18 Nov 2021

D3 Collection library (1.0.1)

1. Overview

D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS.

## Use case

### The D3.js Collection Libraries contains components that integrate the visualization features with Iris. You can use these components in scenarios such as: an Analytics app, in which you want to visually represent statistical data to the user.

The following charts are bundled in the collection library -

1. Population Pyramid
2. Sequence Sunburst
3. Collapsible Indented Tree
4. Live Update Chart

## Percentage of re-use:

Approximate 85% of reuse.

# Getting Started

## Prerequisites

This is a getting started section for the D3.js collection library. It contains information about setting up and running components from the library in an application.

Before you start using the D3.js collection library, make sure that you meet all the prerequisites.

• [HCL Foundry](https://manage.hclvoltmx.com/)

• Volt MX Iris

## Platforms Supported

### PWA/Responsive Web

### After you make sure that you meet all the prerequisites, you can expand any of the following drop-down sections for details about every step of getting started with the collection library

## Importing the collection library to your project

## You can import the Forge components only into the apps that are of the Reference Architecture type.

## Make sure that you are signed into Volt MX Iris.

## You can download the collection library onto your system as a zip file from Volt MX Forge.

## You can then import the zip file by selecting the Import Library option from My Libraries as shown in below.

## 

After Iris downloads the collection library, it adds the library to your workspace. You can access the library from the My Libraries tab on the lower-left panel on Iris.

After you import the collection library, you can add components from the library to your app.

Graphical user interface, text, application

Description automatically generated

After you add the components to your form, you can Preview or Build your app.

## Adding components to your app

#### From the lower-left panel of Iris, switch to the My Libraries tab

#### A screenshot of a phone Description automatically generated with medium confidence

#### From the drop-down list, select the D3.js library that you imported from Forge.

#### From Collections, drag and drop a component onto your form.

#### 

After you add a component to a form, you can [Preview or Build your app](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Getting_Started.htm#Preview).

For more information on the components in the library, you can refer to the following sections of the document.

[Collapsible Tree Component](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/CollapsibleTree.htm)

[Live Update Chart Component](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/LiveUpdateChart.htm)

[Population Pyramid Component](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/PopulationPyramid.htm)

[Sequence Sunburst Component](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/SequenceSunburst.htm)

Modifying the UI of a component

You can modify the UI of a component by editing the CSS files that are associated with the component.

To access the CSS files that are associated with a component, follow these steps:

1. From the **Project** explorer, go to **Web** → **Localfiles**.
2. Open the folder that has the same name as the component you want to modify.  
   For example: Open **d3populationpyramid**.
3. From the **styles** folder, select a CSS file that you want to edit.
4. Edit the CSS based on your preference.

## Text Description automatically generated

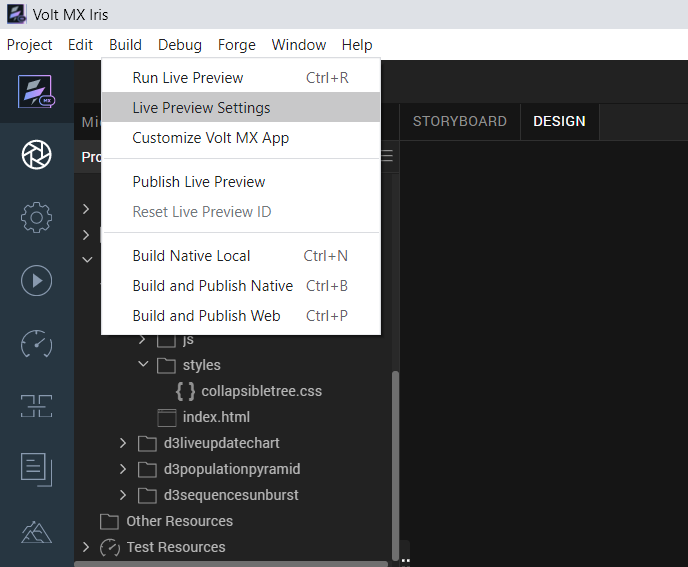
After you edit the CSS, you can [Build and Preview your app](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Getting_Started.htm#Preview) to view the modifications to the UI.

