TOC

Logi Symphony 24 ..... **3**

Symphony Managed Dashboards / Managed Report Embedding 3

Quick Start Reference - Embedding Managed Dashboards and Managed Reports **4**

Embedding 4

Cross-Origin Resource Sharing 4

Using Cross-Origin Resource Sharing **6**

Configure Cross-Origin Resource Sharing (CORS) **7**

Pass Your Application's JavaScript to the Client Application Example **9**

Create a New Dashboard 9

Add the CORS JavaScript 10

Test Your Sample 11

Other CORS JavaScript Examples **12**

Change View Options and Return Data 12

Change View Options and Return Data Asynchronously 12

REST API with CORS Example **14**

Troubleshooting CORS Issues **15**

Using the Managed Embed Library **16**

Downloading the Embed Library **17**

Embedding Without JavaScript **18**

User Log On Options 20

Embedding With JavaScript **21**

Get a Logon Token 21

Creating an Embedded Application Object **24**

Using the Embedded Application **26**

Application Properties 26

Embedded Application Example 28

Embedded Methods **31**

Working With View Parameters **35**

Token Helpers 36

Basic 36

SingleDate 36

DateRange 37

URL Path Length Restrictions **38**

Host Embed Library File **40**

Logi Symphony 24

Symphony Managed Dashboards / Managed Report Embedding

Quick Start Reference - Embedding Managed Dashboards and Managed Reports

This applies to: *Managed Dashboards, Managed Reports*

Embedding

- Using the Managed Embed Library
- Downloading the Embed Library
- Embedding Without JavaScript
- Embedding With JavaScript
- Creating an Embedded Application Object
- Using the Embedded Application
- Embedded Methods
- Working With View Parameters
- URL Path Length Restrictions
- Host Embed Library File

Cross-Origin Resource Sharing

- Using Cross-Origin Resource Sharing
- Configure Cross-Origin Resource Sharing (CORS)
- Pass Your Application's JavaScript to the Client Application Example

- Other CORS JavaScript Examples
- REST API with CORS Example
- Troubleshooting CORS Issues

Using Cross-Origin Resource Sharing

This applies to: *Managed Dashboards, Managed Reports*

This section explains how to enable and use Cross-Origin Resource Sharing (CORS) in Symphony. This allows your web application's JavaScript to communicate with Symphony's APIs or whenSymphony is embedded even if your application and Symphony are hosted on different domains.

Important: Browser support for embedding HTTP cross-domain content is changing, and Symphony should now either run on HTTPS or have the same URL origin (protocol, hostname/domain, and port number) as the parent application.

Cross-origin resource sharing must be configured in Symphony to enable either of these two types of communication with Symphony from your own client-side application's JavaScript:

- Communication with the Symphony server via its API, if the URLs for your application and Symphony have different domains or origins.
- Communication with the Symphony client application embedded on your page in an iFrame or using the embed library, even if both applications have the same domain/origin.

For more information on using CORS, see the following topics:

- Configure Cross-Origin Resource Sharing (CORS)
- Pass Your Application's JavaScript To The Client Application Example
- Other CORS JavaScript Examples
- REST API With CORS Example
- Troubleshooting CORS Issues

Configure Cross-Origin Resource Sharing (CORS)

This applies to: *Managed Dashboards, Managed Reports*

CORS must be enabled for your site through the configuration settings if you want to perform certain embed library functions using your own JavaScript on a page that embeds managed dashboards and reports.

These include listening for the client-side `ready` event or passing JavaScript to run within Symphony in the browser from your own page, as listed in the methods table.

Set up CORS even if both your page's URL and Symphony's URL have the same origin (protocol, hostname/domain, and port number) if you want to use the embed library's methods.

CORS also authorizes your page to make REST calls to the Symphony server from the browser. See REST API with CORS Example.

Important: Browser support for embedding HTTP cross-domain content is changing, and Symphony should now either run on HTTPS or have the same URL origin (protocol, hostname/domain, and port number) as the parent application.

Configure cross-origin resource sharing

1. While logged in as a member of the system administrator's group, navigate to and select the **Admin** section of Symphony in the main menu. The Admin menu opens.
2. Select **Setup** to expand the menu and see the setup options, then select **Config**. A configuration settings work opens.
3. From the options presented, select **Advanced Settings** , then select and edit **CORS (Cross-Domain) Origins** in the **Web Application** category.
4. Select **Edit** value, then enter the URL origin of the page that will embed Symphony, such as `https://domain.com` or `http://hostname:8000`.

   Note: To restrict which sites can embed this Symphony instance on its pages, use the **Allowed Embedded Origins** configuration setting. See also the other configuration best practices.
5. Select **Submit** to apply your changes.

   If needed, use the in-dialog guidance to add multiple domains.

   Note: Do not include a slash at the end or any part of the path when setting an origin.

Defining the URLs here lets Symphony know:

- The origin/domain you specify is yours and is authorized to access the Symphony REST API in the browser from that origin if different than Symphony's.
- Your page's JavaScript from any origin (even if the same as Symphony) is yours and authorized to pass JavaScript to run in the Symphonyembedded client application or monitor for when its page is ready.

Any other site not listed or included is prevented by browsers and by Symphony from performing these functions. Additionally, this prevents:

- Access to Symphony APIs.
- Passing of messages to embedded applications.
- Execution of scripts from posted messages unless the embedding application's origin is listed or included, even if the same as Symphony.

