a sample embedding `"interactivityOverrides"`:

```
embedManager.createComponent('dashboard',
        const componentConfig = {
                "dashboardId": "<dashboard-ID>",
                "componentInstanceId":"<component-instance-ID>",
                "type":"dashboard",
                "application":{
                        "banner":false,
                        "logo":true
                }
                "interactivityProfileName":"interactive",
                "interactivityOverrides":"<interactivity-overrides-ID>",
        }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"interactivityOverrides":"<interactivity-overrides-ID>"` | none | Specifies specific dashboard and visual interactivity settings for an embedded dashboard. The visual interactivity settings specified in this object will override any interactivity settings specified for the individual visuals.<br><br>These settings will also override the dashboard interactivity profile linked to the dashboard (the one saved with the dashboard) and the dashboard interactivity profile passed by the `interactivityProfileName` property.<br><br>Type: object |

Here is a sample embedding `"mode"`:

```
embedManager.createComponent('dashboard',
       const componentConfig = {
               "dashboardId": "<dashboard-ID>",
               "componentInstanceId":"<component-instance-ID>",
               "type":"dashboard",
               "application":{
                       "banner":false,
                       "logo":true
               }
               "interactivityProfileName":"interactive",
               "interactivityOverrides":"<interactivity-overrides-ID>",
               "mode":"interactive",
       }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"mode":"interactive"` | `interactive` | When a dashboard interactivity profile is specified (using the `interactivityOverrides` object or the `interactivityProfileName` parameter), the `mode` parameter is ignored. The `mode` parameter is also deprecated in Composer and will be removed in a future release.<br><br>Type: string |

Here is a sample embedding `"theme"`:

```
embedManager.createComponent('dashboard',
       const componentConfig = {
              "dashboardId": "<dashboard-ID>",
              "componentInstanceId":"<component-instance-ID>",
              "type":"dashboard",
              "application":{
                     "banner":false,
                     "logo":true
              }
              "interactivityProfileName":"interactive",
              "interactivityOverrides":"<interactivity-overrides-ID>",
              "mode":"interactive",
              "theme":"modern",
       }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"theme":"dark"` | `composer` | The theme for the embedded dashboard. Valid values are `composer`, `modern`, or `dark`. The initial default theme, `composer`, is the same as the `modern` theme. However, if you add your own themes to the application, more options are available in this list and you may have introduced a different default.<br><br>Type: string |

Here is a sample embedding `"editor"`:

```
embedManager.createComponent('dashboard',
       const componentConfig = {
              "dashboardId": "<dashboard-ID>",
              "componentInstanceId":"<component-instance-ID>",
              "type":"dashboard",
              "application":{
                     "banner":false,
                     "logo":true
              }
              "interactivityProfileName":"interactive",
              "interactivityOverrides":"<interactivity-overrides-ID>",
              "mode":"interactive",
              "theme":"modern",
              "editor":{
```

```
                        "placement": "docRight"
                }
        }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"editor.placement":"docRight"` | `modals` | Indicates where the dashboard editor appears. Valid editor placements are `dockRight` and `modals`.<br>Type: string |

Here is a sample embedding `"header"`:

```
embedManager.createComponent('dashboard',
        const componentConfig = {
                "dashboardId": "<dashboard-ID>",
                "componentInstanceId":"<component-instance-ID>",
                "type":"dashboard",
                "application":{
                        "banner":false,
                        "logo":true
                }
                "interactivityProfileName":"interactive",
                "interactivityOverrides":"<interactivity-overrides-ID>",
                "mode":"interactive",
                "theme":"modern",
                "editor":{
                        "placement": "docRight"
                }
                "header": {
                        "title": "My Dash"
                        "showActions": false,
                        "showTitle": false,
                        "visible": false
                }
        }
);
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `"header.title":"<newtitle>"` | none | Allows you to overwrite the title of the embedded dashboard. The embedded dashboard title is read only; it cannot be changed while the dashboard is embedded.<br>Type: string |
| `"header.showActions": false` | `true` | Indicates whether dashboard actions should be visible for the embedded dashboard. Valid values are `true` or `false`.<br>Type: boolean |
| `"header.showTitle": false` | `true` | Indicates whether the dashboard title should be shown for the embedded dashboard. Valid values are `true` or `false`.<br>Type: Boolean |
| `"header.visible": false` | `true` | Indicates whether the dashboard header should be shown for the embedded dashboard. Valid values are `true` or `false`. |

Here is a sample embedding `"intialFilters"`:

```
intialFilters: {
       sourceId: "<source-id>"
       timeFilter: {
               from: "+$start_of_data",
               to: "+$end_of_data",
               timeField: "_saledate",
               },
       filters: [{
               "operation": "BETWEEN"
               "paths": "returns",
               "value": [ 1, 25 ]
       }]
}
```

The following table describes available options for these properties and objects.

| Property/Object | Default | Description |
|---|---|---|
| `initialFilters` | none | Allows you to pass initial filters to the specified dashboard. Pass parameters for `filters`, `sourceId`, `timeFilter`, and `applyFiltersStrategy`.<br>Type: string |

| Property/Object | Default | Description |
| --- | --- | --- |
| `applyFiltersStrategy` | `overrideSamePath` | Use `overrideSamePath` to overwrite filters using the same path.<br>Use `replaceExisting` to remove all filters and use only defined `initialFilters`. |

You can also refresh your data in the dashboard as needed.

| Method | Description |
| --- | --- |
| `component.refreshData()` | Refreshes data in the dashboard when called. |

# Embedded Visual Authoring Properties and Objects

The following properties can be passed as parameters to the `createComponent` method when embedding Composer visual authoring components.

Here is a sample.

**Note:** This example assumes that the EmbedManager has already been initialized, and the ID of a visual template is passed.

```
            const getToken = async () => {
        const response = await fetch('<playground-uri-and-username-token>', {
                method: 'GET',
                credentials: 'same-origin'
});
return response.json().then((result) => {
        return {
                access_token: result.token,
                expires_in: result.expiresIn,
                };
        });
};

const embedManagerPromise = window.initComposerEmbedManager({ getToken: getToken });

        const componentConfig = {
                "theme":"modern",
                "header": {
                        "showTitle": false,
                        "visible": false,
                        "showActions": false,
                        "title": "<title-text>",
                }
        }

const createEmbedComponent = (embedManager, config, containerElementId = 'widget-holder') => {
        embedManager.createComponent("visual-builder", config).then(component => {
                component.render(document.getElementById(containerElementId), { width:"100%", height: "100%" });
                })
        }