For more information on the components in the library, you can refer to the following sections of the document.

* [Collapsible Tree Component](D3%20Collections%20-%20HowTo.docx)
* [Live Update Chart Component](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/LiveUpdateChart.htm)
* [Population Pyramid Component](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/PopulationPyramid.htm)
* [Sequence Sunburst Component](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/SequenceSunburst.htm)

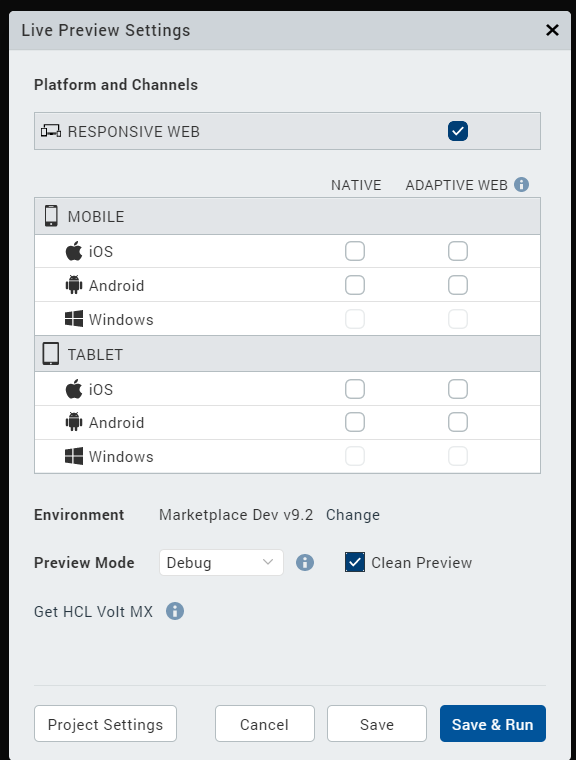
## Building and previewing your app

After performing all the above steps, you can perform either of the following tasks.



Graphical user interface, table

Description automatically generated



For more information, you can refer to the [Building and Viewing an Application](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_user_guide/Content/Cloud_Build_in_VoltMX_Iris.html#cloud) section of the Volt MX User Guide.

# **D3.js:** [**Collapsible Tree Component**](D3%20Collections%20-%20HowTo.docx)

The Collapsible Tree component in the D3.js displays data in a tree structure. A user can expand or collapse different parts of the data tree by clicking the header nodes.

**Note:**

>> To render the tree with data, call the setData API by using Actions.

>> If you copy the code from any example, make sure that you replace componentID with the name of your component.

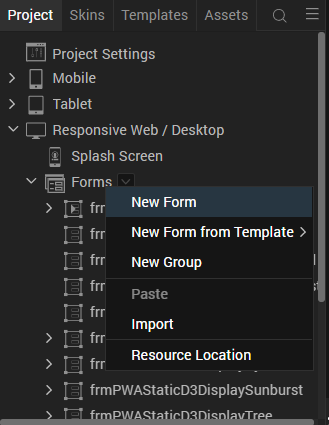


## [Dynamic Usage](javascript:void(0);)

If you want to use a component dynamically, you need to add the component to your project **Templates**. Follow the given steps to do so.

From the **Project** explorer, expand **Responsive Web/Desktop**.

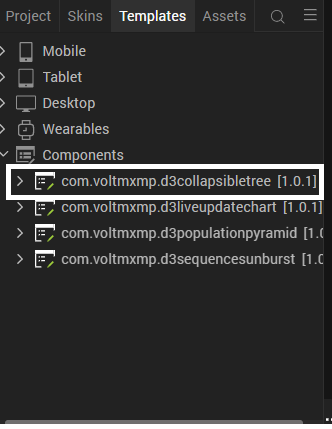
Right-click forms and select **New Form**



From the **My Libraries** tab, drag and drop the component onto the form.

Right-click the form that you created and select **Delete**.

From the **Project** explorer, go to the **Templates** tab.  
The component that you want to add should be available in the **Components** section.



After you add the component into your project templates, follow the given steps to add the component to your app dynamically.