Pass Your Application's JavaScript to the Client Application Example

This applies to: *Managed Dashboards, Managed Reports*

The following is an example of cross-origin resource sharing, where your own application's JavaScript passes JavaScript to run in Symphony's client application embedded within yours in the browser.

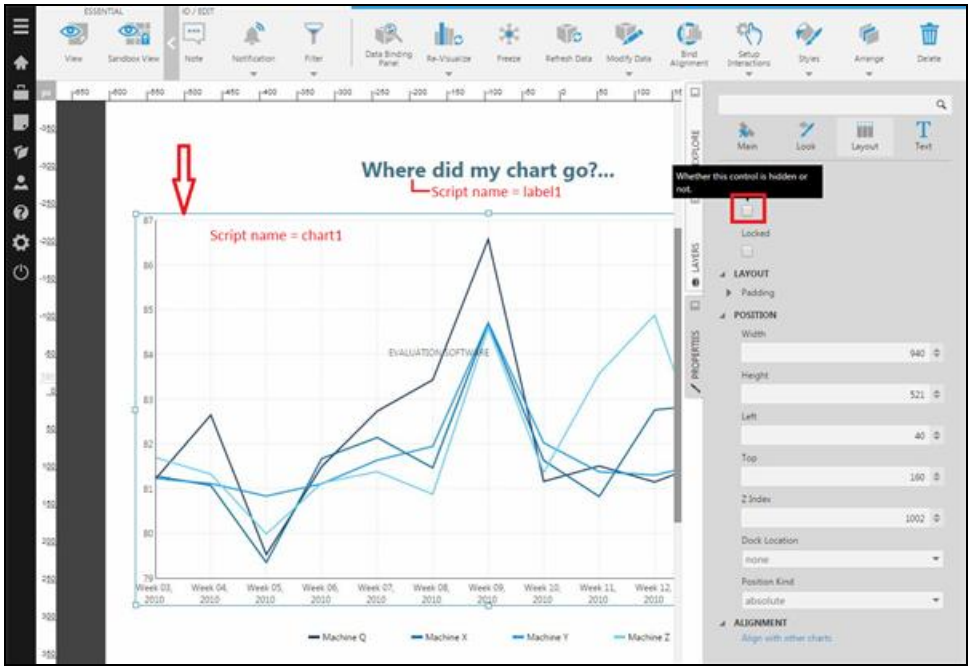
For the examples in this section where Symphony is embedded on the page, it is embedded using an inline frame (iFrame). You can also use the embed library and use its `runScript` method. See the viewer integration sample.

Create a New Dashboard

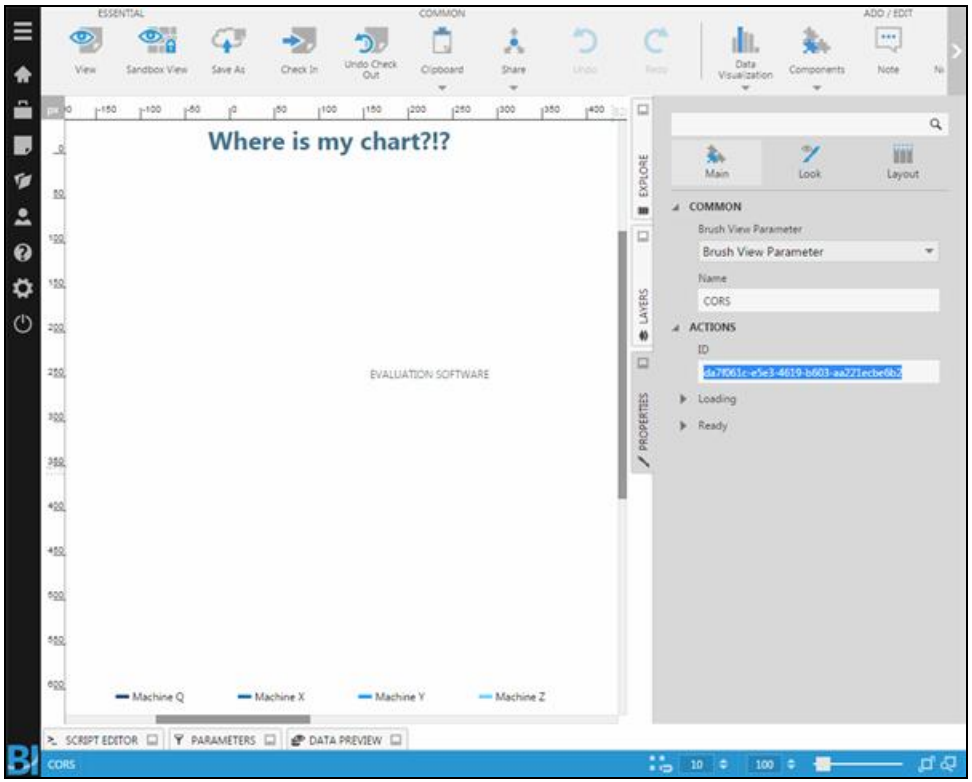
On a new dashboard, add a chart (with Script Name chart1) and a label (with the Script Name label1).

For this example, we will be altering the text in the label and unhiding the chart. This will be done by sending a script from the host site into the iFrame containing Symphony.