        embedManagerPromise.then(embedManager => {
                createEmbedComponent(embedManager, componentConfig, 'widget-holder');
});
```

The following table describes the available properties for `componentConfig`.

| Property/Object | Default | Description |
|---|---|---|
| `"theme":"dark"` | `composer` | The theme for the embedded visual. Valid values are `composer`, `modern`, or `dark`. The initial default theme, `composer`, is the same as the `modern` theme. However, if you add your own themes to the application, more options are available in this list and you may have introduced a different default.<br>Type: string |

The following table describes the available `header` properties.

| Property/Object | Default | Description |
|---|---|---|
| `"showTitle": true` | `true` | Indicates whether the visual header title should be shown for the embedded visual. Valid values are `true` or `false`.<br>Type: boolean |
| `"visible": false` | `true` | Indicates whether the visual header should be visible for the embedded visual. Valid values are `true` or `false`.<br>Type: boolean |
| `"showActions": false` | `true` | Indicates whether the visual header actions should be shown for the embedded visual. Valid values are `true` or `false`.<br>Type: boolean |
| `"title": <text>` | none | Specifies a static title for the visual.<br>Type: string |

The following table describes the available `breadcrumb` properties.

| Property/Object | Default | Description |
|---|---|---|
| `<optional breadcrumb properties>` | | Optional breadcrumb properties are described in Optional Embedded Visual Authoring Breadcrumb Properties. |
| `"breadcrumbs.onClick"` | none | The click action handler for the breadcrumb title.<br>Type: function |
| `"breadcrumbs.target":"_blank"` | none | The link target parameter. Use `"_blank"` to open in a new tab. A valid link target should be specified in quotes (`"<target>"`).<br>Type: string |

| Property/Object | Default | Description |
|---|---|---|
| `"breadcrumbs.title":"<title>"` | none | The first item breadcrumbs title. A valid title should be specified in quotes (`"<title>"`).<br>Type: string |

The following table describes the available properties for source visualID.

| Property/Object | Default | Description |
|---|---|---|
| `"source.visualId":"<id>"` | null | The source ID of a visual template, used for creating a new visual in visual authoring. This represents a predefined visual template. A valid source ID should be specified in quotes (`"<source-visual-ID>"`). If neither `visualId` or `source.visualId` is specified, an empty visual authoring instance will be opened. If both are provided, `visualId` is used.<br>Type: string |
| `"interactivityOverrides"` | none | Specifies interactivity override settings for visual authoring. The interactivity settings specified in this object will override any interactivity settings specified for the individual visuals. For a list of visual interactivity settings you can specify, see Control How Users Interact With A Visual.<br>Type: object |
| `"visualId":"<id>"` | null | The visual ID for an existing visual for visual authoring. A valid visual ID should be specified in quotes (`"<visual-ID>"`). If neither `visualId` or `source.visualId` is specified, an empty visual authoring instance will be opened. If both are provided, `visualId` is used.<br>Type: string |

The following table describes all of the available properties.

| Property/Object | Default | Description |
|---|---|---|
| `<optional breadcrumb properties>` | Optional breadcrumb properties are described in Optional Embedded Visual Authoring Breadcrumb Properties. | |
| `"breadcrumbs.onClick"` | none | The click action handler for the breadcrumb title.<br>Type: function |
| `"breadcrumbs.target":"_blank"` | none | The link target parameter. Use `"_blank"` to open in a new tab. A valid link target should be specified in |

| Property/Object | Default | Description |
|---|---|---|
| | | quotes ("<target>").<br>Type: string |
| "breadcrumbs.title":"<title>" | none | The first item breadcrumbs title. A valid title should be specified in quotes ("<title>").<br>Type: string |
| "header.showActions": false | true | Indicates whether the visual header actions should be shown for the embedded visual. Valid values are true or false.<br>Type: boolean |
| "header.showTitle": false | true | Indicates whether the visual header title should be shown for the embedded visual.<br>Valid values are true or false.<br>Type: boolean |
| "header.title": <text> | none | Specifies a static title for the visual.<br>Type: string |
| "header.visible": false | true | Indicates whether the visual header should be visible for the embedded visual.<br>Valid values are true or false.<br>Type: boolean |
| "interactivityOverrides" | none | Specifies interactivity override settings for visual authoring.<br>The interactivity settings specified in this object will override any interactivity settings specified for the individual visuals.<br>For a list of visual interactivity settings you can specify, see *Control How Users Interact With A Visual*.<br>Type: object |
| "source.visualId":"<id>" | null | The source ID of a visual template, used for creating a new visual in visual authoring. This represents a predefined visual template. A valid source ID should be specified in quotes ("<source-visual-ID>").<br>If neither visualId or source.visualId is specified, an empty visual authoring instance will be opened.<br>If both are provided, visualId is used.<br>Type: string |

| Property/Object | Default | Description |
|---|---|---|
| `"theme":"dark"` | `composer` | The theme for the embedded visual. Valid values are `composer`, `modern`, or `dark`.<br>The initial default theme, `composer`, is the same as the `modern` theme.<br>However, if you add your own themes to the application, more options are available in this list and you may have introduced a different default.<br>Type: string |
| `"visualId":"<id>"` | null | The visual ID for an existing visual for visual authoring. A valid visual ID should be specified in quotes (`"<visual-ID>"`).<br>If neither `visualId` or `source.visualId` is specified, an empty visual authoring instance will be opened.<br>If both are provided, `visualId` is used.<br>Type: string |

**Note:** You can include several controls for embedded visuals, allowing users to select and deselect favorite visuals, as well as filter the list of visuals in the embedded Visual Gallery by favorite status. See *Use The Visual Gallery*.

# Embedded Library Properties and Objects

The library embed feature also includes pre-defined actions for adding a dashboard and opening a dashboard.

Here's what you can do with pre-defined actions:

- Specify an action for embed action
- Specify an action for navigate to link
- Change a cursor if a column has onClick action specified
- Pass an item id to the link `("https://dashboards.company.com/edit/${inventoryitemId}")`

The following Actions are supported:

| Action |
| --- |
| Embed |
| Open |

Below is a code sample for using the onClick property:

| Type | Example |
| --- | --- |
| FunctionClickHandler | `(data?: InventoryItem) => console.log(data)` |
| EmbedClickHandler | `{ type: "embed", parentElement: "#editor-container", // OR document.querySelector("#editor-container") replaceComponent: true, componentParams: { ... } }` |
| LinkClickHandler | |

| Type | Example |
| --- | --- |
| | <pre>{<br>type: "link",<br>href: "https://dashboards.com/edit/${inventoryItemId}",<br>target: "_blank"<br>}</pre> |

# Embedded Source Inventory Properties and Objects

The source inventory embed feature allows you to define the look and feel of the Sources page for your embedded users.

When you embed a list of sources, you can define which columns to include, order them in your preferred layout, and define what controls are available by enabling and disabling interactivity settings. Your users can filter sources, search for sources, and add sources to favorites.

Use the `InteractivityValue` property to specify interactivity parameters for the sources inventory.

| Parameter | Description |
|---|---|
| `"ADD_NEW": true` | When set to `true`, users can create a new data source configuration. When set to `false`, the button to create a new data source is hidden.<br>Type: boolean |
| `"FILTER": true` | When set to `true`, users can filter sources using the quick filter icons to the left of the **Search** field on the sources page. When set to `false`, they cannot filter sources.<br>Type: boolean |
| `"DELETE": true` | When set to `true`, users can delete data source configurations not currently in use by a visual.<br>When set to `false`, they cannot delete data source configurations, and the delete icon is not visible in the UI.<br>Type: boolean |
| `"DESCRIPTION": true` | When set to `true`, users can view and search for items (sources, visual gallery visuals, dashboard in the library) by the contents of an item's description.<br>When set to `false`, description options are not available to users.<br>Type: boolean |
| `"EXPORT": true` | When set to `true`, users can export data source configurations.<br>When set to `false`, users cannot export data source configurations, and the option is not visible in the UI.<br>Type: boolean |
| `"PERMISSIONS": false` | When set to `true`, users can manage user and group permissions for data sources. When set to `false`, they cannot manage permissions for data sources, and the related icons are not visible in the UI.<br>Type: boolean |
| `"FAVORITES": true` | When set to `true`, users can mark the data source as a favorite. When set to `false`, they cannot mark data source favorites and the favorites icons are not visible in the UI.<br>Type: boolean |

| Parameter | Description |
| --- | --- |
| `"ROW_SECURITY": false` | When set to `true`, users can define row security for data sources. When set to `false`, they cannot define row security for data sources, and the related icons are not visible in the UI.<br><br>Type: boolean |
| `"COLUMN_SECURITY": false` | When set to `true`, users can define column security for data sources. When set to `false`, they cannot define column security for data sources, and the related icons are not visible in the UI.<br><br>Type: boolean |
| `"CLEAR_CACHE": true` | When set to `true`, users can clear the cache for a data source. When set to `false`, users cannot clear the cache for a data source and the related icons are not visible in the UI.<br><br>Type: boolean |
| `"AVAILABLE_VISUAL_TYPES": true` | When set to `true`, users invoke the **Available Visual Types** work area to enable and disable available visual types for a source. When set to `false`, they cannot affect available visual types, or see related icons are not visible in the UI.<br><br>Type: boolean |

# Optional Embedded Visual Authoring Breadcrumb Properties

Optionally, you can pass breadcrumb properties as parameters to the `createComponent` method when embedding Composer visual authoring components.

Here is a sample:

```
                    "breadcrumbs": {
                    "title": "<title>",
                    "onClick": () => {console.log('clicked')},
                    "href": "<uri-or-fqdn>",
                    "target": "_blank"
                }
```

The following table describes the optional breadcrumb properties.

| Property/Object | Default | Description |
|---|---|---|
| `"title":"<title>"` | none | The first item breadcrumbs title. A valid title should be specified in quotes (`"<title>"`).<br>Type: string |
| `"onClick"` | none | The click action handler for the breadcrumb title.<br>Type: function |
| `"href":"<uri-or-fqdn>"` | none | The link address of the breadcrumb title.<br>A valid link address, such as a URI or Fully Qualified Domain Name should be specified in quotes (`"<uri-or-fqdn>"`).<br>Type: string |
| `"target":"_blank"` | none | The link target parameter. Use `"_blank"` to open in a new tab.<br>A valid link target should be specified in quotes (`"<uri-or-fqdn>"`).<br>Type: string |

# Interactivity Properties

You can add interactivity override settings to your embed script to provide granular control over what your users can do with an embedded Composer component. Properties can be passed with parameters in the `interactivityOverrides` object of the `createComponent` method when embedding Composer components.

The `interactivityProfileName` property must be specified before the interactivity overrides (`interactivityOverrides` object) can work. See Embedded Dashboard Properties And Objects.

The `interactivityOverrides` object includes several property settings: `visualSettings`, `InteractivityValue`and `editorUserSettings`.

- The parameters of the `settings` property affect the current embedded dashboard interactivity. See Dashboard Interactivity Parameters (settings Property).
- The parameters of the `visualSettings` object property affect the current embedded visual interactivity. See Visual Interactivity Parameters (visualSettings Property).
- The parameters of the `InteractivityValue` property define the source inventory dashboard. See Sources Inventory Interactivity Parameters (InteractivityValue Property) .
- The parameters of the `editorUserSettings` property define the editor configuration dashboard. See Editor Configuration (editorUserSettings Property).

The following example depicts the use and placement of dashboard and visual interactivity parameters.