Access the **FormController** of the form you want to add the component into.

Graphical user interface, application

Description automatically generated

Create a function called createComponent(); and type a code inside the function to create and configure the component.  
You can refer to the given sample code for more information.

|  |
| --- |
| /\* Creating the component's object \*/  var collapsibleTreeObj = new com.voltmxmp.d3collapsibletree(  {  "clipBounds": true,  "height": "100%",  "id": "CollapsibleTree",  "isVisible": true,  "left": "0dp",  "top": "0dp",  "width": "100%",  "zIndex": 1  }, {}, {});  /\* Setting the component's properties \*/  collapsibleTreeObj.sampleData = false;  collapsibleTreeObj.headerColor = "#fff";  collapsibleTreeObj.itemColor = "#dcdcdc";  /\* Adding the component to the form \*/  this.view.add(collapsibleTreeObj);  /\* Calling the component's API to render the tree \*/  this.view.CollapsibleTree.setData("https://gist.githubusercontent.com/mbostock/1093025/raw/b40b9fc5b53b40836ead8aa4b4a17d948b491126/flare.json"); |

To execute the code, you need to call this function using **Actions**. For more information, refer to the [Add Actions](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#working_with_Action_Editor.htm?TocPath=Designing%2520an%2520Application|Add%2520Actions|_____0) section of the Iris User Guide.

## **Properties**

You can use a component's **Properties** to customize and configure the elements. These elements can be UI widgets, service parameters, and other elements. For more information about properties, you can refer to the [Components Overview](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#C_ComponentsOverview.htm?TocPath=Creating%2520Applications%2520with%2520Components|_____1) section of the Iris User Guide.

You can set the properties from the **Properties** panel on the right side of Iris. You can also configure these properties using a JavaScript code.

#### General

[[Open](javascript:void(0);)Show Sample Data](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Toggles the visibility of the sample data of the component. |
| **Syntax:** | sampleData |
| **Type:** | Boolean |
| **Read/Write:** | Write |
| **Default Value:** | True |
| **Example:** | this.view.componentID.sampleData = false; |

[[Open](javascript:void(0);)  
Data URL](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Specifies the URL that the component uses to fetch the data. |
| **Syntax:** | userData |
| **Type:** | String |
| **Read/Write:** | Write |
| **Remarks:** | The URL must contain data in a JSON format |
| **Example:** | this.view.componentID.userData = “https://gist.githubusercontent.com /mbostock/1093025/raw/ b40b9fc5b53b40836ead8aa4b4a17d948b491126/flare.json |

[[Open](javascript:void(0);)Header Color](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Specifies the color of the headers of the tree. |
| **Syntax:** | headerColor |
| **Type:** | String |
| **Read/Write:** | Write |
| **Remarks:** | * Make sure that you enter the hex code with the pound (**#**) symbol. |
| **Example:** | this.view.componentID.headerColor = "#fff"; |

[[Open](javascript:void(0);)  
Item Color](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Specifies the color of the items of the tree. |
| **Syntax:** | itemColor |
| **Type:** | String |
| **Read/Write:** | Write |
| **Remarks:** | * Make sure that you enter the hex code with the pound (**#**) symbol. |
| **Example:** | this.view.componentID.itemColor = "#dcdcdc"; |

## **Events**

The component invokes events when its corresponding action is performed. You can configure any logic you want the component to perform whenever an event occurs. You can configure the events directly on the Actions tab or by writing a JavaScript, For more information, refer to [Add Actions](https://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Content/working_with_Action_Editor.htm) in the Iris User Guide.

[[Open](javascript:void(0);)onItemClick](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked when the user selects an item from the tree. |
| **Syntax:** | onItemClick |
| **Parameters:** | *itemName [String]* : Name of the selected item. |
| **Example:** | this.view.componentID.onItemClick = function(itemName)  {  alert("Selected Item: "+itemName);  }.bind(this); |

[[Open](javascript:void(0);)onErrorCallback](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked when an error occurs within the component. |
| **Syntax:** | onErrorCallback |
| **Parameters:** | *errObj [JSON]* : Information about the error such as the error code, and the error message. |
| **Example:** | this.view.componentID.onErrorCallback = function(errObj)  {  alert("Error Occured: "+JSON.stringify(errObj));  }.bind(this); |

## **APIs**

The following APIs pertain to the D3.js: Collapsible Tree component.

