Chart Scripting

Functional Specifications

Draft 4: October 28, 2005

Abstract

This document describes the functional specifications of the Chart Scripting for BIRT release 1 and 2.

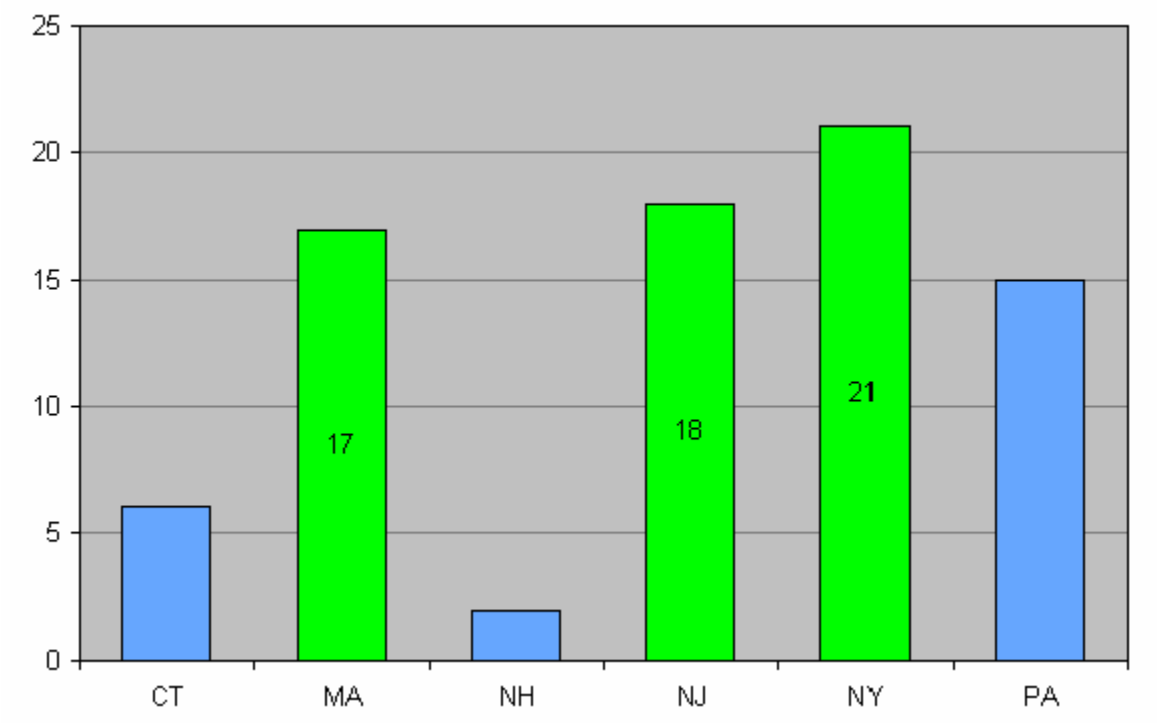
Document Revisions

Draft	Date	Primary Author(s)	Description of Changes
1	10/5/2005	David Michonneau	Initial Draft
2	10/24/2005	David Michonneau	Added Use Cases. Changed start/finish names. Added Chart Engine preparation API.
3	10/25/2005	David Michonneau	Added sections about Chart Model instances and script flow
4	10/28/2005	David Michonneau	Minor corrections to preparation, and diagram.

Contents

1. Introduction

Chart Scripting is used for customizing the output of the chart. Here is an example of output that is done using chart scripting:



Based on the data values, the script changes the colors of the bars and shows the value. More generally scripting allows the user to customize any aspect of the chart based on real-time data, and that could be the series, the legends, the axes, the plot, etc…

This scripting is done at generation time. This is not to be confused with interactivity scripting (viewing time)

2. Scripting functionalities

2.1 Scripting Language

Scripting is supported both in Java (from v2.0) and JavaScript (from v1.0). The Chart engine will detect the type of scripting automatically at run-time, by trying to resolve the script string in the model to a Java class, and assume JavaScript otherwise.

