SisenseJS

Sisense.js is a JavaScript library that enables you to embed Sisense components in web pages without the use of iFrames.

By embedding the SisenseJS library, you can:

- Load Sisense runtime anywhere, without iFrames
- Load dashboards in runtime
- Render all/part/new widgets in any DOM container
- Embed Sisense visualizations into your mobile applications

This page describes the SisenseJS library's functionality and how you can leverage it to embed widgets into your site or application.

Limitations

SisenseJS includes the following limitations:

- Sisense.js overrides the "Date" prototype with the "toISOString" function.

Dependencies

SisenseJS has the following dependencies

- Highcharts
- D3
- JQuery

Overview

A widget is a dynamic visualization of your data with its own unique ID. Through the SisenseJS library, you define which widgets appear in the dashboard. In Sisense, a dashboard is a collection of one or more widgets that visualize the data that you select and design. However, in SisenseJS, the dashboard represents a JavaScript object that you can embed into your site or application.

The dashboard object is a container for widget objects. As a container of dashboards, and not a dashboard itself, it provides you with more flexibility by allowing you to add existing widgets from various dashboards to the dashboard object you embed on your site. After including the SisenseJS library into your site or application, you embed dashboard objects and populate those objects with your widgets.

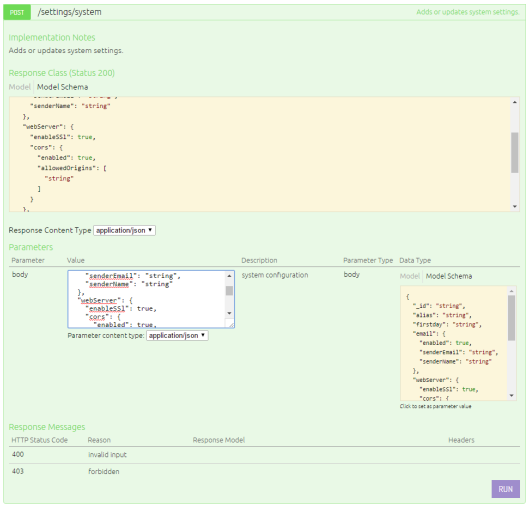
Prerequisites

CORS

As the application in which you are embedding Sisense elements most likely resides on a different domain than where Sisense is installed, you must enable CORS (Cross-Origin Resource Sharing) to access the JavaScript resources needed. For more information, see Cross Origin Resource Sharing.

To enable CORS:

1. From the Sisense REST API, send a POST request to the settings/system API.



2. In the CORS object, set enabled to **true**.
3. In the allowedOrigins array, enter every domain from which you might make a request to Sisense.

Enabling all origins

```
{
 "webServer": {
   "enableSSl": false,
   "cors": {
     "enabled": true,
     "allowedOrigins": [
       "*"
     ]
   }
 }
}
```

4. Click **Run**.

Read more about the settings API here | Read more about CORS here

Including the Sisense.js Library

You must include the sisense.js runtime file in your page as defined below where `localhost` should be your Sisense installation's URL and port.

```
 <script type="text/javascript" src="http://localhost/js/sisense.js"><
/script>
```

AngularJS Support in SisenseJS

In Sisense version 6.4 and later, Sisense.js supports Angular 2.x.

In Sisense version 6.5 and later, Sisense.js supports Angular 1.x and 2.x.

Angular 2.x heavily emphasizes loading templates asynchronously, which means some HTML content is loaded until later on. For Sisense.js to work, this HTML content should be loaded prior to connecting to the Sisense server through the sisense.connect function.

Sisense.js bootstraps a div element with the ID "sisenseApp" to ensure the sisenseApp element and other container elements are loaded before using the sisense.connect function.

Encapsulation

```
<body>
        <div id="sisenseApp">
                Code...
       </div>
```

Connecting to Sisense

After you have included the Sisense.js library in your site, you must connect to Sisense.