[Open](javascript:void(0);)[setData](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Renders a collapsible tree based on the specified data. |
| **Syntax:** | setData(json) |
| **Parameters:** | *json [String]* : A URL that contains the data (in JSON format) that you want to display on the tree. |
| **Return Value:** | None |
| **Example:** | var json =  "https://gist.githubusercontent.com/ mbostock/1093025/raw/ b40b9fc5b53b40836ead8aa4b4a17d948b491126/flare.json";  this.view.componentID.setData(json); |

# **D3.js:** [**Live Update Chart Component**](D3%20Collections%20-%20HowTo.docx)

The Live Update Chart component in the D3.js displays data as a horizontal chart that updates the displayed data after a specified time interval.

**Note:**

To render the chart with data, call the setChartData API by using Actions.

If you copy the code from any example, make sure that you replace componentID with the name of your component.

If the JSON that is specified in the Data URL property or the setChartData API is updated, the component updates the data after the specified Time Interval.

## [Dynamic Usage](javascript:void(0);)

If you want to use a component dynamically, you need to add the component to your project **Templates**. Follow the given steps to do so.

From the **Project** explorer, expand **Responsive Web/Desktop**.

Right-click forms and select **New Form**

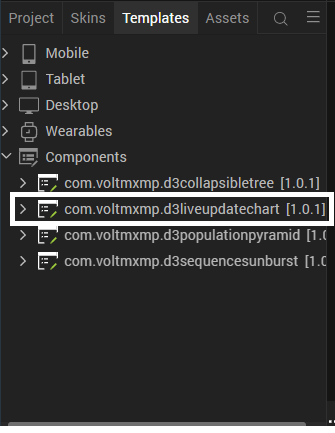
Graphical user interface

Description automatically generated

From the **My Libraries** tab, drag and drop the component onto the form.

Right-click the form that you created and select **Delete**.

From the **Project** explorer, go to the **Templates** tab.  
The component that you want to add should be available in the **Components** section.



After you add the component into your project templates, follow the given steps to add the component to your app dynamically.

Access the **FormController** of the form you want to add the component into.

Graphical user interface, application

Description automatically generated

Create a function called createComponent(); and type a code inside the function to create and configure the component.  
You can refer to the given sample code for more information.

|  |
| --- |
| /\* Creating the component's object \*/  var liveUpdateChartObj = new com.voltmxmp.d3liveupdatechart(  {  "clipBounds": true,  "height": "100%",  "id": "LiveUpdateChart",  "isVisible": true,  "left": "0dp",  "top": "0dp",  "width": "100%",  "zIndex": 1  }, {}, {});  /\* Setting the component's properties \*/  liveUpdateChartObj.timeInterval = 3;  liveUpdateChartObj.isSampleData = false;  /\* Adding the component to the form \*/  this.view.add(liveUpdateChartObj);  /\* Calling the component's API to render the chart \*/  this.view.LiveUpdateChart.setChartData("https://gist.githubusercontent.com/charlesdguthrie/11356441/raw/cc4659e14869f01c88cf461b817f84cd0e1353d0/fakeData.json"); |

To execute the code, you need to call this function using Actions. For more information, refer to the [Add Actions](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#working_with_Action_Editor.htm?TocPath=Designing%2520an%2520Application|Add%2520Actions|_____0) section of the Iris User Guide.

## **Properties**

You can use a component's **Properties** to customize and configure the elements. These elements can be UI widgets, service parameters, and other elements. For more information about properties, you can refer to the [Components Overview](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#C_ComponentsOverview.htm?TocPath=Creating%2520Applications%2520with%2520Components|_____1) section of the Iris User Guide.

You can set the properties from the **Properties** panel on the right side of Iris. You can also configure these properties using a JavaScript code.