```
<script>
   src="<server>:8443/composer/embed/embed.js",
     {
        "type": "dashboard",
        "dashboardId": "<dashboard-ID>",
        "theme": "modern",
        "interactivityProfileName": "interactive",
        "interactivityOverrides": {
          "settings":{
             "CHANGE_LAYOUT": true
          },
          "visualSettings":{
             "FILTER": false
          },
        }
     }
</script>
```

Custom user attributes can be used as variables in these property value settings. For example, the following property setting is valid and will insert the value of the `var2` custom user attribute in the property setting. If a `var2` custom user attribute is not found for a user, a value of `true` is assumed.

While it is possible to save a dashboard profile using custom user attributes, you cannot pass the dashboard profile with custom user attributes in the JavaScript.

```
"CHANGE_LAYOUT": "${User.var2|true}"
```

# Dashboard Interactivity Parameters (`settings` Property)

Use the `settings` property to specify interactivity parameters for the dashboard. The default for all dashboard interactivity parameters is `true`.

| Parameter | Description |
|---|---|
| `"ADD_TO_FAVORITES": false` | When set to `true`, users can mark the dashboard as a favorite. When set to `false`, they cannot mark dashboard favorites and the dashboard icon is not available in the UI.<br>Type: boolean |
| `"ADD_VISUALS": false` | When set to `true`, these options are accessible in the dashboard UI:<br>▪ Select the dashboard icon to add existing visuals to the dashboard (**Add Existing Visual**).<br>▪ Select **Add to Visual Gallery** from the menu to replace local visuals with Visual Gallery visuals. See *Add Visuals To The Visual Gallery*.<br>When set to `false`, users cannot add existing visuals or replace visuals in the dashboard.<br>Type: boolean<br>**Note:** If all menu item parameters are `false`, the dashboard icon is not shown in the UI. If any menu item parameters are `true`, the dashboard icon is shown in the UI, and includes any items set to `true`. |
| `"CHANGE_LAYOUT": true` | When set to `true`, users can resize or move visuals in the dashboard. When set to `false`, they cannot resize or move visuals in the dashboard.<br>Type: boolean |
| `"COMMENTS": true` | When set to `true`, users with appropriate permissions can access comments (read, write, edit their own, and delete comments, depending on their access level). When set to `false`, comments are not available in the UI.<br>Type: boolean |

| Parameter | Description |
|---|---|
| `"CREATE_FILTER_SNIPPETS": false` | When set to `true`, users can add new filter snippets (  ) to the dashboard using the dashboard icons. When set to `false`, they cannot create new snippets to add to the dashboard.<br><br>Type: boolean<br><br>**Note:** If all menu item parameters are `false`, the dashboard icon is not shown in the UI. If any menu item parameters are `true`, the dashboard icon is shown in the UI, and includes any items set to `true`. |
| `"CREATE_TEXT_SNIPPETS": false` | When set to `true`, users can add new text snippets (  ) to the dashboard using the dashboard icons. When set to `false`, they cannot create new snippets to add to the dashboard.<br><br>Type: boolean<br><br>**Note:** If all menu item parameters are `false`, the dashboard icon is not shown in the UI. If any menu item parameters are `true`, the dashboard icon is shown in the UI, and includes any items set to `true`. |
| `"CREATE_VISUALS": false` | When set to `true`, users with **Owner** and **Editor** access levels or the **Administer Visuals** privilege, these options are accessible in the dashboard UI:<br><br>▪ Select the dashboard icon  to add new local visuals to the dashboard (**Add New Visual**).<br><br>▪ Select **Convert to Local** from the  menu to convert Visual Gallery visuals to local visuals. See *Convert Visual Galley Visuals To Local Visuals*.<br><br>When set to `false`, they cannot create new local visuals or convert Visual Gallery visuals in the dashboard.<br><br>Type: boolean<br><br>**Note:** If all menu item parameters are `false`, the dashboard icon is not shown in the UI. If any menu item parameters are `true`, the dashboard icon is shown in the UI, and includes any items set to `true`. |
| `"DASHBOARD_ALERTS": false` | When set to `true`, users with appropriate privileges can create and edit dashboard alerts using the dashboard icon  . When set to `false`,the dashboard icon is hidden and users cannot create or edit dashboard alerts. |
| `"DASHBOARD_INTERACTIONS": true` | When set to `true`, users can link fields between disparate data sources to create cross-source links. When set to `false`, they cannot create cross-source links and the  dashboard icon is not available in the UI. |

| Parameter | Description |
|---|---|
| `"DASHBOARD_LINKS": true` | When set to `true`, users can link dashboards. When set to `false`, they cannot link dashboards and the dashboard icon is not available in the UI.<br><br>Type: boolean |
| `"DELETE": true` | When set to `true`, users can delete the dashboard. When set to `false`, they cannot delete the dashboard and the dashboard icon is not available in the UI.<br><br>Type: boolean |
| `"EXPORT_CONFIGURATION": true` | When set to `true`, users can export the JSON configuration for the dashboard. When set to `false`, the Export dialog in the UI no longer provides an option to export the configuration. If this setting and the `EXPORT_PNG_PDF` setting are both turned off, the dashboard icon is not available in the UI.<br><br>Type: boolean |
| `"EXPORT_CSV": true` | When set to `true`, users can Export the dashboard as a CSV file. When set to `false`, they cannot export the dashboard as a CSV file and the Export dialog in the UI no longer provides this export option. If all export settings and the `EXPORT_CONFIGURATION` setting are both turned off, the dashboard icon is not available in the UI.<br><br>Type: boolean |
| `"EXPORT_PNG_PDF": true` | When set to `true`, users can Export the dashboard as a PNG or PDF file. When set to `false`, they cannot export the dashboard as a PNG or PDF file and the Export dialog in the UI no longer provides this export option.<br>If all export settings and the `EXPORT_CONFIGURATION` setting are both turned off, the dashboard icon is not available in the UI.<br><br>Type: boolean |
| `"EXPORT_XLSX": true` | When set to `true`, users can Export the dashboard as an Excel (.xlsx) file. When set to `false`, they cannot export the dashboard as an Excel (.xlsx) file and the Export dialog in the UI no longer provides this export option.<br>If all export settings and the `EXPORT_CONFIGURATION` setting are both turned off, the dashboard icon is not available in the UI.<br><br>Type: boolean |
| `"FILTER": true` | When set to `true`, users can apply filters to the dashboard. When set to `false`, they cannot apply dashboard filters and the dashboard icon is not available in the UI. |

| Parameter | Description |
|---|---|
| | Type: boolean |
| `"REFRESH": true` | When set to `true`, users can refresh the dashboard data. When set to `false`, they cannot refresh dashboard data and the  dashboard icon is not available in the UI.<br>Type: boolean |
| `"RENAME": true` | When set to `true`, users can rename the dashboard. When set to `false`, they cannot rename the dashboard.<br>Type: boolean |
| `"SAVE_AS": true` | When set to `true`, users can save the dashboard (make a copy) using a new name. When set to `false`, they cannot save the dashboard with a new name and the  dashboard icon is not available in the UI.<br>Type: boolean |
| `"SAVE": true` | When set to `true`, users can save the dashboard. When set to `false`, they cannot save the dashboard and the  dashboard icon is not available in the UI.<br>Type: boolean |
| `"SCHEDULE_REPORTS": true` | When set to `true`, users with appropriate privileges can schedule dashboard reports using the dashboard icon  . When set to `false`, the dashboard icon is hidden and they cannot schedule dashboard reports.<br>Type: boolean |
| `"SHARE_DASHBOARD": true` | When set to `true`, users can share dashboards using the dashboard icon  . When set to `false`, the dashboard icon is hidden and they cannot share dashboards.<br>Type: boolean |
| `"SHARE_FILTER_SETS": true` | When set to `true`, users can share saved filter sets with other users. When set to `false`, users cannot view or use filter sets for the dashboard.<br>Type: boolean |
| `"WIDGETS": true` | When set to `true`, users can share saved filter sets with other users. When set to `false`, users cannot view or use the widget settings sidebar menu.<br>Type: boolean |

# Visual Interactivity Parameters (`visualSettings` Property)

Use the `visualSettings` property to specify visual interactivity parameters for the embedded visuals. If the `overrideVisualInteractivity` property is set to `true` within these settings, the visual interactivity parameters will override any interactivity properties specified for the individual visuals.

The defaults for these parameters are determined by the interactivity profile.

- If the interactivity profile is `readonly`, then the `overrideVisualInteractivity` property is `true` and all `visualSettings` parameters are `false`. If the interactivity profile is `interactive`, then the `overrideVisualInteractivity` property is set to `false`, all `visualSettings` parameters are empty, and the visual interactivity settings for the individual visuals are used.

- When a custom interactivity profile is saved for a dashboard and `overrideVisualInteractivity` is set to `false`, then all `visualSettings` parameters are empty and the visual interactivity settings for the individual visuals are used. If `overrideVisualInteractivity` is set to `true`, then the `visualSettings` for the dashboard are used.

| Parameter | Description |
|---|---|
| `"ACTIONS": false` | When set to `true`, users can invoke an action using the **Actions** option from the visual drop-down menu. When set to `false`, they cannot invoke actions.<br>Type: boolean |
| `"ACTIONS_ACTION": false` | When set to `true`, users can invoke an action using the **Actions** option on the context menu. When set to `false`, they cannot invoke actions.<br>Type: boolean |
| `"COLORS": false` | When set to `true`, users can change the color palette used by the visual.<br>When set to `false`, they cannot change the color palette . The color settings (  ) visual sidebar menu option is disabled and they cannot access the color sidebar.<br>Type: boolean |
| `"CONDITIONAL_FORMATTING": false` | When set to `true`, users define conditional formatting for the visual.<br>When set to `false`, they cannot define conditional formatting. The conditional formatting visual sidebar menu option is disabled and can't be accessed.<br>Type: boolean |