The connect(url, saveChanges) method is a static method of the Sisense.js library that takes the URL of your dashboard as an argument, and supports an optional second argument for persisting changes done by the user.

The then() method returns the app object as a promise. The app objects contains your dashboards and defines how your widgets are embedded.

```
  Sisense.connect('http://localhost:8081', false).then(function(app) {
    // your code here
});
```

Embedding Dashboards

There are two ways to embed dashboards through the SisenseJS library. You can create an empty dashboard object in which you add existing widgets to the dashboard object. The second way is to embed an existing dashboard where you retrieve the existing dashboard with its filters and embed them into the page.

Retrieving Dashboard and Widget IDs

Regardless of how you embed your dashboard, you will need to provide the IDs of dashboards and widgets. These IDs are visible in the URL when you are viewing a dashboard or widget.

To view the full URL structure, open the relevant widget in Edit mode:

http://localhost/app/main#/dashboards/ {dashboard_id} /widgets/ {widget_id}

Example:

http://localhost/app/main#/dashboards/ 574eb8cd92be9b504b000006 /widgets/ 574eb8d892be9b504b000009

Creating a New Dashboard

To create a new dashboard, you construct a dashboard object, which is an empty container for widgets. After constructing the dashboard object, you can load existing widgets that you have defined in the Sisense Web Application to your new dashboard.

You can create an empty dashboard object through the Dashboard() object constructor:

```
var myEmptyDashboard = new Dashboard();
```

To embed the new dashboard object, add it to your site through the add() method:

```
app.dashboards.add(myEmptyDashboard);
```

Once you have added your dashboard to the site, you can add existing widgets to the dashboard object using the load() method:

```
myEmptyDashboard.widgets.load('570a1cd5d814c6245b000014').then(function
(w) {
    //code
}
```

When you have added all your widgets to the dashboard object, you need to render it to display the widgets in your site through the refresh() method:

```
myEmptyDashboard.refresh();
```

Embedding an Existing Dashboard

If you already have an existing dashboard and you want to embed it, you can load the existing dashboard through the load() method. The load() method takes the dashboard ID as an argument.

```
app.dashboards.load('574603d5fc726c3430000037').then(function(dash) {
```

After loading the dashboard, all of your existing widgets are loaded into memory. You can use the get() method to retrieve and display the widget.

```
dash.widgets.get('5746043ffc726c3430000043').container = document.
getElementById("widget1");
dash.widgets.get('3563564603ffc726c34300918').container = document.
getElementById("widget2");
```

For each widget, you must have a div element with an ID, for example, "widget1" in your page. This DIV that contains your widget must have the height, width, and position defined for the widget to render correctly.

You can also use libraries like jQuery to make DOM manipulation and querying easier.

If you want to add widgets that are not associated with the existing dashboard, you can load the widgets from the server through the load() method.

```
dash.widgets.load('570a1cd5d814c6245b000014').then(function(w) {
    //code
}
```

When you have added all your widgets to the dashboard object, you need to render it to display the widgets in your site through the refresh() method:

```
dash.refresh();
```

Adding Filters

After you have loaded a dashboard, you can load its filters through the renderFilters() method.

Note : A dashboard object can only contain filters if it's datasource property is set. When loading existing dashboards, they are always loaded with a datasource. When creating a new object, you must set it yourself.

```
dash.renderFilters(document.getElementById("filters"));
```

SSO and SisenseJS

When embedding Sisense.JS into your site, you should implement Single Sign On. Sisense implements SSO by redirecting users to a script /application that creates a login token for the user, and redirects them back to the destination or resource they were trying to access. If the resource requested is a dashboard, that's where the user will be redirected back to. For Sisense.JS the resource requested will be a JavaScript file, so you need to ensure the redirect takes you back to this file. When Sisense redirects your customers to your login script, Sisense passes a return_to parameter in the URL. This parameter defines the page that Sisense redirects your customer after authenticating them. For example:

1. A customer visits your site opens a dashboard embedded through an iFrame.
2. Sisense recognizes that the user is not authenticated.
3. Sisense redirects the user to:

```
https://yourcompany.com/sisense/sso?return_to=https://yourcompany.sisense.
com/dashboards/
```

All your script needs to do, is take the return_to value from the invoked URL and pass it back to Sisense when submitting the JWT token. In other words, upon authentication on your side, your script redirects the user to:

```
https://yourcompany.com/access/jwt?
jwt=payload&return_to=https://yourcompany.sisense.com/dashboards/
```

For more information regarding SSO and Sisense, click here.