#### General

[Open](javascript:void(0);)[Time Interval](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Specifies the time (in seconds) after which the component updates the data. |
| **Syntax:** | timeInterval |
| **Type:** | Integer |
| **Read/Write:** | Write |
| **Example:** | this.view.componentID.timeInterval = 3; |

[[Open](javascript:void(0);)Sample Data](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Toggles the visibility of the sample data of the component. |
| **Syntax:** | isSampleData |
| **Type:** | Boolean |
| **Read/Write:** | Write |
| **Default Value:** | True |
| **Example:** | this.view.componentID.isSampleData = false; |

[Open](javascript:void(0);)[Data URL](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Specifies the URL that the component uses to fetch the data. |
| **Syntax:** | userData |
| **Type:** | String |
| **Read/Write:** | Write |
| **Remarks:** | * The URL must contain data in a JSON format. |
| **Example:** | this.view.componentID.userData =  "https://gist.githubusercontent.com/ charlesdguthrie/11356441/raw/ cc4659e14869f01c88cf461b817f84cd0e1353d0 /fakeData.json"; |

## 

## **Events**

The component invokes events when its corresponding action is performed. You can configure any logic you want the component to perform whenever an event occurs. You can configure the events directly on the Actions tab or by writing a JavaScript, For more information, refer to [Add Actions](https://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Content/working_with_Action_Editor.htm) in the Iris User Guide.

[[Open](javascript:void(0);)onTimeIntervalExpiry](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked after the specified [Time Interval](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/LiveUpdateChart.htm#timeInterval) ends. |
| **Syntax:** | onTimeIntervalExpiry |
| **Parameters:** | *chartData [JSON]* : The data on the chart at the end of the time interval. |
| **Remarks:** | To update the chart after the time interval ends, use the [setChartData](https://docs.kony.com/marketplace/D3CollectionLibrary/Content/Components/LiveUpdateChart.htm" \l "setData) API in the actions of the onTimeIntervalExpiry event. |
| **Example:** | this.view.componentID.onTimeIntervalExpiry = function(chartData)  {  alert("Chart Data: "+JSON.stringify(chartData));  }.bind(this); |

[[Open](javascript:void(0);)onErrorCallback](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked when an error occurs within the component. |
| **Syntax:** | onErrorCallback |
| **Parameters:** | *errObj [JSON]* : Information about the error such as the error code, and the error message. |
| **Example:** | this.view.componentID.onErrorCallback = function(errObj)  {  alert("Error Occured: "+JSON.stringify(errObj));  }.bind(this); |

APIs

The following APIs pertain to the D3.js: Live Update Chart component.

[OpensetChartData](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Renders a chart based on the specified data. |
| **Syntax:** | setChartData(chartData) |
| **Parameters:** | *chartData [String]* : A URL that contains the data (in JSON format) that you want to display on the chart. |
| **Return Value:** | None |
| **Example:** | var chartData =  "https://gist.githubusercontent.com/  charlesdguthrie/11356441/raw/  cc4659e14869f01c88cf461b817f84cd0e1353d0/  fakeData.json";  this.view.componentID.setChartData(chartData); |

# **D3.js: Population Pyramid Component**

The Population Pyramid component in the D3.js displays data as a bar-graph and also displays the selected year on the upper-left corner. A user can change the year and move the graph by using the left and right arrow keys.

**Note:**

To render the graph with data, call the setData API by using Actions.

If you copy the code from any example, make sure that you replace componentID with the name of your component.

# [Dynamic Usage](javascript:void(0);)

If you want to use a component dynamically, you need to add the component to your project **Templates**. Follow the given steps to do so.

From the **Project** explorer, expand **Responsive Web/Desktop**.

Right-click forms and select **New Form**

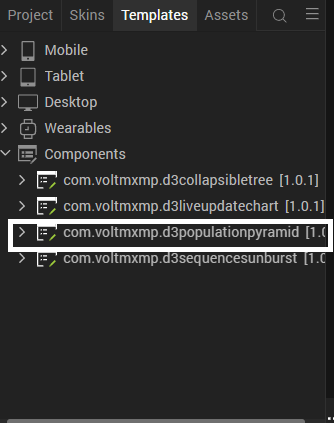
Graphical user interface

Description automatically generated

From the **My Libraries** tab, drag and drop the component onto the form.