| `"COPY": false` | When set to `true`, users can copy a visual using the **Copy Visual** option from the visual drop-down menu. When set to `false`, they cannot copy visuals. |

| Parameter | Description |
|---|---|
| | Type: boolean |
| `"DETAILS_ACTION": false` | When set to `true`, users can display additional information about a specific visual data element using the **Details** option on the context menu. When set to `false`, users cannot display additional information about a specific visual data element using the context menu.<br><br>Type: boolean |
| `"EXPORT_CSV": false` | When set to `true`, users can Export the visual as a CSV file. When set to `false`, they cannot export the visual this way, and the UI no longer provides this export option. If all export settings are turned off, the export menu is removed.<br><br>Type: boolean |
| `"EXPORT_PNG_PDF": false` | When set to `true`, users can Export the visual as a PNG or PDF file. When set to `false`, they cannot export the visual this way, and the UI no longer provides this export option. If all export settings are turned off, the export menu is removed.<br><br>Type: boolean |
| `"EXPORT_XLSX": false` | When set to `true`, users can Export the visual as an Excle (.xlsx) file. When set to `false`, they cannot export the visual this way, and the UI no longer provides this export option. If all export settings are turned off, the export menu is removed.<br><br>Type: boolean |
| `"FILTER": false` | When set to `true`, users can filter visual data using the **Filter** option from the visual drop-down menu and to the left of the visual name on a visual, or display name if viewed in a dashboard. When set to `false`, they cannot filter visual data.<br><br>Type: boolean |
| `"FILTER_ACTION": false` | When set to `true`, users can filter data using the **Filter** option on the context menu. When set to false, users cannot filter data using the **Filter** option on the context menu.<br><br>Type: boolean |
| `"FORMAT": false` | When set to `true`, users can Format specific data in a visual. When set to `false`, they cannot format visual data.<br><br>Type: boolean |
| `"GROUPING": false` | When set to `true`, users can change the **Group** field on the x-axis of the visual. When set to `false`, they cannot change the **Group** field.<br><br>Type: boolean |

| Parameter | Description |
|---|---|
| `"INFO": false` | When set to `true`, users can see visual details in the Info sidebar menu. When set to `false`, they cannot see those details.<br><br>Type: boolean |
| `"KEYSET": false` | When set to `true`, users can create a keyset from the visual data using the **Create Keyset** option from the visual drop-down menu. When set to `false`, users cannot create keysets.<br><br>Type: boolean |
| `"KEYSET_ACTION": false` | When set to `true`, users can create a keyset from a selected data point using the **Keyset** option on the context menu. When set to `false`, users cannot create a keyset using the context menu. |
| `"LINK_ACTION": false` | When set to `true`, users can link to another dashboard using the **Link** option on the context menu. When set to `false`, users cannot link to another dashboard using the **Link** option on the context menu.<br><br>Type: boolean |
| `"MAXIMIZE": false` | When set to `true`, users can maximize a visual for optimal viewing. When set to `false`, they cannot maximize a visual.<br><br>Type: boolean |
| `"METRICS": false` | When set to `true`, users can change metric fields (other than a metric that might be in the **Group** field) on the axes for the visual. This setting also controls whether a user can control the aggregation method (SUM, AVG, MIN, MAX, etc.) used for metrics in a table. When set to `false`, users cannot change metric fields or control the aggregation method used for metrics on a visual.<br><br>Type: boolean |
| `"overrideVisualInteractivity": false` | When set to `true`, the individual visual interactivity settings for each visual on the dashboard are overridden and ignored. Instead, the visual interactivity settings set for the dashboard are used for all of the visuals in the dashboard. These settings are only applied to the visuals when they are used in this dashboard and not universally.<br><br>When set to `false`, the individual visual interactivity settings for each visual are honored.<br><br>Type: boolean |
| `"REMOVE": false` | When set to `true`, users can remove a visual using the **Remove Visual** option from the visual drop-down menu. When set to `false`, users cannot remove visuals.<br><br>Type: boolean |
| `"RENAME": false` | When set to `true`, users can change the display name of a visual. When set to `false`, they cannot change the display name of visual. |

| Parameter | Description |
|---|---|
| | Type: boolean |
| `"RULERS": false` | When set to `true`, users can add visual reference lines and customize the markers used on the metric axis.<br><br>When set to `false`, users cannot add reference lines and markers. The ruler settings (  ) visual sidebar menu option is disabled when this switch is off and you cannot access the ruler sidebar.<br><br>Type: boolean |
| `"SAVE_AS": false` | When set to `true`, users can save a shared visual (make a copy) using a new name. When set to `false`, they cannot save the shared visual with a new name and the option is not available in the UI.<br><br>Type: boolean |
| `"SAVE": false` | When set to `true`, users can save the shared visual. When set to `false`, they cannot save the shared visual and the option is not available in the UI.<br><br>Type: boolean |
| `"SETTINGS": false` | When set to `true`, users can specify settings for the visual using the visual settings option on the visual sidebar menu.<br><br>When set to `false`, users cannot specify visual settings. The visual settings (  ) visual sidebar menu option is disabled and users cannot access the visual settings sidebar.<br><br>Type: boolean |
| `"SORT": false` | When set to `true`, users can sort and limit the data in a visual.<br><br>When set to `false`, users cannot sort and limit visual data. The sort and limit (  ) visual sidebar menu option is disabled when this switch is off and you cannot access the Sort & Limit sidebar.<br><br>Type: boolean |
| `"TIMEBAR_FIELD": false` | When set to `true`, users can change the time field on the time bar. When set to `false`, users cannot change the time field.<br><br>Type: boolean |
| `"TIMEBAR_PANEL": false` | When set to `true`, users can show and hide the time bar on a visual. When set to `false`, the time bar is hidden for this visual.<br><br>Type: boolean |

| Parameter | Description |
|---|---|
| `"TREND_ACTION": false` | When set to `true`, users can view trends for a selected data point using the **Trend** option on the context menu. When set to `false`, users cannot view trends for a selected data point using the context menu.<br><br>Type: boolean |
| `"VISUAL_STYLE": false` | When set to `true`, users can change the style of a visual.<br><br>When set to `false`, users cannot change the visual style. The visual style settings (  ) visual sidebar menu option is disabled when this switch is off and users cannot access the visual type sidebar.<br><br>Type: boolean |
| `"ZOOM_ACTION": false` | When set to `true`, users can zoom into a selected data point on a visual using the **Zoom** option on the context menu. When set to `false`, users cannot zoom into a selected data point on a visual using the **Zoom** option on the context menu.<br><br>Type: boolean |
| `"ZOOM_ACTION": false` | When set to `true`, users can drill down in a selected data point on a visual using the **Zoom** option on the context menu. When set to `false`, users cannot drill down in a selected data point on a visual using the **Zoom** option on the context menu.<br><br>Type: boolean |

# Sources Inventory Interactivity Parameters (`InteractivityValue` Property)

Use the `InteractivityValue` property to specify interactivity parameters for the sources inventory.

| Parameter | Description |
|---|---|
| `"ADD_NEW": true` | When set to `true`, users can create a new data source configuration. When set to `false`, the button to create a new data source is hidden.<br><br>Type: boolean |
| `"FILTER": true` | When set to `true`, users can filter sources using the quick filter icons to the left of the **Search** field on the sources page. When set to `false`, they cannot filter sources.<br><br>Type: boolean |
| `"DELETE": true` | When set to `true`, users can delete data source configurations not currently in use by a visual. |

| Parameter | Description |
|---|---|
| | When set to `false`, they cannot delete data source configurations, and the delete icon is not visible in the UI.<br>Type: boolean |
| `"DESCRIPTION": true` | When set to `true`, users can view and search for items (sources, visual gallery visuals, dashboard in the library) by the contents of an item's description.<br>When set to `false`, description options are not available to users.<br>Type: boolean |
| `"EXPORT": true` | When set to `true`, users can export data source configurations.<br>When set to `false`, users cannot export data source configurations, and the option is not visible in the UI.<br>Type: boolean |
| `"PERMISSIONS": false` | When set to `true`, users can manage user and group permissions for data sources. When set to `false`, they cannot manage permissions for data sources, and the related icons are not visible in the UI.<br>Type: boolean |
| `"FAVORITES": true` | When set to `true`, users can mark the data source as a favorite. When set to `false`, they cannot mark data source favorites and the favorites icons are not visible in the UI.<br>Type: boolean |
| `"ROW_SECURITY": false` | When set to `true`, users can define row security for data sources. When set to `false`, they cannot define row security for data sources, and the related icons are not visible in the UI.<br>Type: boolean |
| `"COLUMN_SECURITY": false` | When set to `true`, users can define column security for data sources. When set to `false`, they cannot define column security for data sources, and the related icons are not visible in the UI.<br>Type: boolean |
| `"CLEAR_CACHE": true` | When set to `true`, users can clear the cache for a data source. When set to `false`, users cannot clear the cache for a data source and the related icons are not visible in the UI.<br>Type: boolean |
| `"AVAILABLE_VISUAL_TYPES": true` | When set to `true`, users invoke the **Available Visual Types** work area to enable and disable available visual types for a source. When set to `false`, they cannot affect available visual types, or see related icons are not visible in the UI.<br>Type: boolean |

# Editor Configuration (`editorUserSettings` Property)

Use the `editorUserSettings` property to configure the interactivity parameters for the dashboard.