2.2 Script Context (v2)

An interface is available to allow the script to get access to common chart variables and communicate with an external context. This is available in v2, both in Java and JavaScript. It deprecates the JavaScript global functions.

interface IChartScriptContext

{

```
Chart getDesignTimeModel( );

Chart getRunTimeModel( );
```

```
        GeneratedChartState getGeneratedChartState( );

        Locale getLocale( );

        Object clone(Object) // to clone EMF objects if needed


            // To log messages using the default chart logging framework:
            // logger.logFromScript("running from script")
        Logger getLogger();

            // To access an external context, that is passed through the
    Generator
        Object getExternalContext();
}
```

The Generator class will allow the external context to be passed by overloading the build and render method, similarly to the ScriptableObject for JavaScript.

2.3 Script functions

Here are the list of functions the user can implement in JavaScript Note that the v1 ones are deprecated in v2, and all the inactive ones in v1 will simply be removed from v2 (not a breaking change since it was not working originally). Some new methods with new arguments will be available in v2.

Most of the "after" methods are fully deprecated in v2, the purpose of those was usually to restore the object changes by the "before" script, for further processing. This will be automatically handled in v2.

Legend

	Inactive in v1, fully removed in v2
	Functional in v1 and v2. Deprecated in v2
	New in v2. Replaces some of deprecated functions

Scope	Function	Arguments	v 1.x	v 2.0	Description
Design	onPrepare	Chart, IChartScriptContext	-	✓	Called only once for each chart design in the report, before any databinding occurred. Styles are already flattened in the design.
Data	startDataBinding	-	Inactive	-	-
	beforeQueryExecution	-	Inactive	-	-
	afterQueryExecution	-	Inactive	-	-
	beforeDataSetFilled	Series, IDataSetProcessor, IChartScriptContext	-	✓	Called before populating the series dataset using the DataSetProcessor

Scope	Function	Arguments	v 1.x	v 2.0	Description
	beforeDataSetFilled	Series, IDataSetProcessor	Inactive	-	Called before populating the series dataset using the DataSetProcessor
	afterDataSetFilled	Series, DataSet, IChartScriptContext	-	✓	Called after populationg the series dataset
	afterDataSetFilled	Series, DataSet	Inactive	-	Called after populationg the series dataset
	finishDataBinding	-	Inactive	-	-
Generation	beforeGeneration	Chart, IScriptContext	-	✓	Called before generation of chart model to GeneratedChartState
	startGeneration	Chart	✓	Deprecated	Called before generation of chart model to GeneratedChartState is started
	beforeLayout	Chart	✓	Deprecated	Called before block layout is performed
	afterLayout	Chart	✓	Deprecated	Called after blocks layout has been performed
	beforeComputations	Chart, Object	✓	Deprecated	Called before performing size computations
	afterComputations	Chart, Object	✓	Deprecated	Called after performing size computations
	finishGeneration	GeneratedChartState	✓	Deprecated	Called when generation to GeneratedChartState is completed
Rendering	beforeRendering	GeneratedChartState, IScriptContext	-	✓	Called before the chart is rendered
	startRendering	GeneratedChartState	✓	Deprecated	Called when rendering of the GeneratedChartState starts
	finishRendering	GeneratedChartState	✓	Deprecated	Called after the rendering is completed
Block	beforeDrawBlock	Block, IScriptContext	-	✓	Called before drawing the outermost block