Select the **Hidden** property of the chart, which is located in the **Properties** window in the **Layout** tab.



In the Dashboard's properties, copy the dashboard ID from the **Main** tab.



Now check in the dashboard, and use as the file ID in your web application that embeds Symphony.

Add the CORS JavaScript

JavaScript like the following could be added to your application/page, which calls postMessage. This particular example assumes a jQuery library reference is included on the page:

```
$(document).ready(function () {
        // Synchronous Example
        var element = $("iframe").get(0);
        $(element).load(function () {
                // Since the iframe will have loaded before the dashboard has,
                // we need to pass in a function that runs when the dashboard is ready.
                element.contentWindow.postMessage({
                        "isAsync": false,
```

```
                        "script": "\n\
                                window.dundas.context.ready(function () {\n\
                                        var myView = dundas.context.baseViewService.currentView;\n\
                                        var myLabel = myView.getAdapters().filter(function(a) { return a.name == \"label1\" })[0];\n\
                                        var myChart = myView.getAdapters().filter(function(a) { return a.name == \"chart1\" })[0];\n\
                                        myChart.hidden = false;\n\
                                        myLabel.labelText = \"There it is!!!\";\n\
                                });\n\
                        "
                }, "https://dundas.mysite.com");
        });
});
```

The script sent to the embedded Symphony client application waits for the dashboard to load, then finds and modifies the properties of the chart and label.

The `\n\` is required to include the line breaks in the string.

Test Your Sample

Build and publish your project's server-side changes (if applicable).

You should now see the new label text and the chart should be visible.

Other CORS JavaScript Examples

This applies to: *Managed Dashboards, Managed Reports*

For the examples in this section where Symphony is embedded on the page, it is embedded using an inline frame (iFrame). You can also use the embed library and use its `runScript` method. See the viewer integration sample.

Change View Options and Return Data

The following is another example of running JavaScript within the embedded Symphony client application from your own application's JavaScript. It sets the view options of a dashboard to hide the menu, toolbar and taskbar, then sends a message back to the parent application.

```
$(document).ready(function() {

	// Hook up to receive window postMessage calls.
	$(window).bind("message", function (e) {
		var result = e.originalEvent.data;
		// result will be 'hi'
	});

	// Synchronous Example
		var iframe = $("#iFrame").get(0);
		$(iframe).load(function() {
			iframe.contentWindow.postMessage({
				"isAsync": false,
				"script": " console.log(e);
				window.dundas.context.baseViewService.viewOptions = dundas.ViewOptions.VIEW_ONLY;
				window.dundas.context.updateViewFromViewOptions();
				return 'hi';
				"
			}, "https://dundas.mysite.com");
	});
```

Change View Options and Return Data Asynchronously

The following example is the same as the previous but sends a message back to the parent application asynchronously (when a timeout expires, for demonstration purposes). The `isAsync` property is set to `true` in the posted message object, in which case the accompanying script should return a jQuery Promise or Deferred as done below:

```
$(document).ready(function() {

	// Hook up to receive window postMessage calls.
	$(window).bind("message", function (e) {
		var result = e.originalEvent.data;
		// result will be 'hi'
	});

	// Synchronous Example
	var iframe = $("#iFrame").get(0);
	$(iframe).load(function() {
		iframe.contentWindow.postMessage({
			"isAsync": false,
			"script": " console.log(e);
			window.dundas.context.baseViewService.viewOptions = dundas.ViewOptions.VIEW_ONLY;
			window.dundas.context.updateViewFromViewOptions();
			return 'hi';
			"
		}, "https://dundas.mysite.com");
	});
});
```

REST API with CORS Example

This applies to: *Managed Dashboards, Managed Reports*

For the examples in this section where Symphony is embedded on the page, it is embedded using an inline frame (iFrame). You can also use the embed library and use its `runScript` method. See the viewer integration sample.

The following example sends a /Dashboard/ request to the Symphony server from your own application's JavaScript. In this case, there may not be any embedded application on the page.

```
$(document).ready(function() {

        // REST API example
        var def = $.ajax("https://dundas.mysite.com/api/dashboard/[dashboardId]",
                        { headers: { "Authorization": "Bearer " + sessionId } }
        );
        def.done(function (dashboard) {
                // perform action here.
        });
        // Note that this method returns a plain object rather than a full Dundas class instance
});
```

Troubleshooting CORS Issues

This applies to: *Managed Dashboards, Managed Reports*

Use your browser's developer tools, and check the console for errors.

If you are posting a message as in examples in this section or using the embed library to run script:

- If everything went well, in the console you should see a message such as:

  Running window message received from origin '[domain]' which was in the accepted list. If this was not expected, this might be an XSS attack.

- If you see a message such as below, there was probably a problem with the CORS application configuration setting you entered:

  Attempt to post message from origin '[domain]', but this is not in the allowed list from configuration. Post ignored.

Using the Managed Embed Library

This applies to: *Managed Dashboards, Managed Reports*

The managed embed library is designed to make it easy to add any part of Symphony to existing web pages and web applications. This is a guide to using the Symphony embed library, which can be done in two different ways:

- Embedding without writing JavaScript – Use only HTML markup, which can be simpler and easier to maintain.
- Embedding with JavaScript – This allows for more advanced functionality such as running scripts and hooking to events in the Symphony application.