```
{
                        "type": "dashboard",
                        editorUserSettings: {
                        isViewMode: true,
                        canSwitchMode: false,
                        allowInteractivityConfiguration: false,
                        }
                }
```

| Parameter | Description |
|---|---|
| `isViewMode` | When set to `true`, the dashboard opens for editors and creators in View mode. When set to `false`, the dashboard opens for editors and creators in Edit mode.<br>Type: boolean<br>Default: `false` |
| `canSwitchMode` | When set to `true`, editors can toggle between View and Edit modes. When set to `false`, editors can't toggle between View and Edit modes.<br>Type: boolean<br>Default: `true` |
| `allowInteractivityConfiguration` | When set to `true`, editors can change interactivity settings for the dashboard and visuals. When set to `false`, editors can't change interactivity settings for the dashboard and visuals.<br>Type: boolean<br>Default: `false` |

# Embedded Events

Events can be used in your JavaScript to control an embedded Composer component when specific events occur.

**Note:** The ability for your end-users to perform some of the events listed here is controlled by the permissions granted to them with their Composer credentials. See Embedded Composer Component Controls.

Embedded events are described in the following tables:

- Document Events
- Dashboard Events
- Visual Authoring Events
- Visual Events

Sample code showing how to subscribe to the events that trigger the event listeners is provided in each description. In these examples, when the event is triggered, a log message is written to the console. You can alter this behavior using custom JavaScript appropriate for your installation. For example, you can trigger a pop-up message (instead of a log message) using the Javascript `alert()` method instead of the `console.log()` method. For example: `alert ("Dashboard has been loaded !!!");`

**Important:** The event detail payload that is passed to the event listener is ***experimental*** and might be changed. Mutating the detail object (`e.detail.*`) will not update the UI; the object is read-only.

**Note:** Composer provides a `componentInstanceId` provides a as part of event details for dashboard, source, and visual events.

# Document Events

| Event | Description |
|---|---|
| `composer-init-failed` | Triggered by document if the component fails to initialize. No data is passed in the event.<br>`document.addEventListener("composer-init-failed", () => { console.log("composer-init-failed");` |

| Event | Description |
| --- | --- |
| | `});` |
| `composer-unauthorized` | Triggered by document when the authorization token expires. No data is passed in the event.<br>`document.addEventListener("composer-unauthorized", () => {`<br>`  console.log("composer-unauthorized");`<br>`});` |

# Dashboard Events

| Event | Description |
| --- | --- |
| `composer-dashboard-changed` | Triggered by the dashboard when there is any change in the dashboard configuration. This is useful for tracking the active state of the dashboard. The data passed in the event includes the dashboard and dashboard data (`e.detail.dashboard`).<br>`dashboard.addEventListener("composer-dashboard-changed", (e) => {`<br>`  console.log(e.detail.dashboard);`<br>`});` |
| `composer-dashboard-deleted` | Triggered by the dashboard when the dashboard is deleted. The data passed in the event includes the dashboard and dashboard data (`e.detail.dashboard`).<br>`dashboard.addEventListener("composer-dashboard-deleted", (e) => {`<br>`  console.log(e.detail.dashboard);`<br>`});` |
| `composer-dashboard-failed` | Triggered by the dashboard when the dashboard fails to load. The data passed in the event includes the failure reason (`failedReason`).<br>`dashboard.addEventListener("composer-dashboard-failed", (e) => {`<br>`  console.log(e.detail.failedReason);`<br>`});` |

| Event | Description |
|---|---|
| `composer-dashboard-loaded` | Triggered when a dashboard resource is loaded. The data passed in the event includes the dashboard and dashboard data (`e.detail.dashboard`).<br>`dashboard.addEventListener("composer-dashboard-loaded", (e) => {`<br>`  console.log(e.detail.dashboard);`<br>`});` |
| `composer-dashboard-ready` | Triggered by the dashboard when all visuals on the dashboard have been rendered. The data passed in the event includes the dashboard and dashboard data (`e.detail.dashboard`).<br>`dashboard.addEventListener("composer-dashboard-ready", (e) => {`<br>`  console.log(e.detail.dashboard);`<br>`});` |
| `composer-dashboard-saved` | Triggered by the dashboard when a dashboard resource has been successfully saved. The data passed in the event includes the dashboard and dashboard data (`e.detail.dashboard`).<br>`dashboard.addEventListener("composer-dashboard-saved", (e) => {`<br>`  console.log(e.detail.dashboard);`<br>`});` |
| `composer-dashboard-widget-added` | Triggered by the dashboard when a visual is added to the dashboard. The data passed in the event includes the visual and visual data (`e.detail.visual`).<br>`dashboard.addEventListener("composer-dashboard-widget-added", (e) => {`<br>`  console.log(e.detail.visual);`<br>`});` |
| `composer-dashboard-widget-removed` | Triggered by the dashboard when a visual is removed from the dashboard. The data passed in the event includes the visual and visual data (`e.detail.visual`).<br>`dashboard.addEventListener("composer-dashboard-widget-removed", (e) => {`<br>`  console.log(e.detail.visual);`<br>`});` |
| `composer-dashboard-pristine` | Triggered by the dashboard when changes are made but not saved. The data passed in the event includes the |

| Event | Description |
|---|---|
| | dashboard and dashboard data (`e.detail.dashboard`).<br><br>`dashboard.addEventListener("composer-dashboard-pristine", (e) => {`<br>`  console.log(e.detail.dashboard);`<br>`});` |
| `composer-dashboard-dirty` | Triggered by the dashboard when there are no unsaved changes to the dashboard. The data passed in the event includes the dashboard and dashboard data (`e.detail.dashboard`).<br><br>`dashboard.addEventListener("composer-dashboard-widget-removed", (e) => {`<br>`  console.log(e.detail.dashboard);`<br>`});` |

# Visual Authoring Events

| Event | Description |
|---|---|
| `composer-visual-builder-changed` | Triggered when there is any change in the nested visual or the visual authoring configuration. The data passed in the event includes the visual authoring configuration (`e.detail.visualBuilder`).<br><br>`embeddedComponent.addEventListener("composer-visual-builder-changed", (e) => {`<br>`  console.log(e.detail.visualBuilder);`<br>`});` |
| `composer-visual-builder-failed` | Triggered when visual authoring fails to load. The data passed in the event includes the visual builder data as well as the failure reason (`e.detail.failedReason`).<br><br>`embeddedComponent.addEventListener("composer-visual-builder-failed", (e) => {`<br>`  console.log(e.detail.visualBuilder);`<br>`  console.log(e.detail.failedReason);`<br>`});` |
| `composer-visual-builder-loaded` | Triggered when visual authoring is loaded and the visual authoring shell is rendered. The data passed in the event includes the visual authoring configuration (`e.detail.visualBuilder`). |

| Event | Description |
| --- | --- |
| | `embeddedComponent.addEventListener("composer-visual-builder-loaded", (e) => { console.log(e.detail.visualBuilder); });` |
| `composer-visual-builder-ready` | Triggered when the nested visual is rendered. The data passed in the event includes the visual authoring configuration (`e.detail.visualBuilder`).<br>`embeddedComponent.addEventListener("composer-visual-builder-ready", (e) => { console.log(e.detail.visualBuilder); });` |

# Visual Events

| Event | Description |
| --- | --- |
| `composer-visual-failed` | Triggered by the dashboard or by visual authoring when a visual within the dashboard fails to load. The data passed in the event includes the visual and visual data as well as the failure reason (`failedReason`).<br>`embeddedComponent.addEventListener("composer-visual-failed", (e) => { console.log(e.detail.visual); console.log(e.detail.failedReason); });` |
| `composer-visual-loaded` | Triggered by the dashboard or visual authoring when a visual within the dashboard is loaded. The data passed in the event includes the visual and visual data as well as the visualization instance ID (`e.detail.visual`).<br>`embeddedComponent.addEventListener("composer-visual-loaded", (e) => { console.log(e.detail.visual); });` |
| `composer-visual-rendered` | Triggered by the dashboard or visual authoring when a visual within the dashboard is fully rendered. The data passed in the event includes the visual and visual data as well as the visualization instance ID (`e.detail.visual`). |

| Event | Description |
|---|---|
| | `embeddedComponent.addEventListener("composer-visual-rendered", (e) => { console.log(e.detail.visual); });` |
| `composer-visual-saved` | Triggered when visual authoring has been successfully saved. The data passed in the event includes the visual authoring configuration (`e.detail.visual`).<br>`vb.addEventListener("composer-visual-saved", (e) => { console.log(e.detail.visual); });` |
| `composer-visual-series-mousemove` | Triggered when a user hovers over one item in a series, such as a bar on a bar chart, or sector in a pie visual. The data passed in the event includes `componentInstanceId`, `visualApi`, and the current series in `e.detail`.<br>`embeddedComponent.addEventListener("composer-visual-series-mousemove", (e) => { console.log(e.detail.componentInstanceId); console.log(e.detail.visualApi); console.log(e.detail.data); });` |
| `composer-visual-series-mouseout` | Triggered when a user stops hovering over an item in a series, such as a bar on a bar chart, or sector in a pie visual.<br>`embeddedComponent.addEventListener("composer-visual-series-mouseout", (e) => { console.log(e.detail.componentInstanceId); console.log(e.detail.visualApi); });` |
| `composer-visual-mousemove` | Triggered when a user hovers over a space that is empty but related to a visual, such as between the bars of a visual, or in a blank spot on a pie visual.<br>`embeddedComponent.addEventListener("composer-visual-series-mousemove", (e) => { console.log(e.detail.componentInstanceId); console.log(e.detail.visualApi); });` |

| Event | Description |
| --- | --- |
| composer-visual-mouseout | Triggered when a user stops hovering the visual.<br>`embeddedComponent.addEventListener("composer-visual-series-mouseout", (e) => { console.log(e.detail.componentInstanceId); console.log(e.detail.visualApi); });` |

# Source Editor Events

| Event | Description |
| --- | --- |
| composer-source-editor-ready | Triggered when the source editor is rendered the first time. The data passed in the event is `undefined`.<br>`sourceEditor.addEventListener("composer-source-editor-ready", (e) =>{ console.log(e); });` |
| composer-source-definition-ready | Triggered when a source is loaded on Source Creation tab. The data passed in the event include the source definition: `e.detail.source`.<br>`sourceEditor.addEventListener("composer-source-definition-ready", (e) => { console.log(e.detail.source); });` |
| composer-source-fields-ready | Triggered when source fields are loaded on the Fields tab. The data passed in the event includes the source fields:`e.detail.fields`.<br>`sourceEditor.addEventListener("composer-source-fields-ready", (e) => { console.log(e.detail.fields); });` |
| composer-source-cache-ready | Triggered when source cache settings are loaded on the Cache tab. The data passed in the event includes the source cache settings: `e.detail.cache`.<br>`sourceEditor.addEventListener("composer-source-cache-ready", (e) => { console.log(e.detail.cache); });` |
| composer-source- | Triggered when source global settings are loaded on the Global Settings tab. The data passed in the event includes the source global |

| Event | Description |
|---|---|
| `settings-ready` | settings: `e.detail.settings`.<br>`sourceEditor.addEventListener("composer-source-settings-ready", (e) => {`<br>`  console.log(e.detail.settings);`<br>`});` |
| `composer-source-created` | Triggered when a source is created. The data passed in the event includes the source definition: `e.detail.source`.<br>`sourceEditor.addEventListener("composer-source-created", (e) => {`<br>`  console.log(e.detail.source);`<br>`});` |
| `composer-source-saved` | Triggered when a source is saved. The data passed in the event includes the source definition: `e.detail.source`.<br>`sourceEditor.addEventListener("composer-source-saved", (e) => {`<br>`  console.log(e.detail.source);`<br>`});` |
| `composer-source-field-created` | Triggered when a source field is created. The data passed in the event includes the source field: `e.detail.field`.<br>`sourceEditor.addEventListener("composer-source-field-created", (e) => {`<br>`  console.log(e.detail.field);`<br>`});` |
| `composer-source-field-saved` | Triggered when a source field is saved. The data passed in the event includes the source field: `e.detail.field`.<br>`sourceEditor.addEventListener("composer-source-field-saved", (e) => {`<br>`  console.log(e.detail.field);`<br>`});` |
| `composer-source-field-deleted` | Triggered when a source field is deleted. The data passed in the event includes the source field: `e.detail.field`.<br>`sourceEditor.addEventListener("composer-source-field-deleted", (e) => {`<br>`  console.log(e.detail.field);`<br>`});` |

| Event | Description |
| --- | --- |
| composer-source-metric-created | Triggered when a source metric is created. The data passed in the event includes the source metric: `e.detail.metric`.<br>`sourceEditor.addEventListener("composer-source-metric-created", (e) => {`<br>`  console.log(e.detail.metric);`<br>`});` |
| composer-source-metric-saved | Triggered when a source metric is saved. The data passed in the event includes the source metric: `e.detail.metric`.<br>`sourceEditor.addEventListener("composer-source-metric-saved", (e) => {`<br>`  console.log(e.detail.metric);`<br>`});` |
| composer-source-metric-deleted | Triggered when a source metric is deleted. The data passed in the event includes the source metric: `e.detail.metric`.<br>`sourceEditor.addEventListener("composer-source-metric-deleted", (e) => {`<br>`  console.log(e.detail.metric);`<br>`});` |

# Embedded JavaScript Examples

This section provides JavaScript examples that use methods, properties, and embedded events of the `EmbedManager` class to embed, refresh, reauthenticate, or remove Composer components from your application.

- Embedded Dashboard JavaScript Example
- Embedded Visual Authoring JavaScript Example

# Embedded Dashboard JavaScript Example

The following JavaScript example uses methods, properties, and embedded events of the `EmbedManager` class to embed, refresh, reauthenticate, and remove a dashboard.

- The `createOrGetEmbedManager` asynchronous function provides an initial access token using Trusted Access.
- The `addDashboard` asynchronous function embeds a dashboard in the application. It uses the dashboard ID as input so it knows which dashboard to embed. It also specifies some properties for the dashboard.
- The `clearDashboard` asynchronous function removes a dashboard from the application. It uses the dashboard ID as input so it knows which dashboard to remove.
- The `refreshDashboard` asynchronous function refreshes the authorization token for the embedded dashboard using Trusted Access.
- The `application` asynchronous function at the end combines the use of all the previous functions and uses embedded events to trigger some of them.

**Note:** The Composer `EmbedManager` class must be made available to your HTML application prior to using this sample in your application. See Embed Composer Components Using JavaScript And Trusted Access.

**Note:** Composer provides a `componentInstanceId` provides a as part of event details for dashboard, source, and visual events.