Scope	Function	Arguments	v 1.x	v 2.0	Description
	beforeDrawBlock	Block	✓	Deprecated	Called before drawing each block
	afterDrawBlock	Block	✓	Deprecated	Called after drawing each block
Chart	beforeDrawPlot		Inactive	-	
	afterDrawPlot		Inactive	-	
	beforeDrawTitle		Inactive	-	
	afterDrawTitle		Inactive	-	
Legend	beforeDrawLegend	-	Inactive	-	
	afterDrawLegend	-	Inactive	-	
	beforeDrawLegendEntry	Label, IScriptContext	-	✓	Called before drawing each entry in the legend
	beforeDrawLegendEntry	Label	✓	Deprecated	Called before drawing each entry in the legend
	afterDrawLegendEntry	Label	✓	Deprecated	Called after drawing each entry in the legend
Series	startComputeSeries	Series	✓	Deprecated	Called before calling compute() on the Series renderers
	finishComputeSeries	Series	✓	Deprecated	Called after calling compute() on the Series renderers
	beforeDrawSeries	Series, ISeriesRenderer, IScriptContext	-	✓	Called before rendering Series
	beforeDrawSeries	Series, ISeriesRenderer	✓	Deprecated	Called before rendering Series
	afterDrawSeries	Series, ISeriesRenderer	✓	Deprecated	Called after rendering Series
	beforeDrawSeriesTitle	Series, Label, IScriptContext	-	✓	Called before rendering the title of a Series (only available for PieSeries)
	beforeDrawSeriesTitle	Series, Label	✓	Deprecated	Called before rendering the title of a Series (only available for PieSeries)
	afterDrawSeriesTitle	Series, Label	✓	Deprecated	Called after rendering the title of a Series (only available for PieSeries)

Scope	Function	Arguments	v 1.x	v 2.0	Description
Marker	beforeDrawMarkerLine	Axis, MarkerLine, IScriptContext	-	✓	Called before drawing each marker line in an Axis
	beforeDrawMarkerLine	Axis, MarkerLine	✓	Deprecated	Called before drawing each marker line in an Axis
	afterDrawMarkerLine	Axis, MarkerLine	✓	Deprecated	Called after drawing each marker line in an Axis
	beforeDrawMarkerRange	Axis, MarkerRange	-	✓	Called before drawing each marker range in an Axis
	beforeDrawMarkerRange	Axis, MarkerRange	✓	Deprecated	Called before drawing each marker range in an Axis
	afterDrawMarkerRange	Axis, MarkerRange	✓	Deprecated	Called after drawing each marker range in an Axis
DataPoint	beforeDrawDataPoint	DataPointHint, Fill, IScriptContext	-	✓	Called before drawing each datapoint graphical representation or marker
	beforeDrawElement	DataPointHint, Fill	✓	Deprecated	Called before rendering each datapoint graphical representation defined by the Series and each marker
	afterDrawElement	DataPointHint, Fill	✓	Deprecated	Called after rendering each datapoint graphical representation and each marker
	beforeDrawDataPointLabel	DataPointHint, Label, IScriptContext	-	✓	Called before rendering the label for each datapoint
	beforeDrawDataPoint	DataPointHint, Label	✓	Deprecated	Called before rendering the label for each datapoint
	afterDrawDataPoint	DataPointHint, Label	✓	Deprecated	Called after rendering the label for each datapoint
Axis	beforeDrawAxis	-	Inactive	-	
	afterDrawAxis	-	Inactive	-	
	beforeDrawAxisLabel	Axis, Label, IScriptContext	-	✓	Called before rendering each label on a given Axis

Scope	Function	Arguments	v 1.x	v 2.0	Description
	beforeDrawAxisLabel	Axis, Label	✓	Deprecated	Called before rendering each label on a given Axis
	afterDrawAxisLabel	Axis, Label	✓	Deprecated	Called after rendering each label on a given Axis
	beforeDrawAxisTitle	Axis, Label, IScriptContext	-	✓	Called before rendering the Title of an Axis
	beforeDrawAxisTitle	Axis, Label	✓	Deprecated	Called before rendering the Title of an Axis
	afterDrawAxisTitle	Axis, Label	✓	Deprecated	Called after rendering the Title of an Axis

3. Chart model instances

There are different parameters to access the chart model, this section details the characteristics for each of them.