For more information, see:

- Downloading The Embed Library
- Embedding Without JavaScript
- Embedding With JavaScript
- Host Embed Library File

Downloading the Embed Library

This applies to: *Managed Dashboards, Managed Reports*

The Symphony managed embed library is a single JavaScript file, and you reference it with an HTML script tag.

The file can be found at the following location:

```
[InstanceDirectory]\www\BIWebsite\wwwroot\Scripts\Embed\dundas.embed.min.js
```

You can place it in your own website or refer to it from this location. Update the path in the example below depending on where the file is relative to the page:

```
<head>
       <script src="/Scripts/Embed/dundas.embed.min.js" type="text/javascript"></script>
</head>
```

Important: Browser support for embedding HTTP cross-domain content is changing, and Symphony should now either run on HTTPS or have the same URL origin (protocol, hostname/domain, and port number) as the parent application.

Embedding Without JavaScript

This applies to: *Managed Dashboards, Managed Reports*

This method is a quick and simple way of embedding Symphony that doesn't require any JavaScript, and uses the embed library as an alternative to creating an iFrame element yourself and setting its URL. Add this HTML markup to your page or generate it in your server-based web application with the required information specified as attributes.

The following example embeds a Symphony instance located at `https://placeholder.dundas-bi-url.com`, and a dashboard with the specified ID:

```
   ...
<head>
       <script src="/Scripts/Embed/dundas.embed.min.js" type="text/javascript" ></script>
</head>
       ...

<body>
       <div class="dundas-bi-embed"
              data-dundas-bi-url="https://placeholder.dundas-bi-url.com/"
              data-controller-type="Dashboard"
              data-file-system-id="727ae6b6-b5b6-4c1e-bcd7-72561000098d">
       </div>
</body>
```

Note: This element must be added to the page before the browser's DOMContentLoaded event fires to embed Symphony without JavaScript.

The following table describes the HTML attributes that can be used when embedding without JavaScript:

Attribute	Description
`class`	The class must be set to `dundas-bi-embed` to become a valid Symphony embedded application object.
`data-dundas-bi-url`	The base URL pointing to the installed Symphony instance. For example: `https://sym.example.com:8000/`
`data-controller-type`	The type of object you are embedding, which will normally match the portion of a URL for that object immediately following the base URL. Options include: **Dashboard**, **Scorecard**, **Report**, **SmallMultiple**, **Slideshow**, **Shortlink**, **MetricSet**, **DataCube**, **CubePerspective**, **Hierarchy**, **Admin**, **Home**. Optional.
`data-file-system-id`	The ID of an existing file you want to embed if `data-controller-type` is set to a file type such as a dashboard or metric set. This can be left unset to create a new file of that type.

Attribute	Description
	Optional.
`data-shortlink`	The short link value following the `/Link/?shortLink=` portion of the link (such as `ae6swdok5p5r0mm3m4oagueuwo`) if `data-controller-type` is **Shortlink**, for example when using a shared link.
`data-logon-token-id`	The logon token ID (populated using code). Optional.
`data-session-token`	The session token (populated using code). Optional.
`data-view-options`	The view options when displaying a view such as a dashboard. The following are the possibilities: **menuonly**, **menutoolbar**, **menutoolbartaskbar**, **none**, **toolbaronly**, or **viewonly**. By default, **viewonly** is used when you specify the ID of an existing view. This can be left unset for a **shortlink** to use the view options chosen when creating the shortlink. Optional.
`data-parametervalue-1`	The way to set view parameter values for a view such as a dashboard. You can use multiple of these attributes by increasing their number (`data-parametervalue-2`, `data-parametervalue-3`, etc.) The values are in a similar format as when passed via query string, such as `viewParameter2=$Today$`. Optional.
`data-urlparameteroption-1`	The way to add other URL parameters. You can use multiple of these attributes by increasing their number (`data-urlparameteroption-2`, `data-urlparameteroption-3`, etc.) For example, `cultureName=fr-FR`, or `e=true` to open a dashboard in edit mode. Optional.

The following example embeds the Symphony instance located at `https://placeholder.your-url.com`, displaying the dashboard with the specified ID. It will log on automatically with the specified logon token, set two view parameters, and set the culture to `fr-FR`:

```
...
<head>
	<script src="/Scripts/Embed/dundas.embed.min.js" type="text/javascript" ></script>
</head>
...

<body>
	<div class="dundas-bi-embed"
```

```
                data-dundas-bi-url="https://placeholder.dundas-bi-url.com/"
                data-controller-type="Dashboard"
                data-file-system-id="727ae6b6-b5b6-4c1e-bad7-72561040298d"
                data-logon-token-id="13ECB6B5-8324-473B-BCA5-6D18E2DC3988"
                data-parametervalue-1="viewParameter2=$Today$"
                data-parametervalue-2="viewParameter4=2500~3700"
                data-urlparameteroption-1="cultureName=fr-FR">
        </div>
</body>
```

User Log On Options

There are a few options to choose from for how your users will be logged onto Symphony:

Single Sign-On - To allow your users to be logged into Symphonyautomatically, use one of the various options described in the article Single sign-on (SSO).

Server-side code - A logon token or session token can be specified in the HTML as shown in the example markup above by code that runs at the server to retrieve it for each user that logs on. See the viewer integration sample web application for an example, and the section about embedding in the article Single sign-on (SSO) for more information.