```
async function createOrGetEmbedManager(<token>) {
   return window.initComposerEmbedManager({ //embed manager is a singleton and will be created only on the first call
      getToken: function () {
          // transform for the embed syntax
          return getToken().then((result) => {
              return {
                  "access_token": "result.token",
                  "expires_in": "result.expiresIn",
              };
          });
      }
   });
}
```

```
async function addDashboard(<id>) { // id of the composer dashboard
   const embedManager = await createOrGetEmbedManager();

   const dashboard = await embedManager.createComponent('dashboard', {
      "dashboardId": "<id>", // required
      "theme": "composer",
      "interactivityProfileName": "interactive",
      interactivityOverrides {
        "settings":{
           "CHANGE_LAYOUT": true,
        },
        "visualSettings":{
           "FILTER": false,
        },
     },
     "editor": {
         "placement": "dockRight" // use eve sidepanel
      },
      "header": {
         "visible": true,
         "showTitle": true,
         "showActions": false, // will hide actions bar
         "title": "Static custom title"
      }
   });

   dashboard.render(document.querySelector('#dashboard'), { width: '800px', height: '400px' });
   return dashboard;
}

async function clearDashboard(<id>) {
   const embedManager = await createOrGetEmbedManager();
   embedManager.removeComponent(<id>);
}

async function refreshDashboard(<newToken>) {
   const embedManager = await createOrGetEmbedManager();
   embedManager.updateToken(<newToken>).then(() => {
      embedManager.refresh();
   });
}