Right-click the form that you created and select **Delete**.

From the **Project** explorer, go to the **Templates** tab.  
The component that you want to add should be available in the **Components** section.



After you add the component into your project templates, follow the given steps to add the component to your app dynamically.

Access the **FormController** of the form you want to add the component into.

Graphical user interface, application

Description automatically generated

Create a function called createComponent(); and type a code inside the function to create and configure the component. You can refer to the given sample code for more information.

|  |
| --- |
| /\* Creating the component's object \*/  var populationPyramidObj = new com.voltmxmp.d3populationpyramid( {  "clipBounds": true,  "height": "100%",  "id": "PopulationPyramid",  "isVisible": true,  "left": "0dp",  "top": "0dp",  "width": "100%",  "zIndex": 1  }, {}, {});  /\* Setting the component's properties \*/ populationPyramidObj.sampleData = false;  /\* Adding the component to the form \*/ this.view.add(populationPyramidObj);  /\* Calling the component's API to render the graph \*/ this.view.PopulationPyramid.setData("https://gist.githubusercontent.com/mbostock/4062085/raw/dbd4a66dcadd54fa189f84c106795c644401e753/population.csv"); |

To execute the code, you need to call this function using **Actions**. For more information, refer to the [Add Actions](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#working_with_Action_Editor.htm?TocPath=Designing%2520an%2520Application|Add%2520Actions|_____0) section of the Iris User Guide.

## **Properties**

You can use a component's **Properties** to customize and configure the elements. These elements can be UI widgets, service parameters, and other elements. For more information about properties, you can refer to the [Components Overview](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#C_ComponentsOverview.htm?TocPath=Creating%2520Applications%2520with%2520Components|_____1) section of the Iris User Guide.

You can set the properties from the **Properties** panel on the right side of Iris. You can also configure these properties using a JavaScript code.

#### General

#### [[Open](javascript:void(0);)Sample Data](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Toggles the visibility of the sample data of the component. |
| **Syntax:** | sampleData |
| **Type:** | Boolean |
| **Read/Write:** | Write |
| **Default Value:** | True |
| **Example:** | this.view.componentID.sampleData = false; |

[OpenData URL](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Specifies the URL that the component uses to fetch the data. |
| **Syntax:** | userData |
| **Type:** | String |
| **Read/Write:** | Write |
| **Remarks:** | * The URL must contain data in a CSV format. |
| **Example:** | this.view.componentID.userData =  "https:// gist.githubusercontent.com/ mbostock/4062085/raw/ dbd4a66dcadd54fa189f84c106795c644401e753 / population.csv"; |

## **Events**

The component invokes events when its corresponding action is performed. You can configure any logic you want the component to perform whenever an event occurs. You can configure the events directly on the Actions tab or by writing a JavaScript, For more information, refer to [Add Actions](https://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Content/working_with_Action_Editor.htm) in the Iris User Guide.

[[Open](javascript:void(0);)onKeyClick](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked when the user presses a key on the keyboard. |
| **Syntax:** | onKeyClick |
| **Parameters:** | *keyValue [String]* : Value of the key that was pressed on the keyboard. |
|  |  |
| **Example:** | this.view.componentID.onKeyClick = function(*keyValue*)  {  alert "Key Pressed: "+keyValue);  }.bind(this); |

[[Open](javascript:void(0);)onErrorCallback](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked when an error occurs within the component. |
| **Syntax:** | onErrorCallback |
| **Parameters:** | *errObj [JSON]* : Information about the error such as the error code, and the error message. |
| **Example:** | this.view.componentID.onErrorCallback = function(errObj)  {  alert("Error Occured: "+JSON.stringify(errObj));  }.bind(this); |

APIs

The following APIs pertain to the D3.js: Population Pyramid component.

