raswct Web Client Toolkit

Developer Guide

rasdaman version 9.0

rasdaman Version 9.0 raswct Web Client Toolkit Developer Guide

Preface

Overview

Purpose of the raswct (“rasdaman Web Client Toolkit”) toolkit is to allow developers creating Web user interfaces for displaying data from a raster database.

Implementation

The toolkit is developed in Javascript and uses popular libraries like jQuery. Its structure follows the principle of separating data transmission and processing from the presentation, the two main namespaces reflecting this philosophy:

- **Query namespace** - containing all the classes that can be used to retrieve data from a server, be it a simple HTTP server or a rasdaman server.
- **Widget namespace** - containing all the classes that can be used to display the data in meaningful ways

This document describes how to create widgets and modify them to suit particular purposes. At the end of each widget description an example of use is given. More examples can be found in the `docs/examples` folder in the toolkit package.

Throughout the document, the code fragments will be represented using italic.

Audience

Information in this manual is intended primarily for Web application developers.

Rasdaman Documentation Set

This manual should be read in conjunction with the complete rasdaman documentation set which this guide is part of. The documentation set in its completeness covers all important information needed to work with the rasdaman system, such as programming and query access to databases, guidance to utilities such as the graphical-interactive query tool *rView*, and release notes.

In particular, current restrictions, known bugs, and workarounds are listed in the Release Notes. All documents, therefore, always have to be considered in conjunction with the Release Notes.

The rasdaman Documentation Set consists of the following documents:

- C++ Developer's Guide
- Java Developer's Guide
- Query Language Guide
- Web Client Toolkit Guide
- Installation and Administration Guide
- Error Messages
- rView Guide
- Release Notes

Table of Contents

1 Introduction

1.1 Purpose and Use

This toolkit allows developers to easily create individualized Web interfaces for displaying multi-dimensional raster data. For example, diagrams serve to present 1-D query results, images and a geo Web Map interface serve to display 2-D query results. 3-D displays are under development. All such data can stem from multi-dimensional database contents, such as 1-D extracts from a 4-D climate data set.

Database queries can be hidden behind interactive parameter setting through sliders, gauges, etc., thereby hiding the complexity of the query language to casual users.

Crafting such Web interfaces often is as easy as writing HTML, without resorting to JavaScript, which is the raswct implementation language. That said, all JavaScript is available to advanced developers for designing high-end interactive data interfaces.

1.2 Implementation

The raswct toolkit is developed in Javascript and uses popular libraries, like jQuery. Its structure follows the principle of separating data transmission and processing from the presentation:

- The **Query namespace**, Rj.query, contains all the classes for data retrieval from a server, be it a simple HTTP server or a rasdaman server.
- The **Widget namespace**, Rj.widget, contains all the classes for displaying data in various ways.
- **Utility functions**, gathered in the Rj.util namespace, compensate for Javascript's lack of features in interaction widget basics.

1.3 References

Raswct tutorial material is provided at http://raswct.flanche.net/apps/doc and http://raswct.flanche.net/apps/trainer.

The raswct toolkit is heavily used in the EarthLook geo service standards showcase.

2 Query Namespace

2.1 Base Query

Description

Base class for the query classes. It does nothing on its own, but is a good starting point for any class that wants to have data transport capabilities.

Attributes

Name	Type	Description
- id	Int	Unique identifier of the BaseQuery object
- query	String	The string query that will be executed.

Methods

Name	Para-meters	Return Type	Description
+ getId()		Int	Standard getter for the **id** attribute.
+ getQuery()		String	Standard getter for the **query**

			attribute.
+ setQuery(query)	query: string	Base-Query	Sets a new value to the query and returns the BaseQuery object on which the operation has been performed.