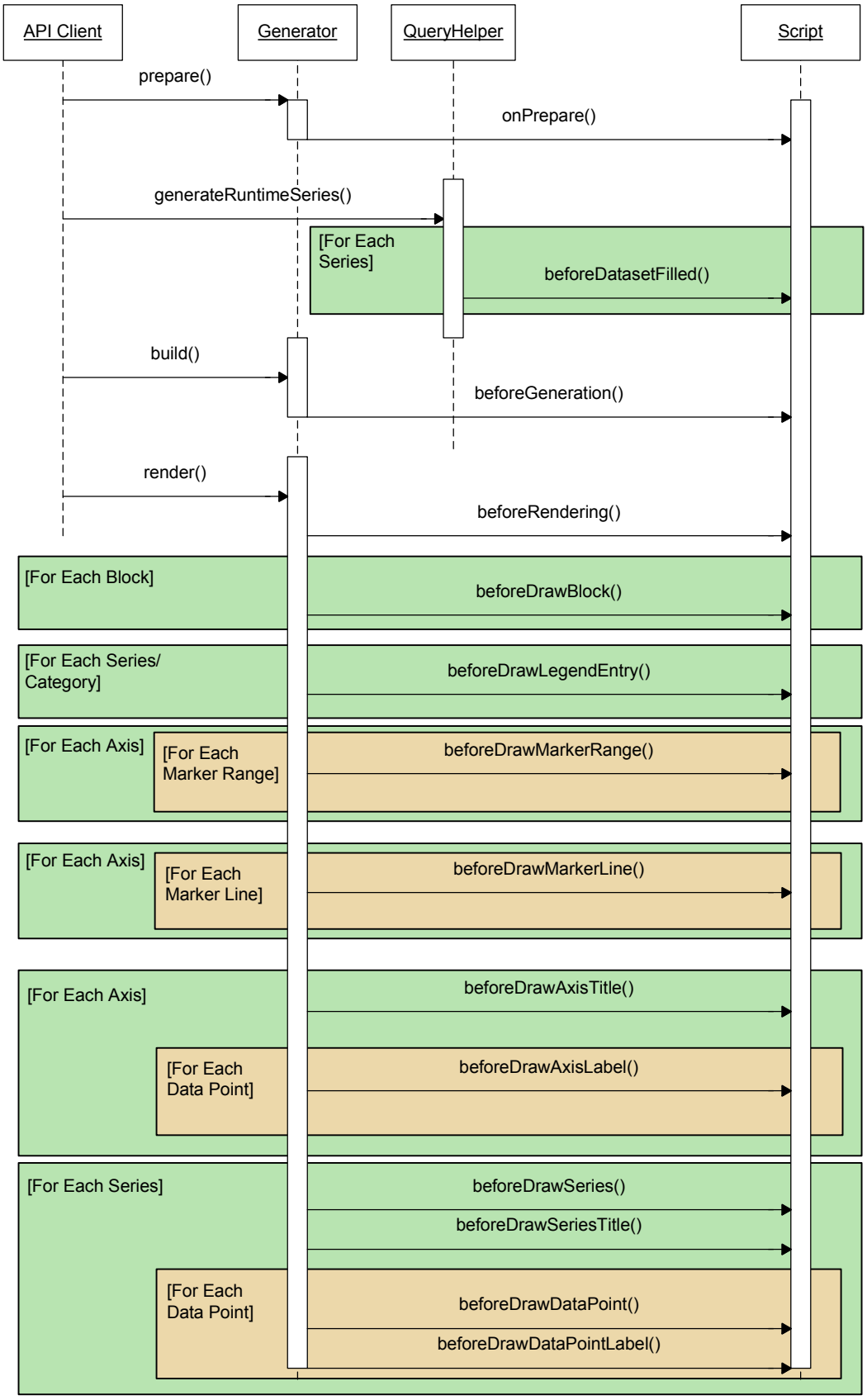
Design Chart : This is the chart design model. It holds all design properties of the chart, any change to it will reflect in all runtime instances of the chart. It is also bound to data (except in onPrepare), so it holds chart-formatted data.. The data is cleared out before being bound again (which can happen inside a repeater control such as a table).

Runtime Chart: this is a deep copy of the chart design model. There is a one-to-one correspondence between runtime chart and chart instances in the report. So if you have a table with a chart in the group header, you will have one design chart, and as many runtime charts as you have groups.