Manual log on - If a single sign-on option is not used and a valid logon or session token is not specified, users will be prompted to log on before they can navigate to the specified dashboard or file.

Embedding With JavaScript

This applies to: *Managed Dashboards, Managed Reports*

Using the embed library with JavaScript provides similar options compared to others available in Symphony, but you can gain more control of the embedded application. For example:

- Set view parameters from JavaScript.
- Load different Symphony pages or refresh them.
- Monitor the embedded SymphonyI client application's events.
- Run JavaScript within the embedded Symphony client application and get messages sent back (if enabled).

Some of these functions may require setting up cross-origin resource sharing (CORS) for your site in Symphony to authorize it. See Configure Cross-Origin Resource Sharing (CORS).

After you define your logon option(s), see the following topics for more information about embedding with JavaScript.

- Creating An Embedded Application Object
- Using The Embedded Application
- Embedded Methods
- Configure Cross-Origin Resource Sharing (CORS)
- Working With View Parameters
- URL Path Length Restrictions

Get a Logon Token

When not using one of the other automatic logon options described in the article Single sign-on (SSO), it is common to use logon tokens or session tokens to log users on automatically based on their existing session in the application that is embedding Symphony.

Important: When getting a logon or session token, it is best practice to do so at the server. Specifying hard-coded credentials within the client-side script itself makes these credentials available to users viewing the page script. Examples for server-side code are given below.

The following example gets a logon token for use with a Symphony embedded application at the client using specified credentials for demonstration purposes only. For an example of how to use server-side code to get a logon token securely and use it with the embed library, see the Symphony viewer integration sample web application and the examples provided for POST /LogOn/Token in the REST API. For more information and links regarding logon and session tokens, see Single sign-on (SSO).

```
...
<head>
	<script src="/Scripts/Embed/dundas.embed.min.js" type="text/javascript" ></script>
</head>
...

<body>
	<script type="text/javascript">

document.addEventListener("DOMContentLoaded", function (event) {

	// ************************************************************************
	// It is a best practice to get the logon token at the server.
	// Credentials should not be specified directly in script unless they should be
	// freely available to users.
	// ************************************************************************

	dundas.embed.logon.getLogonToken(
		'https://placeholder.dundas-bi-url.com/',
		{
			"accountName": "viewer",
			"password": "1234",
			"isWindowsLogOn": true
		},
		function(getLogOnTokenResultData)
		{
			alert(getLogOnTokenResultData.logOnToken);
		}
	);
});

		</script>
</body>
```

Note: When getting a logon token to a site using HTTPS, the `xmlHttpRequest` adds the `withCredentials` flag with a `true` value to ensure the origin header is sent. The property `dundas.embed.logon.disableWithCredentialsFlag`, when set to `true` will not send the `withCredentials` property when getting the logon token.

Creating an Embedded Application Object

This applies to: *Managed Dashboards, Managed Reports*

The embedded application object is used to control the Symphony embedded application. Create one by calling the following method, passing a container HTML element for the application (such as a div element):

```
dundas.embed.create(domElement, embedOptions)
```

The `embedOptions` argument is optional and allows you to set any desired properties while creating the object instead of setting those properties on it later.

The example below shows how to create and load the Symphony embedded application object.

```
...
<head>
	<script src="/Scripts/Embed/dundas.embed.min.js" type="text/javascript" ></script>
</head>
...
<body>

	<!-- Create the container for the embedded application -->
	<div id="dundasBI2" />

	<script type="text/javascript">

		var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';

		document.addEventListener("DOMContentLoaded", function (event) {

			var embedOptions = {
			dundasBIUrl: dundasBIUrl,
			controllerType: dundas.embed.ControllerType.DASHBOARD,
			fileSystemId: 'f22ce55f-04cd-4207-9dd7-2cadeb44b96c',
			viewOptions: dundas.embed.ViewOptions.TOOLBAR_ONLY,
			parameterValues: []
			};

			// Create the  embedded application object.
			var dundasApp = dundas.embed.create(
			document.getElementById('dundasBI2'),
			embedOptions
			);
```

```
                        // Load the application
                        dundasApp.load();
                });

        </script>
</body>
```

Using the Embedded Application

This applies to: *Managed Dashboards, Managed Reports*

After creating the embedded application object, you can access the following properties on it. See Embedded Methods for information on methods to use.

Application Properties

Property	Description
`controllerType`	Gets or sets the controller type to load, which matches the portion of a Symphony URL following the base URL. Type: `dundas.embed.ControllerType` or `String` (can be null). Predefined values include: ■ `dundas.embed.ControllerType.DASHBOARD` ■ `dundas.embed.ControllerType.REPORT` ■ `dundas.embed.ControllerType.SCORECARD` ■ `dundas.embed.ControllerType.SHORTLINK` ■ `dundas.embed.ControllerType.SLIDESHOW` ■ `dundas.embed.ControllerType.SMALL_MULTIPLE` ■ `dundas.embed.ControllerType.METRIC_SET` ■ `dundas.embed.ControllerType.DATA_CUBE` ■ `dundas.embed.ControllerType.CUBE_PERSPECTIVE` ■ `dundas.embed.ControllerType.HIERARCHY` ■ `dundas.embed.ControllerType.ADMIN`