Notes

This class should not be used on its own, it is provided only as a means for exposing new ways of querying the data from the server. Please note that although there is no mechanism in JavaScript to enforce this any child class **should implement a transport method** (see class QueryExecutor for more details)

2.2 Select Query

Description

The SelectQuery class provides an abstraction over the select query sent to the server. It allows widget developers to easy modify queries by adding new variables to the query that can be replaced with meaningful values at the transmission time.

Attributes

Name	Type	Description
- url	String	The url to the service that can execute the raster query
- query	String	The initial query attached to this object.
- variables	Object	The variables that are attached to the query, each of them can be modified using setVariable.

Methods

Name	Para-meters	Return Type	Description
+ getVariable(variable)	String	String	Returns the value attached to the given variable.
+ setVariable(variable, value)	variable: string, value: mixed	[[Select Query]]	Assigns a value to a variable and returns the query object the action is being performed on.
+ replaceVari-ablesInQuery()		String	Replaces the variables from the query and returns its new value.
+ transport()		Object	Return the query in a transport format, as requested by the QueryExecutor specs.

Examples

The following code snippet creates a raster query:

```
var rasQuery = new Rj.Query.SelectQuery(
    "http://example.org/raster_service",
    "SELECT (x.red > $red, x.green > $green)
     FROM collection as x", {});
rasQuery.setVariable( "$red", 25);
rasQuery.setVariable( "$green", 50);
rasQuery.setVariable( "$red",
                      rasQuery.getVariable("$red") + 10);
console.log( rasQuery.replaceVariablesInQuery() );
```

This will output SELECT jpeg(x.red > 35, x.green > 50) FROM collection as x.

Note

In most cases **you will not need to use** the replaceVariablesInQuery() method as all transport methods call it automatically.

2.3 URL Query

Description

The UrlQuery class provides an abstraction over queries sent to a server.

It allows seamless request-response transactions to a http server.

Attributes

Name	Type	Description
- baseUrl	String	The URL to start from.
- type	String	The request type (e.g. GET || POST)
- parameters	Array	An array of parameters to be used.

Methods

Name	Para-meters	Return Type	Description
+ addParameter(parameter)	parameter: object	Void	Adds a parameter to the request.
+ removeParameter (parameter)	parameter: object	Void	Removes a parameter from the request.
+ transportGet()		String	Returns a formatted get string URL.
+ transport()		Object	Implements the transport method required for all objects that are handled by an executor.

Examples

The following code snippet creates an UrlQuery and modifies its parameters. Please see QueryExecutor for the data retrieval procedure

```
var exQuery = new Rj.Query.UrlQuery(
    "http://example.org/data_service/",
    Rj.Constants.UrlQuery.POST, {
        param1 : "some value",
        param2 : "some value 2"
});
exQuery.addParameter( "param3", "value 3" );
exQuery.removeParameter( "param2" );
```

We now have a query object that retrieves data from example.org/data_service via a POST request sending several parameters (param1 and param3).

2.4 Query Executor

Description

The QueryExecutor is a singleton class that is responsible for the communication with the server. It can receive queries from any BaseQuery descendants and then send them to the server through an HTTP request responding to the requester with the result object received from the server. The requests are done asynchronous in a non-blocking way so that multiple widgets can request queries from the server without waiting one for the other.

Attributes

Name	Type	Description
- query	Object	An object of type descendant of BaseQuery Class.

Methods

Name	Para-meters	Return Type	Description
+ sendRequest(data, handler)	Object data, Function handler	none	send the request to the server containing the query and calls the handler function provided with an array of results. Please note that the function doesn't return a result, but calls the handler once the server has responded

Examples

```
var rasQuery = new Rj.Query.SelectQuery(
    "http://example.org/raster_service",
    "SELECT jpeg(x.red > $red, x.green > $green)
```

```
     FROM collection as x", {} );
rasQuery.setVariable( "$red", 25 );
rasQuery.setVariable( "$green", 50 );
var executor = Rj.Query.QueryExecutor( rasQuery );
executor.callback( function(response){ // here response
    console.log( response );            // from server
});
```

Note that the executor is a deferrable object, similar to Future objects in Java, so that means you will get the result asynchronously, e.g. you can register a handle that will be executed when the data is retrieved from the server.