GeneratedChartState: this holds a reference to the runtime chart, the computations, the device renderer, the display server, runtime context, etc…

4. Scripting flow

This sequence diagram shows the flow of the Chart script calls. This diagram is valid both for BIRT or Standalone Charts, the only BIRT specific class being the QueryHelper.


API Client
Generator
QueryHelper
Script
prepare()
onPrepare()
generateRuntimeSeries()
[For Each Series]
beforeDatasetFilled()
build()
beforeGeneration()
render()
beforeRendering()
[For Each Block]
beforeDrawBlock()
[For Each Series/ Category]
beforeDrawLegendEntry()
[For Each Axis]
[For Each Marker Range]
beforeDrawMarkerRange()
[For Each Axis]
[For Each Marker Line]
beforeDrawMarkerLine()
[For Each Axis]
beforeDrawAxisTitle()
[For Each Data Point]
beforeDrawAxisLabel()
[For Each Series]
beforeDrawSeries()
beforeDrawSeriesTitle()
[For Each Data Point]
beforeDrawDataPoint()
beforeDrawDataPointLabel()

5. JavaScript API

The script method's arguments refer to Java classes associated with the runtime model.

e.g. fill is defined by class org.eclipse.birt.chart.model.attribute.Fill

label is defined by class org.eclipse.birt.chart.model.component.Label

axis is defined by class org.eclipse.birt.chart.model.component.Axis

… etc

If a callback JavaScript method is undefined, the internal script handler will not attempt to call it.

The following code snippet illustrates how to register JavaScript in the model:

```
Chart cm = …;
cm.setScript("
function beforeDrawDataPoint(dataPointHints, label)
 { val = dataPointHints.getOrthogonalValue();
   clr = label.getCaption().getColor();
   if (val < 0)
         clr.set(255, 0, 0);
   else
         clr.set(0, 0, 255); }
");
```

This JavaScript method would attempt to set the text color of a rendered data point to red if the orthogonal value being plotted is negative and to blue if the value is zero or positive.

5.1 Global functions (deprecated in v2)

In addition, several global functions/objects are available. Note that all the globals variables are deprecated in v2.

```
Chart getDesignTimeModel( )

Chart getRunTimeModel( )

GeneratedChartState getGeneratedChartState( )

Locale getLocale( )

Object clone(Object) // to clone EMF objects if needed
```

`logger` (a globally accessible object capable of logging values using the default logging framework) e.g.

```
logger.logFromScript("running from script")
```

5.2 External Scripting

In addition, if an existing library of functions contained in an externally written scriptable instance is to be plugged into the chart library for data definition access, this is possible via the Generator's build method as shown:

```
    GeneratedChartState build(IDisplayServer, Chart, Scriptable
scParent, Bounds, RunTimeContext) throws GenerationException
```

Note: The **scParent** argument represents an external scriptable instance that is to be used in conjunction with the chart callback scripts. Hence, library functions defined in **scParent** may be invoked through the chart callback scripts.

This Scriptable instance corresponds to the External context in the Java API.

For v1, the Scriptable is accessible directly as global variables. In v2, this behaviour is deprecated, it should be accessed as the External Context in IChartScriptContext.

5.2.1 Example

Let's take an example where the chart engine is embedded in a custom application (not BIRT) that has a session object. In order to access this session object inside a script, the user creates a class called JavaScriptSession, implementing Scriptable.

```
class JavaScriptSession implements Scriptable
{
            // Define a getUserId() javascript method
            …
}
```

Then the user passes an instance of the JavaScriptSession through the Generator.build() method. In the chart script it is then possible to access the user ID::

```
function beforeDrawDataPoint( hint, fill, context)
{
            var user = getUserId();
            …
}
```

6. Java API (v2)

In Java, an interface IChartItemScriptHandler will need to be implemented. This interface has got all the methods defined in the table for v2.0 (the deprecated JavaScript methods are not available).

An adapter implementation ChartItemScriptHandler is also available to make it easy for the user to only implement the needed methods.