Property	Description
	■ `dundas.embed.ControllerType.HOME`
`dundasBIUrl`	Gets or sets the Symphony URL to connect to. For example, https://dbi.example.com:8000/ Type: String
`fileSystemId`	Gets or sets the file system entry ID of an existing file such as a dashboard, metric set, etc., if `controllerType` is set to one of those file types. This can also be unset (null) to create a new file of that type. Type: String (can be null)
`id`	Gets the ID for the Symphony embedded application. Type: String
`logonTokenId`	Gets or sets the logon token ID. Type: String (can be null)
`sessionToken`	Gets or sets the session token. Type: String (can be null)
`otherUrlParameterOptions`	Gets or sets other URL parameter options. For example: ■ `[new dundas.embed.UrlParameterOption("cultureName", "fr-FR")]` To provide an option for French ■ `[new dundas.embed.UrlParameterOption("e", "true")]` To open a dashboard in edit mode Type: Array (can be null) ElementType: `dundas.embed.UrlParameterOption` - construction: `new dundas.embed.UrlParameterOption(name, value)`
`parameterValues`	Gets or sets the parameter values to pass to a view such as a dashboard. The values are in a similar format as when passed via query string. For example: `[new dundas.embed.ParameterValue("viewParameter2", "$Today$")` Type: Array (can be null)

Property	Description
	ElementType: `dundas.embed.ParameterValue` - construction: `new dundas.embed.ParameterValue (name, value)`
`shortLink`	Gets or sets the short link value following the `/Link/?shortLink=` portion of the link (for example: `ae6swdek5p5r3gm3m4oaguqujo`) if `controllerType` is `SHORTLINK`, for example when using a shared link. Type: String (can be null)
`viewOptions`	Gets or sets the view options when displaying a view such as a dashboard. If unset (null), the view options from a shortlink (if applicable) or the default will be used. Type: `dundas.embed.ViewOptions` or `String` (can be null). Possible values include: ■ `dundas.embed.ViewOptions.MENU_ONLY` ■ `dundas.embed.ViewOptions.MENU_TOOLBAR` ■ `dundas.embed.ViewOptions.MENU_TOOLBAR_TASKBAR` ■ `dundas.embed.ViewOptions.NONE` ■ `dundas.embed.ViewOptions.TOOLBAR_ONLY` ■ `dundas.embed.ViewOptions.VIEW_ONLY`

Embedded Application Example

This sample creates a Symphony embedded application, sets some properties, and loads it:

```
<head>
       <script src="/Scripts/Embed/dundas.embed.min.js" type="text/javascript" ></script>
</head>
...
<body>
       <div id="dundasBI2">
       </div>

       <script type="text/javascript">
```

```
var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';

document.addEventListener("DOMContentLoaded", function (event) {

        // ************************************************************************
        // It is usually best practice to get the logon token at the server.
        // Credentials should not be specified directly in script unless they should be
        // freely available to users.
        // ************************************************************************
        dundas.embed.logon.getLogonToken(
                dundasBIUrl,
                {
                        "accountName": "viewer",
                        "password": "1234",
                        "isWindowsLogOn": true
                },
                function(getLogOnTokenResultData)
                {
                        var dundasApp = dundas.embed.create(
                                document.getElementById('dundasBI2'),
                                { }
                        );

                        dundasApp.dundasBIUrl = dundasBIUrl;
                        dundasApp.controllerType = dundas.embed.ControllerType.DASHBOARD;
                        dundasApp.fileSystemId = 'f22ce55f-04cd-4207-9dd7-2cadeb44b96c';
                        dundasApp.viewOptions = dundas.embed.ViewOptions.TOOLBAR_ONLY;
                        dundasApp.logonTokenId = getLogOnTokenResultData.logOnToken;
                        dundasApp.parameterValues = [
                                new dundas.embed.ParameterValue(
                                        "viewParameter2",
                                        "!" + 25000 + "~" + dundas.embed.tokens.basic.OPEN_RANGE
                                )
                        ];
                        dundasApp.otherUrlParameterOptions = [
                                new dundas.embed.UrlParameterOption(
                                        "cultureName",
                                        "fr-FR"
                                )
                        ];
                        dundasApp.load();
                }
        )
```

```
                });

        </script>
</body>
```

Embedded Methods

This applies to: *Managed Dashboards, Managed Reports*

After creating the embedded application object, you can access the following methods on it. See Application Properties for information on properties to use.

Name	Description	Example

`load`	Loads the embedded application frame according to the properties that have been set.	`var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';`
		
		`var embedOptions = {`
		`  dundasBIUrl: dundasBIUrl,`
		`  controllerType: dundas.embed.ControllerType.DASHBOARD,`
		`  fileSystemId: 'f22ce55f-04cd-4207-9dd7-2cadeb44b96c'`
		`};`
		`// Create the embedded application object.`
		`var dundasApp = dundas.embed.create(`
		`  document.getElementById('dundasBI2'),`
		`  embedOptions`
		`);`
		
		`// * Load the application *`
		`dundasApp.load();`

`loadAndCreateShortLink`	Creates a shortlink that includes the current `parameterValues` property setting and uses this link to load the embedded application frame rather than specifying values directly in the query string. CORS is required to call this method.
	