3 Widget Namespace

3.1 Base Widget

Description

The base widget is a wrapper class that has to be extended by any widget that want to interact with the system. It wraps an existing widget from a library like jQuery UI or any other a developer might need, providing it with a simple event-communication system and with a BaseQuery that can modify the database results.

Atrributes

Name	Type	Description
- widget	Object	The library widget that is being used, e.g., jQuery.ui.slider or google.Charts.Visualizat-ionChart
- query	BaseQuery	The query that the widget wants to manipul-ate, any descendant of BaseQuery can be

		used
- listeners	Array	An array of events that the widget wants to listen to. Each element has to be defined as an object of form {eventName : handlerFunction}
- selector	CSS3 / XPath	A CSS3/XPath selector used as indentifier for the position of the widget.

Methods

Name	Para-meters	Return Type	Description
+ renderTo(node)	DOM-Object node	None	Renders the widget in the node provided. Can be anything ranging from body to a specific div
+ show()	none	None	Make the widget visible. By default widgets are rendered invisible
+ hide()	none	None	Make the widget invisible
+ fireEvent(eventName, bubble, args)	String event-Name, boolean bubble, Array args	None	Fires a defined event, with the arguments specified in the third parameter. If bubble is a set to true, the event will be propagated upwards and any widgets that registered for the event will be notified
+ addListener(eventName, handler)	String event-Name, function handler	None	Registers a new handler for a specific event
+ removeListener(eventName)	String event-Name	None	Removes the handler of this widget for the event, the widget will not be notified of these event anymore

Examples

This is a base class for widgets so it shouldn't be initialized or used except for extending the current widget system.

3.2 Input Widget

Description

InputWidget is a simple grouper class that helps better define the relationships between widgets.

Attributes

Name	Type	Description
- value	string	The value displayed in widget.

Methods

Name	Para-meters	Return Type	Description
+ getValue()		string	Standard getter for the **value** attribute.
+ setValue(value)	value : string	Void	Standard setter for the **value** attribute.

Examples

This is a base class for widgets so it shouldn't be initialized or used except for extending the current widget system.

3.3 Text Widget

Description

Defines a widget which allows the user to input text queries.

Attributes

Name	Type	Description
- rows	Int	The number of rows the widget has.
- cols	Int	The number of columns the widget has.
- submitValue	String	The value of the submit button.
- value	String	The value displayed in the widget.

Examples

Js code:

```
var txtWidget = new Rj.Widget.TextWidget();
txtWidget.renderTo("#text-widget-example");
txtWidget.setValue("Hello World");
```

Html code:

```
<div id="#text-widget-example">
  The widget will be rendered here
</div>
```

3.4 Slider Widget

Description

Defines an abstraction of a widget which allows the user to use a multiple level slider.

Attributes

Name	Type	Description
- slideLevel	Int	The current level to which the slider is.

Methods

Name	Para-meters	Return Type	Description
+ getSlideLevel()		int	Standard getter for the **slide-Level** attribute.
+ setSlideLevel(slideLevel)	Slide-Level: int	Void	Standard setter for the **slideLevel** attribute.

Examples

The following example will display a slider with values from 1000 to 11000 with a step size of 500. When the slider is moved a message will be printed to the console.

```
var thresholdSlider = new Rj.Widget.SliderWidget(
    1000, 11000, Rj.Constants.WidgetSlider.VERTICAL,
    500, 8000); //min, max, orientation, step size, default
thresholdSlider.renderTo( "#thr" );
// Using changevalue event to detect slider modification
thresholdSlider.addListener( "myAppName", "changevalue",
    function(value){
        console.log( "This is the new value: "
                    + value.toString();
});
```

3.5 Knob Widget

Description

Defines a knob widget.