In order to register this Java class in the model, simply put the fully qualified java class name in the setScript() method (such as "mypackage.mycomponent.MyClass"). Here is an example

Chart cm=…

cm.setScript("mypackage.mycomponent.MyClass");

It is the chart engine caller responsibility to make sure the specified class will be available in the class path by the chart engine at run-time. The class will then be

automatically instantiated by reflection, with one instance of the class for each chart runtime instance. If the Chart is embedded in a BIRT report, this will be provided by the BIRT Report Library framework.

7. Chart Engine Preparation API (v2)

A new method prepare will be available in the Generator from v2:

RunTimeContext Generator.prepare(Chart designModel, Object ExternalContext, Locale locale);

This method will perform the following tasks:

1- Create and return a runtime context, necessary for the Generator.build method.

2- Enable the Scripting on the runtimeContext object (internally using ScriptHandler), creating the IChartScriptContext and attaching the ExternalContext to it. The External context is optional and can be null.

3- Call the onPrepare script event function.

Note that this method must be called before Generator.build(), and should only be called once per design model. The call is optional, to ensure upward compatibility of existing Chart API users code.

8. BIRT Scripting integration (v2)

8.1 Designer integration

Inside BIRT, it will be possible to use the Report Designer Script Editor to edit the JavaScript or Java Chart script. The IReportItem interface will have additional methods to allow this integration:

```
/*
* This returns the interface used for scripting
*/
Class getScriptInterface();
/*
* The string content is either inline javascript, or a fully
qualified java class name
*/
void setScript(String);
String getScript();
```

8.2 BIRT Context

The BIRT script context for the report can be passed through the Chart Script using the external context in the Generator.

Here is an example:

```
ReportContext context = …
Generator.instance().build(ids, cmDesignTime, context, bo, rtc )
```

Then the script writer will need to cast the external context to the ReportContext.The Application context can then be accessed through the ReportContext:

For instance

```
public void beforeDrawDataPoint(DataPointHint hint, Fill fill,
IScriptContext context)
{
   ReportContext rct = (ReportContext)context.getExternalContext();
   Object appContext = rct.getAppContext();
…
}
```

8.3 BIRT Script methods

In BIRT, the report items define three main methods: onPrepare, onCreate and onRender. There is no direct correspondence with chart script methods since the generation flow is slightly different, and the chart has its own engine and script engine. Therefore the chart execution flow is independent of other report items.

The chart script methods provide naturally enough granularity for scripting before data binding is done, before the chart is generated, and before it is rendered (respectively beforeDataSetFilled, startGeneration, startRendering). It even provides additional granularity on the chart elements, for each data point being rendered.

8.4 OnPrepare Integration

The interface IReportItemGeneration will introduce a new prepare() method in which the Generator.prepare() method will be called. As a result, the onPrepare() script function will behave in a similar fashion to other report items onPrepare().

9. Scripting use cases

This section details how to use the script methods for typical scripting use cases:

Use Case description	Script method to use	Script description
change the bar color based on its value	beforeDrawDataPoint(DataPointHint, Fill, IChartScriptContext).	Check the data, and change the fill accordingly to the data value.
Custom series values	afterDataSetFilled(Series, DataSet, IChartScriptContext) or beforeGeneration(Chart, IChartScriptContext)	Change some of DataSet values after it has been filled. or Access the runtime Series from the Chart model and change its DataSet.
Custom axis label names, mapping value to string	beforeDrawAxisLabel(Axis, Label, IChartScriptContext)	This is called for each label drawn on the Axis. You can change the label value in it.

Use Case description	Script method to use	Script description
Send email or trigger some external action based on values on a point in the series	beforeDrawDataPoint(DataPointHint, Fill, IChartScriptContext)	Check the DataPointHint data value to perform an external action. In Java you need to make sure you can resolve the external class you are calling, in JavaScript you need to use the external scripting as explained in the document
Show/hide series base on user preference	beforeDrawSeries(Series, ISeriesRenderer, IChartScriptContext) or onPrepare(Chart, IChartScriptContext)	Change the visibility property of the series from a preference value you can get from the context. or Access the series to be changed in the Chart model design, and change its visibility (this has the advantage of being done only one time per design).