[OpensetData](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Renders a graph based on the specified data. |
| **Syntax:** | setData(csvUrl) |
| **Parameters:** | *csvUrl [String]* : A URL that contains the data (in CSV format) that you want to display on the graph. |
| **Return Value:** | None |
| **Example:** | var csvUrl =  "https://gist.githubusercontent.com/  mbostock/4062085  /raw/ dbd4a66dcadd54fa189f84c106795c644401e753/  population.csv";  this.view.componentID.setData(csvUrl); |

# **D3.js: Sequence Sunburst Component**

The Sequence Sunburst component in the D3.js displays data as concentric circular charts and also contains a legend for the data that is displayed on the chart. A user can hover over an element on the chart to view the sequence and percentage data of the element.

**Note:**

To render the graph with data, call the setData API by using Actions.

# [Dynamic Usage](javascript:void(0);)

If you want to use a component dynamically, you need to add the component to your project **Templates**. Follow the given steps to do so.

From the **Project** explorer, expand **Responsive Web/Desktop**.

Right-click forms and select **New Form**

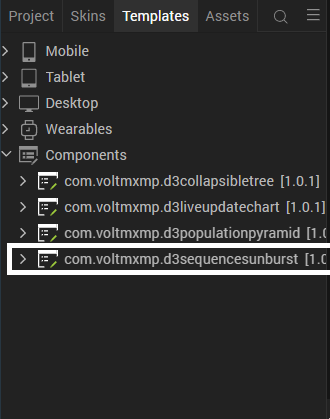
Graphical user interface

Description automatically generated

From the **My Libraries** tab, drag and drop the component onto the form.

Right-click the form that you created and select **Delete**.

From the **Project** explorer, go to the **Templates** tab.  
The component that you want to add should be available in the **Components** section.



After you add the component into your project templates, follow the given steps to add the component to your app dynamically.

Access the **FormController** of the form you want to add the component into.

Graphical user interface, text, application

Description automatically generated

Create a function called createComponent(); and type a code inside the function to create and configure the component.  
You can refer to the given sample code for more information.

|  |
| --- |
| /\* Creating the component's object \*/  var sequenceSunburstObj= new com.voltmxmp.d3sequencesunburst(  {  "clipBounds": true,  "height": "100%",  "id": "SequenceSunburst",  "isVisible": true,  "left": "0dp",  "top": "0dp",  "width": "100%",  "zIndex": 1  }, {}, {});  /\* Setting the component's properties \*/  sequenceSunburstObj.sampleData = false;  sequenceSunburstObj.userData = "https://gist.githubusercontent.com/kerryrodden/766f8f6d31f645c39f488a0befa1e3c8/raw/9fc86efac379f228749b2de3537acd629c0867c8/visit-sequences.csv";  sequenceSunburstObj.chartText = "Visited";  sequenceSunburstObj.legendMasterData =  {  "data":  [  {  "name" : "home",  "color" : "#5687d1"  },  {  "name" : "product",  "color" : "#7b615c"  },  {  "name" : "search",  "color" : "#de783b"  },  {  "name" : "account",  "color" : "#6ab975"  },  {  "name" : "other",  "color" : "#a173d1"  },  {  "name" : "end",  "color" : "#bbbbbb"  }  ]  };  /\* Adding the component to the form \*/  this.view.add(sequenceSunburstObj); |

To execute the code, you need to call this function using **Actions**. For more information, refer to the [Add Actions](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#working_with_Action_Editor.htm?TocPath=Designing%2520an%2520Application|Add%2520Actions|_____0) section of the Iris User Guide.

## **Properties**

You can use a component's **Properties** to customize and configure the elements. These elements can be UI widgets, service parameters, and other elements. For more information about properties, you can refer to the [Components Overview](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#C_ComponentsOverview.htm?TocPath=Creating%2520Applications%2520with%2520Components|_____1) section of the Iris User Guide.

You can set the properties from the **Properties** panel on the right side of Iris. You can also configure these properties using a JavaScript code.