Attributes

Name	Type	Description
- min	Int	The lower bound of the knob.
- max	Int	The higher bound of the knob.
- value	Int	The initial value of the knob.
-reverse	Bool	If true, the values are distributed backwards (from 360 degrees to 0 degrees).
- snap	Int	The number of degrees from which the knob is snapped to 0.

Methods

Name	Para-meters	Return Type	Description
+ getValue()	None	Int	Standard getter for the **value** attribute.

Examples

The following code creates a Knob object within a * * element:

```
var knob = new Rj.Widget.Knob(0, 10, 5, false, 20);
knob.renderTo("knob");
```

3.6 Output Widget

Description

OutputWidget is a simple grouper class that helps better define the relationships between widgets.

Attributes

Name	Type	Description
- query	Object	The query object used for getting the results displayed by the widget.
- widget		Identifier of the widget.

Methods

Name	Para-meters	Return Type	Description
+ refresh()			Stub method for the extending classes.

3.7 Map Widget

Description

Defines an a widget used for displaying maps composed of several layers.

Implementation is based on the OpenLayers library, see http://open-layers.org/.

Attributes

Name	Type	Description
- map	Object	The raw OpenLayers map.

Methods

Name	Para-meters	Return Type	Description
+ getRawMap()		Object	Returns the raw OpenLayers Map.
+ addLayers()		Void	Adds layers to the map.

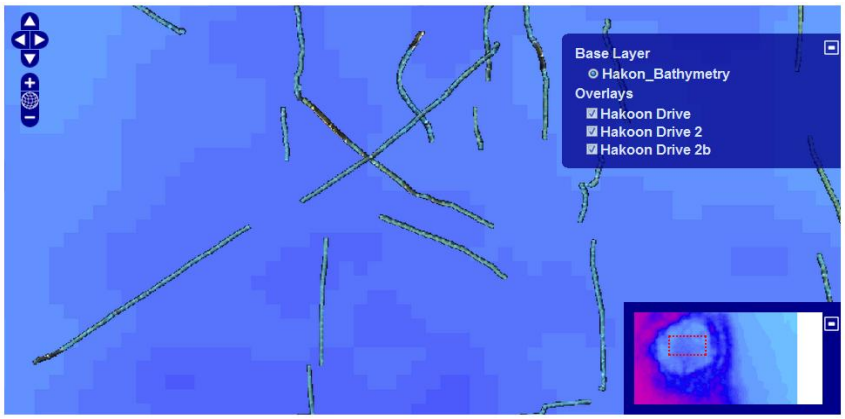
Examples

The following code will display a map with one layer:

```
//Define the map widget and the coordinate system
var map = new Rj.Widget.MapWidget({
    projection : "EPSG:32633",
    maxExtent : new OpenLayers.Bounds( 489750, 7988500,
                                       492750, 7990000 ),
    tileSize : new OpenLayers.Size(500, 500),
    numZoomLevel : 4
});
//Define a new base layer for the map.
// Any WMS service url will do:
var HakoonBathymetryLayer = new Rj.Widget.LayerWidget(
    "Hakon_Bathymetry",
    ”http://212.201.49.173:8080/rasogc/rasogc”, {
        layers: 'Hakon_Bathymetry',
        styles: 'colored',
        format : "image/png",
        version : "1.1.0",
        exceptions : 'application/vnd.ogc.se_xml',
        customdem : 'minLevel,maxLevel,T'
    },{
        transitionEffect : 'resize'
    });

//Add this layer to the map
map.addLayers([HakoonBathymetryLayer]);
//... and render it to the #maps div
map.renderTo("#maps");
```

Visual appearance:



3.8 Diagram Widget

Description

Defines a widget used as a base for all charts.

Attributes

Name	Type	Description
- title	String	The title of this diagram.
- xAxisTitle	String	The title of the X axis.
-yAxisTitle	String	The title of the Y axis.