	Parameters: `options` (properties: `logOnTokenId`, `sessionToken`, `deleteOtherSessions`)
		`var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';`
		
		`var logonToken1 = '1fb29869-b276-41ab-9cbd-098d17b675b4';`
		`var logonToken2 = 'cb316db8-5b6d-488f-8c31-fa20c8d66524';`
		
		`var embedOptions = {`
		`  dundasBIUrl: dundasBIUrl,`
		`  controllerType: dundas.embed.ControllerType.DASHBOARD,`
		`  fileSystemId: 'f22ce55f-04cd-4207-9dd7-2cadeb44b96c',`
		`  logonTokenId: logonToken1`
		`};`
		
		`// Create the embedded application object.`
		`var dundasApp = dundas.embed.create(`
		`  document.getElementById('dundasBI2'),`
		`  embedOptions`
		`);`
		
		`var logOnOptions = {`

Name	Description	Example

		`logOnTokenId: logonToken2,`
		`deleteOtherSessions: true`
		`};`
		
		`// * Load the application *`
		`dundasApp.loadAndCreateShortLink(logOnOptions);`

`loaded`	Handles when the embedded application is first loaded.
	
	Parameters: `loadedFunction` (Function)
		`var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';`
		
		`var embedOptions = {`
		`dundasBIUrl: dundasBIUrl,`
		`controllerType: dundas.embed.ControllerType.DASHBOARD,`
		`fileSystemId: 'f22ce55f-04cd-4207-9dd7-2cadeb44b96c'`
		`};`
		`// Create the embedded application object.`
		`var dundasApp = dundas.embed.create(`
		`document.getElementById('dundasBI2'),`
		`embedOptions`
		`);`
		
		`// * Pass a function to the loaded function *`
		`dundasApp.loaded(function() { alert('Dashboard is Loaded'); });`
		
		`// Load the application`
		`dundasApp.load();`

`ready`	Handles when the embedded application is ready after page load. CORS is required to call this method.
	
	Parameters: `readyFunction` (Function)
		`var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';`
		
		`var embedOptions = {`
		`dundasBIUrl: dundasBIUrl,`
		`controllerType: dundas.embed.ControllerType.DASHBOARD,`
		`fileSystemId: 'f22ce55f-04cd-4207-9dd7-2cadeb44b96c'`
		`};`
		`// Create the embedded application object.`
		`var dundasApp = dundas.embed.create(`
		`document.getElementById('dundasBI2'),`
		`embedOptions`
		`);`
		`// * Pass a function to the ready function *`

Name	Description	Example

		`dundasApp.ready(function() { alert('Dashboard is Ready'); });`
		`// Load the application`
		`dundasApp.load();`

`refresh`	Refreshes the embedded application content.	`var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';`
		
		`var embedOptions = {`
		`dundasBIUrl: dundasBIUrl,`
		`controllerType: dundas.embed.ControllerType.DASHBOARD,`
		`fileSystemId: 'f22ce55f-04cd-4207-9dd7-2cadeb44b96c'`
		`};`
		`// Create the embedded application object.`
		`var dundasApp = dundas.embed.create(`
		`document.getElementById('dundasBI2'),`
		`embedOptions`
		`);`
		
		`// * Refresh the embedded application *`
		`dundasApp.refresh();`

`runScript`	Runs JavaScript within the embedded application in the browser. CORS is required to call this method.
	
	Parameters: `script` (String), `isASync` (Boolean, optional), `messageReceivedFunction` (Function, optional)
		`var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';`
		
		`var embedOptions = {`
		`dundasBIUrl: dundasBIUrl,`
		`controllerType: dundas.embed.ControllerType.DASHBOARD,`
		`fileSystemId: 'f22ce55f-04cd-4007-9dd7-2aedeb44b96c'`
		`};`
		`// Create the embedded application object.`
		`var dundasApp = dundas.embed.create(`
		`document.getElementById('dundasBI2'),`
		`embedOptions`
		`);`
		
		`dundasApp.ready(function (e) {`
		`// * Pop up a hello ready dialog.`
		`dundasApp.runScript(`
		`"var viewService = " +`
		`"  dundas.context.getService('ViewService'); " +`

Name	Description	Example

		`"viewService.showSignal(" +`
		`"'Ready'," +`
		`"dundas.view.NotifyType.INFO," +`
		`"'Hello Ready'" +`
		`");"`
		`);`
		
		`// * Run script and send message back,`
		`// and handle it with function that accepts message.`
		`dundasApp.runScript(`
		`"dundas.embed.sendMessageToDundasEmbeddedApplication(" +`
		`"dundas.context.currentSessionId" +`
		`");",`
		`false,`
		`function (message) { alert(message.detail); }`
		`);`
		`});`

Working With View Parameters

This applies to: *Managed Dashboards, Managed Reports*

View Parameters in the Symphony embed library are defined the same way they would be defined via the query string. The following example sets a view parameter to a value and then loads the embedded application.

```
var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';

document.addEventListener("DOMContentLoaded", function (event) {

    // ************************************************************************
    // It is usually best practice to get the logon token at the server.
    // Credentials should not be specified directly in script unless they should be
    // freely available to users.
    // ************************************************************************
    dundas.embed.logon.getLogonToken(
        dundasBIUrl,
        {
            "accountName": "viewer",
            "password": "1234",
            "isWindowsLogOn": true
        },
        function(getLogOnTokenResultData) {
            var dundasApp = dundas.embed.create(
            document.getElementById('dundasBI2'),
            {
                dundasBIUrl: dundasBIUrl,
                controllerType: dundas.embed.ControllerType.DASHBOARD,
                fileSystemId: 'f22ce55f-04cd-4207-9dd7-2cadeb44b96c',
                logonTokenId: getLogOnTokenResultData.logOnToken
            });

            dundasApp.parameterValues = [
                // The object is created by passing the name,
                // and then the value formatted in the same way
                // that would be defined via the query string
                new dundas.embed.ParameterValue(
                    // The name of the view parameter
                    "viewParameter2",
                    // The value formatted to be passed in the
                    // same way as if was passed via query string.
                    "!" + 25000 + "~" + dundas.embed.tokens.basic.OPEN_RANGE
                )
```

```
            ];

            dundasApp.load();
        }
    );
});
```