(async function application() {
   const token = await getComposerToken(); // some function of the 3rd party app that calls application API and return
```

```
composer token retrieved via TA
   const dashboards = await getUserDashboard(); // some function of the 3rd party app that retrieved dashboards for the
user
   createOrGetEmbedManager(token);

   // if token is expired composer will raise next event
   document.addEventListener('composer-unathorized', async () => {
      const token = await getComposerToken();
      const embedManager = createOrGetEmbedManager();
      refreshDashboard(token);
   })
   // render first dashboard in the list
   const embeddedDashboard = addDashboard(dashboards[0].id);

   // embedded component supports an ability to subscribe on the component specific events
   embeddedDashboard.addEventListener('composer-dashboard-ready', () => {
      // triggered when all charts are rendered. put custom logic here.
   })
   document.querySelector('#clear-button').addEventListener('click', () => {
      clearDashboard(embeddedDashboard.componentInstanceId); // clears embedded dashboard when clicked on some button in
the application
   })

})()
```

# Embedded Visual Authoring JavaScript Example

The following JavaScript example uses methods, properties, and embedded events of the `EmbedManager` class to embed a visual authoring instance.

**Note:** The Composer `EmbedManager` class must be made available to your HTML application prior to using this sample in your application. See Embed Composer Components Using JavaScript And Trusted Access.

**Note:** Composer provides a `componentInstanceId` provides a as part of event details for dashboard, source, and visual events.

```
// NOTE: This example assumes that the EmbedManager has already been initialized
const visualBuilder = await embedManager.createComponent('visual-builder', {
    visualId: <id>, // ID of existing visual
    source: {
        "visualId": "<id>", // ID of visual template, used for creating a new visual, do not pass it with visualId
    },
// If neither visualId is passed (visualId or source.visualId) an empty visual builder will be opened)
    "theme": "composer",
    "header": {
        "visible": true,
        "showTitle": true,
        "showActions": false, // hides the visual actions bar
        "title": "Static custom title"
    },
    "breadcrumbs": { // optional configuration of the breadcrumbs
        "title": "Visuals",
        "onClick": () => {console.log('clicked')},
        "href": "http://www.google.com",
        "target": "_blank"
    },
    interactivityOverrides: { // optional overrides for visual interactivity
        "visualSettings": {
            "FILTER": false,
            "GROUPING": true,
            "METRICS": false, // accept both boolean and string values
            "SETTINGS": false,
            "SORT": true,
            "ZOOM_ACTION": false
        }
    }
});
```