Methods

Name	Para-meters	Return Type	Description
+ setData	data: Array	Boolean	Sets the data attribute and fires two events: datapreload - before the data is loaded datapostload - fired once the data is loaded
+ getData		Array	Returns the data assigned to the widget
+ addData-Series	series: Array name: String	Int	Adds a data series to the diagram as an array of form [[x,y] , [x1, y1]] and returns an index of the new data serie
+ removeData-Series	index: Int	Int	Removes a series from the diagram. The index is the same as the one returned by addDataSeries
+ configure	cfg: Object	Object	Configures the chart object before rendering. All subclasses should override this method in order to add their specific configurations.

+ renderTo	selector: String, cfg: Object		Renders the widget to a given DOM element.

Note

This is a base class for graphs so it shouldn't be initialized or used except for extending the current graph system.

3.9 Linear Diagram

Description

Defines a widget used for displaying linear graphs.

Methods

Name	Para-meters	Return Type	Description
+ configure	cfg: Object	Object	Configures the chart object before rendering..

Examples

JS Code:

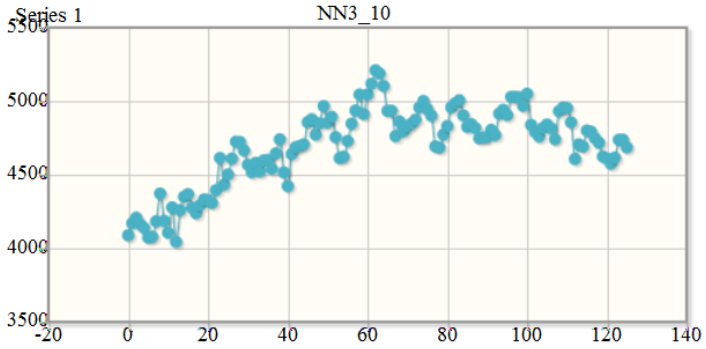
```
//Initialize query - we are using an URL Query object
var source = "NN3_10";
var query = new Rj.Query.UrlQuery( "wcpsParser.php",
    'GET', {
        'coverageId': source
    });
//Create widget
var diagram = new Rj.Widget.LinearDiagram( query,
    "#chartPlace", source );
// Get diagram axis and labels before data is rendered
// by listening to the datapreload event
diagram.addListener( 'wcps','datapreload',
    function(response){
        // Check if any errors occurred,
        // and if so display a nice error message
        if(response.error){
            $("body").append( "<div id='dialog'>"
                              + response.error + '</div>');
            $( "#dialog" ).dialog({
                modal : true,
                title : 'Parse Error'
            }).show();
            throw "Error while processing the data";
```

```
            }
            var values = [];
            for(var i = 0; i < response.data.length; i++){
                values.push( [i, parseInt(response.data[i],
                                10)]);
            }
        //Configure the widget axes
        this.configure({
            axes : {
                xaxis:{ title : response.domainInfo.axisLabel },
                yaxis: { title : "Values" }
            }
        });
        return {
            data : [values]
        };
});
// load data and render widget
diagram.loadData(true);
```

HTML Code:

```
<div id='chartPlace' style='width:600px; height:500px;'>
    <!-- The chart will go here -->
</div>'
```

Visual appearance:



3.10 Area Diagram

Description

Defines a widget used for displaying area graphs.

Methods

Name	Para-meters	Return Type	Description
+ configure	cfg: Object	Object	Configures the chart object before rendering..