SisenseJS Example

The following is an example of an HTML page that has implemented SisenseJS to display a dashboard with three widgets inside.

The first step is to load the SisenseJS library. Then, call the connect method on the Sisense object to connect to your dashboard.

Once you have established a connection, load the dashboard object and its widgets. When loading the dashboard and widget objects, replace the IDs in the example below with your Dashboard and Widget IDs.

After the dashboard and widget objects have been loaded, their filters are also displayed with the renderFilters method.

The data for each widget is then refreshed through the refresh method.

<!DOCTYPE HTML>

```
<html>
   <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <script type="text/javascript" src="http://localhost:8081/js/sisense.js"></script> <!-- replace with your Sisense
server address -->
    <script type = "text/javascript">
```

```
        Sisense.connect('http://localhost:8081').then(function(app) { // replace with your Sisense server
address
          app.dashboards.load('5829b46b3452420000047').then(function(dash){ //replace with your dashboard
id
             dash.widgets.get('5dgb3509e914200001265').container = document.getElementById("widget1");
//replace with one of your widgets' id.
             dash.widgets.get('5ghbdg456446yh00001hb').container = document.getElementById("widget2");
//replace with another of your widgets' id.
             dash.widgets.get('58b45b4ce6ef16b0b4000042').container = document.getElementById("widget3");
//replace with another of your widgets' id.

             dash.renderFilters(document.getElementById("filters"));
             dash.refresh();
          });
         });
      </script>
    </head>
    <body>
       <div id = "sisenseApp">
            <div class = "Tab1" id = "widget1" style = "height: 400px; width: 35%; float: left; top:
500px; display: inline; margin-top:30px;"></div>
            <div class = "Tab1" id = "widget2" style = "height: 400px; width: 40%; float: left; top:
500px; display: inline; margin-top:30px;"></div>
            <div id = "filters" style = "height: 400px; width: 25%; float: left; top: 80px; left: 0px;
display: inline; margin-top:80px;"></div>
            <div class = "Tab1" id = "widget3" style = "height: 400px; width: 100%; float: left; top:
500px; display: block; margin-top:30px;"></div>
       </div>
   </body>
</html>
```

SisenseJS Reference

Sisense

Methods

Property	Arguments	Returns	Description
`connect( url, saveChanges)`	**url**: A string value of the Sisense server URL that you want to connect to. **saveChanges**: (Optional) Default value is false. This argument is a boolean that when **true**, Sisense sav es the user's changes in the Sisense MongoDB. When **false**, changes are not saved.	Promise	Connects the application to a Sisense server. This is the initial step in any SisenseJS application, providing your code with a Sisense context to run in and telling it where to retrieve dashboards and widgets from.

Application

Properties

Property	Type	Description
`serverUrl`	String	Contains the full URL for the Sisense instance.

<table>
<tr>
<td><code>dashboards</code></td>
<td>Object</td>
<td>Entry point for accessing Sisense dashboards, representing a collection of the dashboard objects currently loaded to memory.