```
visualBuilder.render(document.querySelector('#builder'), { width: '800px', height: '100px' });
```

# Embedded Composer Component Controls

When Composer components are embedded in your applications, the way in which a user can interact with them is determined by settings established by the dashboard, the dashboard visuals, the user's Composer user definition, and the groups and accounts to which that user definition belongs.

Specifically:

- The mode setting of the dashboard controls whether your users can interact with a dashboard at all. If the dashboard mode is set to Read Only, the user will not be able to interact with the dashboard at all. See Generate An Embeddable Dashboard HTML Snippet and Embedded Dashboard Properties And Objects.

  **Note:** The mode setting is deprecated; use dashboard interactivity and visual interactivity settings instead.

- The dashboard permissions for the dashboard determine whether a dashboard is even visible for a user, a group, or a tenant. They also control the permitted level (read, write, or delete) at which the user, group, or account can use the dashboard. See About Dashboard Permissions.

- Data source permissions can restrict the use of data sources by Composer users, groups, or accounts. They also control the permitted level (read, write, or delete) at which the user, group, or account can use the data source. For example, if a dashboard is embedded that the user has permissions to read, write, and delete, but the user does not have permission to use the data source used by a visual on the dashboard, an error appears for the visual for that user when the dashboard is embedded. See About Source Permissions.

- Data source row security can be used to filter the data in the data source used for a dashboard. Row security is applied to specific users, groups, or accounts. If row security is in place, you users will only see the data source data that meets the requirements of the row security filters. For example, your dashboard might show sales for many product categories, but the row security filter for the data source might limit the sales a given user can see to jewelry sales only. See Restrict Access To Data Using Row Security.

- Data source column security can restrict the fields within a data source visible to a given group of users. For example, if the group to which a user is assigned does not have access to the actual sales figures provided in the **Actual Sales** field of a sales dashboard, that field will not be visible on the dashboard for that user group when the dashboard is embedded. See Restrict Access To Fields Using Column Security.

- The interactivity settings of the visuals in the embedded dashboard control many aspects concerning how a visual can be used, including controls for what appears on the visual drop-down menu. For example, you can restrict the ability of your end users to use the context menu for a specific visual on a dashboard. A complete list of the controls available using visual interactivity settings is provided in Control How Users Interact With A Visual.

- The interactivity settings of the embedded dashboard control many aspects concerning how the dashboard and all of its visuals can be used, including controls for what appears on the dashboard icon bar. For example you can restrict the ability of your end users to add or remove visuals from the dashboard. A complete list of the controls available using dashboard interactivity settings is provided in Control How Users Interact With A Dashboard.

- Cross-visual filters can be published or subscribed to. Use them to control how filters interact in a dashboard.

  **Note:** When more than one dashboard is embedded in application, the cross-visual filters that are published for a visual on one of the dashboards can be subscribed to by any visual on any of the embedded dashboards. However, this is not true unless the dashboards are embedded in the same application. Visuals in a dashboard open in one window or tab cannot subscribe to cross-visual filters published by visuals in a different window or tab.

# Integrate Visual Data Into Your Applications

You can integrate Composer visual data into your applications using Composer actions and action templates.

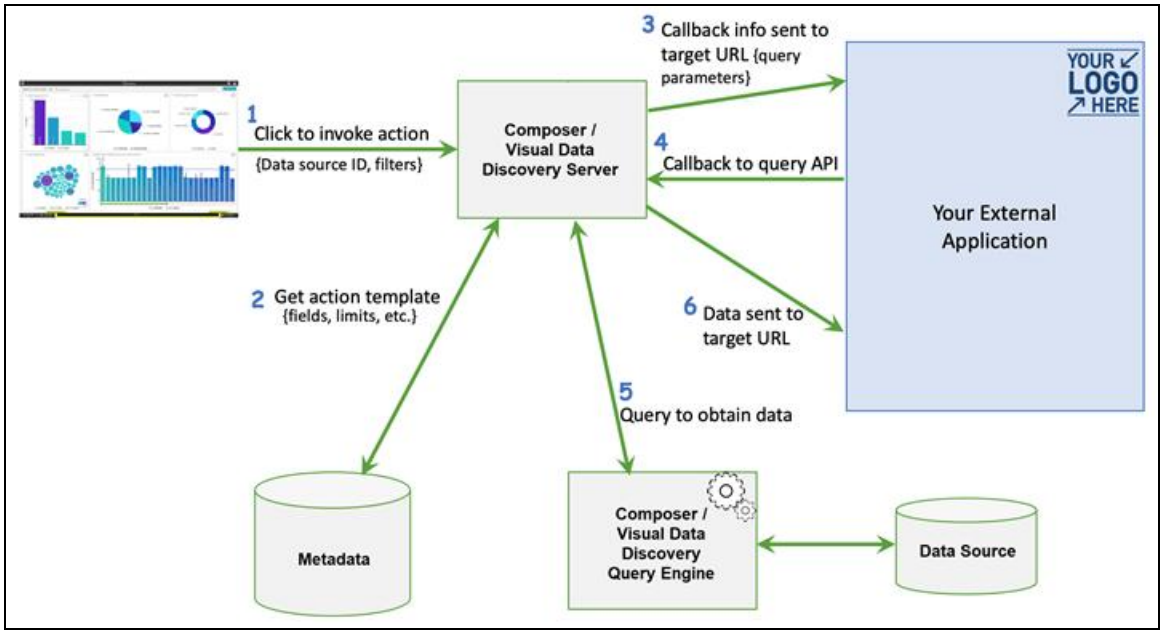
Action templates provide specifics about the external application you want integrated with your data source. Each action template is data source-specific and defines an *application integration definition* that connects Composer with an external application.

After your system and data administrators have defined an action template in Composer and granted proper permissions to use Composer actions, you can invoke an action while working with a visual on your dashboard. The invoked action:

- Creates a query definition based on the filters applied to the visual and on the data and limit specifications in the associated action template.
- Sends the query definition to your application. Your application can use the Composer API to run the query and display or use the data that it collects.

**Note:** You must be logged in as an administrator or as a user with the **Manage Action Templates** privilege to define an action template. You must be logged in as an administrator or as a user with the **Invoke Actions** privilege to invoke an action.

The following diagram depicts action processing.




See the following topics:

- Define An Action Template
- Modify An Action Template
- Enable And Disable An Action Template
- Delete An Action Template
- Invoke An Action

# Define an Action Template

Use action templates to provide specifics about the external application you want integrated with your data source. Each action template defines an application integration definition that connects Composer with an external application.

**Note:** You must be logged in as an administrator or as a user with the **Manage Action Templates** privilege.

# Define a New Action Template

1. Log in as a n administrator or a user with the **Manage Action Templates** privilege.
2. Select **Actions** on the UI menu ( ). The Actions page appears.

   The Actions page is split into two parts. Action templates (if any) you have defined are listed on the left. When you select an action template on the left, its details appear on the right.
3. Select the add icon . Action properties are listed on the right side of the page.
4. Specify appropriate action properties on the right side of the page, as described below.

| Action Property | Description |
|---|---|
| Name | Specify a unique name for the action template definition. |
| Target URL | Specify the URL of the external application for the action template. Specify the URL of the external application for the action template. The supplied context variables `${User.composerUserName}` and `${User.accountId}` can be used in the URL to insert the name or account ID of the user that is currently invoking the action. |
| Data Source | Select the Composer data source configuration for the action template. |
| Fields | Select (check) fields in the data source configuration for which data should collected by the action template when it is invoked. |
| Row Limit | Specify a positive integer ranging from 1 through 999999999 to limit the number of rows of data submitted to your application when the action template is invoked. |
| Enable Action | Slide the **Enable Action** selector to the left or right to disable or enable the action template. See Enable And Disable An Action Template. |

5. Select **Save**.

# Modify an Action Template

## Modify an Existing Action Template

1. Log in as an administrator or a user with the **Manage Action Templates** privilege.
2. Select **Actions** on the UI menu (  ). The Actions page appears.

   The Actions page is split into two parts. Action templates (if any) you have defined are listed on the left. When you select an action template on the left, its details appear on the right.
3. Locate and select the action template you want to modify in the list on the left.

   You can locate an action template using the search bar on the top left side of the Actions page. You can also filter the action template by the data source to which they apply using the drop-down list on the top left side of the Actions page. When you do this, only action templates for the selected data source configuration are shown in the list.

   The properties for the selected action template appear on the right side of the Actions page.
4. Modify action properties on the right side of the page, as needed.

| Action Property | Description |
|---|---|
| Name | Specify a unique name for the action template definition. |
| Target URL | Specify the URL of the external application for the action template. |
| Data Source | Select the Composer data source configuration for the action template. |
| Fields | Select (check) fields in the data source configuration for which data should collected by the action template when it is invoked. |
| Row Limit | Specify a positive integer ranging from 1 through 999999999 to limit the number of rows of data submitted to your application when the action template is invoked. |
| Enable Action | Slide the **Enable Action** selector to the left or right to disable or enable the action template. See Enable And Disable An Action Template. |

5. Select **Save**.

# Enable and Disable an Action Template

Only enabled action templates can be invoked from Composer. You can enable and disable action templates using the Composer UI.

**Note:** You must be logged in as an administrator or as a user with the **Manage Action Templates** privilege.

## Enable an Action Template

1. Log in as an administrator or a user with the **Manage Action Templates** privilege.
2. Select **Actions** on the UI menu ( ). The Actions page appears.

   The Actions page is split into two parts. Action templates (if any) you have defined are listed on the left. When you select an action template on the left, its details appear on the right.
3. Locate and select the action template you want to enable in the list on the left.

   You can locate an action template using the search bar on the top left side of the Actions page. You can also filter the action template by the data source to which they apply using the drop-down list on the top left side of the Actions page. When you do this, only action templates for the selected data source configuration are shown in the list.
4. On the right side of the Actions page, slide the **Enable Action** selector to the right to enable the action template.
5. Select **Save**.

## Disable an Action Template

1. Log in as an administrator or a user with the **Manage Action Templates** privilege.
2. Select **Actions** on the UI menu ( ). The Actions page appears.

   The Actions page is split into two parts. Action templates (if any) you have defined are listed on the left. When you select an action template on the left, its details appear on the right.
3. Locate and select the action template you want to disable in the list on the left.

You can locate an action template using the search bar on the top left side of the Actions page. You can also filter the action template by the data source to which they apply using the drop-down list on the top left side of the Actions page. When you do this, only action templates for the selected data source configuration are shown in the list.

4. On the right side of the Actions page, slide the **Enable Action** selector to the left to disable the action template.

5. Select **Save**.

# Delete an Action Template

1. Log in as an administrator or a user with the **Manage Action Templates** privilege.
2. Select **Actions** on the UI menu ( ). The Actions page appears.

   The Actions page is split into two parts. Action templates (if any) you have defined are listed on the left. When you select an action template on the left, its details appear on the right.
3. Locate the action template you want to delete in the list on the left.

   You can locate an action template using the search bar on the top left side of the Actions page. You can also filter the action template by the data source to which they apply using the drop-down list on the top left side of the Actions page. When you do this, only action templates for the selected data source configuration are shown in the list.
4. Select the delete icon next to the action template you want to delete on the left side of the Actions page.
5. Select **Delete** to delete the action template.

# Invoke an Action

If an action template has been defined and enabled for a data source configuration, the associated action can be invoked from a visual on your dashboard that uses the data source. The invoked action:

- Creates a query definition based on the filters applied to the visual and on the data and limit specifications in the associated action template.
- Sends the query definition to your application. Your application can use the Composer API to run the query and display or use the data that it collects.

> **Note:** You must be logged in as an administrator or as a user with the **Invoke Actions** privilege. In addition, the associated action templates must be enabled to be invoked from your visuals.

To control whether actions can be invoked on a visual, use the interactivity sidebar. See Control How Users Interact With A Visual.

# Invoke an Action from a Visual

1. Log in as an administrator or a user with the **Invoke Actions** privilege.
2. On the Home page, select the dashboard containing a visual that uses the data source for which you have defined and enabled one or more action templates.
3. Filter or group the visual data as needed.

   For example, if you only want to send sales data from the state of Virginia to your external application, filter your sales data visual by the state of Virginia.
4. Select an area of the visual to display the context menu and select **Actions**. A list of the action templates defined for the visual's data source appears.

   Alternatively, select **Actions** from the visual drop-down menu.

   > **Note:** If the **Actions** menu option does not appear on the context menu or the visual drop-down menu, an action template is either not defined or is not enabled for the data source used by the visual.
5. Select an action template from the list. The action is invoked. A query is created and sent to your application. Use the Composer API endpoint `api/action` in your application to run the query.

   API documentation is provided with your Composer installation at this link: *`https://<composer-URL>/composer/swagger-ui.html`*.

**Important:** Some API endpoints are marked as `experimental` in the Swagger documentation we provide. These endpoints are in the early stages of design and are subject to change. We make no commitment to their stability and may remove them without notice. These experimental endpoints are not recommended for use in production.

# Configure Visuals Using Visualization Variables

When a Composer visual is embedded into a custom application, it uses the default settings determined by the metrics and fields or groups present in the data query that supplies the visual with data. For more information about configuring and creating a data query, see Query Configuration Object or Use A Data Query.

When embedding a visual, you also have the option of overriding default configurations by modifying the configuration of visual settings. You modify the default settings with the `variables` key in the parameter passed to the `visualize()` method. Before you can encode settings into an embedded visual, you must identify every required setting as well as any optional settings that you wish to use. Visual settings can be identified using an internal REST API method. The REST API method used in these steps is considered internal to the Composer application. Internal APIs should not be used except as directed. For more information about using internal APIs, see Cautionary Note About Internal APIs.

# Identify a Visual's `visualizationID`

You can identify a Composer visual's `visualizationID` while working with it. For example, to programmatically identify a visual's default settings using a REST API call, you must have the `visualizationID` handy.

**Identify a visual's `visualizationID`**

1. Open the visual in its dashboard.
2. Find the number at the end of the URL in the address bar of your browser. The `visualizationID` is the section of the number after the plus sign (+).

After you have identified the `visualizationID` of a visual, you can use it in other steps such as calling REST API methods.

# Identify a Visual's `sourceID`

You can identify a Composer visual's `sourceID` while working with it. For example, to programmatically identify a visual's default settings using a REST API call, it is easiest to use the `sourceID` in the call to look at its source configuration.

**To identify the `sourceID` of a visual's source:**

1. Open the visual in its dashboard.
2. Find the number at the end of the URL in the address bar of your browser. The `sourceID` is the section of the number before the plus sign (+).

After you have identified the `sourceID` of a visual, you can use it in other steps such as calling REST API methods.