Examples

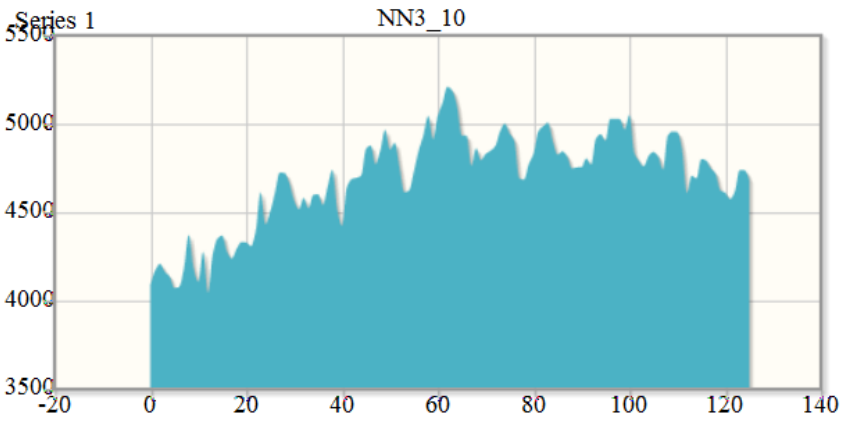
JS Code:

```
var source = "NN3_10"
//Initialize query - we are using an URL Query object
var query = new Rj.Query.UrlQuery("wcpsParser.php", 'GET', {
    'coverageId': source
});
//Create widget
var diagram = new Rj.Widget.AreaDiagram( query,
    "#chartPlace", source);
// Get diagram axis and labels before data is rendered
diagram.addListener( 'wcps','datapreload',
    function(response){
        var values = [];
        for(var i = 0; i < response.data.length; i++){
            values.push( [i, parseInt(response.data[i],
                        10)]);
        }
        //Configure the widget labels
        this.configure({
            axes: {
                xaxis: {
                    title: response.domainInfo.axisLabel },
                yaxis: { title : "Values" }
        }
    });
    return { data : values };
});
//Load the data and render the widget
diagram.loadData(true);
```

HTML Code:

```
<div id='chartPlace' style='width:600px; height:500px;'>
    <!-- The chart will go here -->
</div>'
```

Visual appearance:



3.11 Scatter Diagram

Description

Defines a widget used for displaying scattered graphs.

Methods

Name	Para-meters	Return Type	Description
+ configure	cfg: Object	Object	Configures the chart object before rendering..

Examples

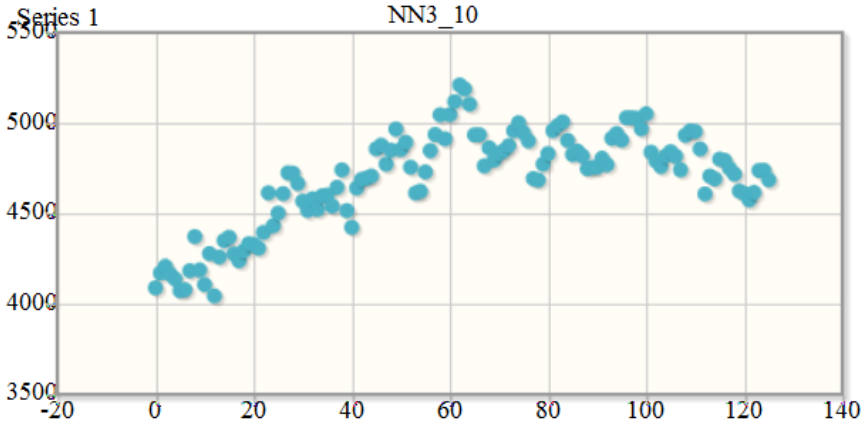
JS Code:

```
// Initialize query - we are using an URL Query object
var source = "NN3_10"
var query = new Rj.Query.UrlQuery("wcpsParser.php", 'GET', {
    'coverageId': source
});
// Create widget
var diagram = new Rj.Widget.ScatterDiagram( query,
    "#chartPlace", source);
// Get diagram axis and labels after data is loaded
// by listening to datapreload event
diagram.addListener( 'wcps','datapreload',
    function(response){
        var values = [];
        for(var i = 0; i < response.data.length; i++){
            values.push( [i, parseInt(response.data[i],
                         10)]);
        }
        // Configure widget labels
        this.configure({
            axes : {
                xaxis: {
                    title : response.domainInfo.axisLabel
                },
                yaxis : { title : "Values" }
            }
        });
        return { data : values };
    });
diagram.loadData(true);
```

HTML Code:

```
<div id='chartPlace' style='width:600px; height:500px;'>
   <!-- The chart will go here -->
</div>'
```

Visual appearance:



3.12 Gauge Widget

Description

Defines a circular gauge widget.