Contains the following methods :

<table>
<tr><th>Method</th><th>Arguments</th><th>Returns</th><th>Description</th></tr>
<tr><td><code>add()</code></td><td></td><td></td><td>Adds the dashboard object to an enumerable collection on the page which can be then retrieved by numerical index using get().</td></tr>
<tr><td><code>get(idx)</code></td><td>Idx : An int value that represents the index of a dashboard to get from the collection.</td><td></td><td>Gets a dashboard object that was already loaded into memory (using either add() or load()).</td></tr>
<tr><td><code>load(dashboard_id)</code></td><td>dashboard_id: A string value that is the ID of the dashboard to be loaded.</td><td>Promise</td><td>Loads an existing dashboard by Id.</td></tr>
</table>
</td>
</tr>
</table>

Dashboard

Properties

Property	Type	Description
`datasource`	Object	Contains an object representing the dashboard's data source.
`id`	String	Contains the full dashboard ID.
`refreshing`	Boolean	True when the dashboard is currently refreshing, or false when it is not.

<table>
<tr>
<td><code>widgets</code></td>
<td>Object</td>
<td>Entry point for accessing a dashboard's widgets, representing a collection of the widget objects currently loaded to memory for a given dashboard object.

Contains the following methods:

<table>
<tr><th>Method</th><th>Arguments</th><th>Returns</th><th>Description</th></tr>
<tr><td><code>get()</code></td><td>oid : String value that is the ID of the widget to get from the collection.</td><td></td><td>Gets a widget object that was already loaded into memory (using either add() or load()).</td></tr>
<tr><td><code>load(dashboard_id)</code></td><td>dashboard_id : A string value that is the ID of the dashboard to be loaded.</td><td>Promise</td><td>Loads an existing dashboard by ID.</td></tr>
</table>
</td>
</tr>
</table>

Methods

Method	Arguments	Returns	Description
`on( event, callback )`	**event**: A string value of the event name. **callback**: The callback function to execute when the event is triggered.		R egisters dashboard events. For more information, see the D ashboard API reference.
`refresh()`			Refreshes the dashboard updating all embedded widgets.
`renderFilter(element)`	**element** : The DOM element that acts as a container for rendering the filters panel.		Renders the dashboard's filters panel for the element provided.

Widget

Properties

Property	Type	Description
`metadata`	Object	Contains an object representing the widget's metadata.
`type`	String	Contains the widget's primary type.
`subtype`	String	Contains the widget's sub-type.
`affectDashboardFilters`	Boolean	Contains `true` if the widget is a selector (clicking it will create a filter), `false` otherwise.
`refreshing`	Boolean	Contains `true` if widget is currently refreshing, `false` otherwise.
`container`	DOM Element	Contains the DOM element where the widget is rendered.

`id`	String	Contains the widget's ID.
`queryResult`	Object	Contains the data returned for the widget's query from Sisense.
`title`	String	Contains the widget's title.

Methods

Method	Arguments	Returns	Description
`on()`	**event** : A string value of the event name. **callback** : The callback function to execute when the event is triggered.		Registers any widget events. For more information, see the W idget API reference.
`refresh()`			Refreshes the widget.

`getDrillItems()`		[itemsForDrill]	Returns an array of items available for drill down.
			
			Items are arrays that contain the following information:
			
			{
			
			column:"Category ID"
			
			dimtype:"numeric"
			
			id:"[Commerce.Category ID]"
			
			indexed:false
			
			merged:true
			
			table:"Commerce"
			
			title:"Category ID"
			
			type:"dimension"
			
			}
			
			Note: This method is available in Sisense V6.7 and later.

`isDrillSupported()`		{
		`supported: boolean`
		`error: string`
		}`
			Checks if the widget supports drill downs and returns an error if not.
			
			Note: This method is available in Sis ense V6.7 and later.

`loadDrillItems()`		promise	Loads items available for drill down. This method should be run for a widget once and before running any other methods. **Note**: This method is available in Sis ense V6.7 and later.
`loadDrillSuggestions()`		promise	Loads suggested items for drill down. **Note**: This method is available in Sis ense V6.7 and later.
`performDrill()`			Performs a drill down. **Note**: This method is available in Sis ense V6.7 and later.

`resetDrill()`			Resets a drill down. **Note**: This method is available in Sis ense V6.7 and later.