Note: For more information, see Pass parameter values via query string.

Token Helpers

For convenience purposes, token helpers have been added that can be used in the values of the `ParameterValue` object.

Basic

- `dundas.embed.tokens.basic.ALL`
- `dundas.embed.tokens.basic.DEFAULT`
- `dundas.embed.tokens.basic.NO_SELECTION`
- `dundas.embed.tokens.basic.NULL`
- `dundas.embed.tokens.basic.OPEN_RANGE`

SingleDate

- `dundas.embed.tokens.SingleDate.BEGINNING_OF_CURRENT_DAY`
- `dundas.embed.tokens.SingleDate.BEGINNING_OF_CURRENT_MONTH`
- `dundas.embed.tokens.SingleDate.BEGINNING_OF_CURRENT_QUARTER`
- `dundas.embed.tokens.SingleDate.BEGINNING_OF_CURRENT_WEEK`

- dundas.embed.tokens.SingleDate.BEGINNING_OF_CURRENT_YEAR
- dundas.embed.tokens.SingleDate.END_OF_CURRENT_YEAR

DateRange

- dundas.embed.tokens.DateRange.CURRENT_DAY
- dundas.embed.tokens.DateRange.CURRENT_MONTH
- dundas.embed.tokens.DateRange.CURRENT_QUARTER
- dundas.embed.tokens.DateRange.CURRENT_WEEK
- dundas.embed.tokens.DateRange.CURRENT_YEAR
- dundas.embed.tokens.DateRange.MONTH_TO_DATE
- dundas.embed.tokens.DateRange.PREVIOUS_DAY
- dundas.embed.tokens.DateRange.PREVIOUS_MONTH
- dundas.embed.tokens.DateRange.PREVIOUS_QUARTER
- dundas.embed.tokens.DateRange.PREVIOUS_WEEK
- dundas.embed.tokens.DateRange.PREVIOUS_YEAR
- dundas.embed.tokens.DateRange.QUARTER_TO_DATE
- dundas.embed.tokens.DateRange.TODAY
- dundas.embed.tokens.DateRange.YEAR_TO_DATE

URL Path Length Restrictions

This applies to: *Managed Dashboards, Managed Reports*

If a URL query generates more than 2,083 characters (for example through the use of view parameters), an error will occur. To work around these browser limitations, create a short link, and then load the Symphony application.

The following example uses two logon tokens to create a short link and load the embedded application:

```
var dundasBIUrl = 'https://placeholder.dundas-bi-url.com/';

document.addEventListener("DOMContentLoaded", function (event) {

	// ************************************************************************
	// It is a best practice to get the logon token at the server.
	// Credentials should not be specified directly in script unless they should be
	// freely available to users.
	// ************************************************************************
	var logOnTokenOptions = { "accountName": "viewer", "password": "1234", "isWindowsLogOn": false };

	// Gets two logon tokens: one to generate the shortlink, and one that then
	// loads using the shortlink.
	dundas.embed.logon.getLogonToken(
		dundasBIUrl,
		logOnTokenOptions,
		function (logonTokenResultData1) {

			dundas.embed.logon.getLogonToken(
				dundasBIUrl,
				logOnTokenOptions,
				function (logonTokenResultData2) {

					var dundasApp = dundas.embed.create(document.getElementById('dundasBI2'));
					dundasApp.logonTokenId = logonTokenResultData1.logOnToken;
					dundasApp.dundasBIUrl = dundasBIUrl;
					dundasApp.controllerType = dundas.embed.ControllerType.DASHBOARD;
					dundasApp.fileSystemId = '460ebfa5-116a-489b-9440-87e1c18ef957';
					dundasApp.parameterValues = [
						new dundas.embed.ParameterValue(
							"viewParameter2",
							"!" + 25000 + "~" + dundas.embed.tokens.basic.OPEN_RANGE
						)
					];
```

```
                                        var logOnOptions = {
                                                'logOnTokenId': logonTokenResultData2.logOnToken,
                                                'deleteOtherSessions': true
                                        };

                                        dundasApp.loadAndCreateShortLink(logOnOptions);
                                }
                        );
                }
        );
});
```

Host Embed Library File

This applies to: *Managed Dashboards, Managed Reports*

You can also download or refer to the embed library on dundas.com with a URL like the one in the following HTML:

```
<head>
       <script src="//www.dundas.com/files/dbi/dundas.embed.js/10.0.0.1000/dundas.embed.min.js"
              type="text/javascript">
       </script>
</head>
```

Note: Speed and uptime are not guaranteed. insightsoftware strongly recommends you place the embed library on your own CDN.