#### General

[[Open](javascript:void(0);)Show Sample Data](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Toggles the visibility of the sample data of the component. |
| **Syntax:** | sampleData |
| **Type:** | Boolean |
| **Read/Write:** | Write |
| **Default Value:** | True |
| **Example:** | this.view.componentID.sampleData = false; |

[[Open](javascript:void(0);)  
Data URL](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Specifies the URL that the component uses to fetch the data. |
| **Syntax:** | userData |
| **Type:** | String |
| **Read/Write:** | Write |
| **Remarks:** | The URL must contain data in a CSV format |
| **Example:** | this.view.componentID.userData = “https://gist.githubusercontent.com /kerryrodden/766f8f6d31f645c39f488a0befa1e3c8  /raw/9fc86efac379f228749b2de3537acd629c0867c8  /visit-sequences.csv |

[[Open](javascript:void(0);)](javascript:void(0);)Chart Text

|  |  |
| --- | --- |
| **Description:** | Specifies the text that you want to display as the suffix to the element data. |
| **Syntax:** | chartText |
| **Type:** | String |
| **Read/Write:** | Write |
| **Example:** | this.view.componentID.chartText = "Visited"; |

Legends data

|  |  |
| --- | --- |
| **Description:** | Specifies the data that you want to display in the chart legend. |
| **Syntax:** | legendMasterData |
| **Type:** | Data Grid JSON |
| **Read/Write:** | Write |
| **Remarks:** | Every row in the Data Grid contains the following keys:   * *name [String]* : The name of the element in the legend * *color [String]* : The hex code of the background color of the element in the legend |
| **Example:** | this.view.componentID.legendMasterData =  {  "data":  [  {  "name" : "home",  "color" : "#5687d1"  },  {  "name" : "product",  "color" : "#7b615c"  },  {  "name" : "search",  "color" : "#de783b"  },  {  "name" : "account",  "color" : "#6ab975"  },  {  "name" : "other",  "color" : "#a173d1"  },  {  "name" : "end",  "color" : "#bbbbbb"  }  ]  } |

## **Events**

The component invokes events when its corresponding action is performed. You can configure any logic you want the component to perform whenever an event occurs. You can configure the events directly on the Actions tab or by writing a JavaScript, For more information, refer to [Add Actions](https://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Content/working_with_Action_Editor.htm) in the Iris User Guide.

[[Open](javascript:void(0);)onChartHover](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked when the user hovers over an element on the chart. |
| **Syntax:** | onChartHover |
| **Parameters:** | *paam [JSON]* : The data of the element that the user hovers on. |
| **Example:** | this.view.componentID.onChartHover = function(param)  {  alert("Hovered Over: "+JSON.stringify(param));  }.bind(this); |

[[Open](javascript:void(0);)onErrorCallback](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked when an error occurs within the component. |
| **Syntax:** | onErrorCallback |
| **Parameters:** | *errObj [JSON]* : Information about the error such as the error code, and the error message. |
| **Example:** | this.view.componentID.onErrorCallback = function(errObj)  {  alert("Error Occured: "+JSON.stringify(errObj));  }.bind(this); |

## **APIs**

The following APIs pertain to the D3.js: Sequence Sunburst component.

[OpensetData](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Renders a sequence sunburst chart based on the specified data. |
| **Syntax:** | setChartData(csvUrl, legendObj, suffix) |
| **Parameters:** | * *csvUrl [String]* : A URL that contains the data (in CSV format) that you want to display on the chart. * *legendObj [JSON]* : Contains the data that you want to display in the legend of the chart. * *suffix [String]* : The text that you want to display as the suffix to the element data |
| **Return Value:** | None |
| **Example:** | var csvUrl =  "https://gist.githubusercontent.com/ kerryrodden/766f8f6d31f645c39f488a0befa1e3c8/  raw/9fc86efac379f228749b2de3537acd629c0867c8  /visit-sequences.csv";  var legendObj =  {  "home": "#5687d1",  "product": "#7b615c",  "search": "#de783b",  "account": "#6ab975",  "other": "#a173d1",  "end": "#bbbbbb"  };  var suffix = "Visited";  this.view.setData.setChartData(csvUrl, legendObj, suffix); |

# Revision History

App version 1.0.1:

## Known Issues

## -

## Limitations

D3 Collections library is a web component. So, it doesn’t support for Android and IOS devices.