Attributes

Name	Type	Description
- value	Int	The initial value displayed.
- labelSuffix	String	The string displayed after the label value.
- taco	Bool	Sets a custom display.

Methods

Name	Para-meters	Return Type	Description
+ getValue()		Int	Standard getter for the **value** attribute.
+ setValue(value)	value: Int	Void	Standard setter for the **value** attribute.

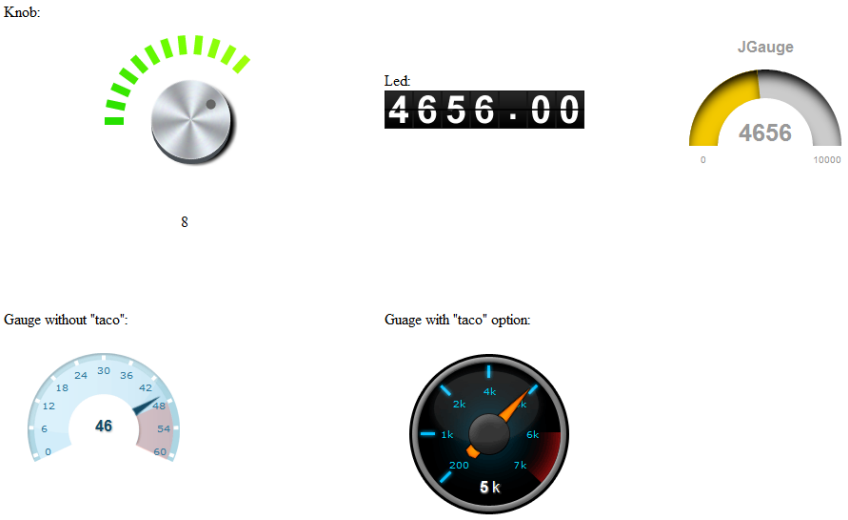
Examples

The following example will display a gauge within a `<div id = "gauge"></div>` element.

```
var gauge = new Rj.Widget.Gauge(null,24);
gauge.renderTo("gauge");
```

Gauge overview

Below the gauges are listed which are available currently; they are described in the subsequence subsections.



3.13 JGauge Widget

Description

Defines a semi-circular gauge widget.

Attributes

Name	Type	Description
- title	String	The title of the widget.
- label	String	The label of the widget.
- min	Int	The lower bound of the displayed values.
- max	Int	The upper bound of the displayed value
- show-MinMax	Bool	Shows or hides the bounding values.
- value	String	The initial value displayed.
- width	Float	The scale at which the widget is displayed. 1 is the reference point.
- shadow	Bool	Shows or hides the shadow of the upper part of the widget.
- color	String	The background color of the widget.
- titleColor	String	The color of the title.
- valueColor	String	The color of the value.
- labelColor	String	The color of the label.

Methods

Name	Para-meters	Return Type	Description
+ getValue()		int	Standard getter for the **value** attribute.
+ setValue(value)	value: int	Void	Standard setter for the **value** attribute.

Examples

The following example will display a JGauge object within a `<div id = ”jgauge”></div>` element.

```
var jGauge = new Rj.Widget.JGauge( “JGauge”, “Degrees”, 0,
    180, 90, 1, true, #fff, #000, #000);
jGauge.renderTo("jgauge");
```

3.14 Led Widget

Description

Defines a led counter widget.

Attributes

Name	Type	Description
- value	Float	The initial value displayed.
- numIntegral-Digits	int	The number of digits of the display.
- numFractional-Digits	Bool	The number of fractional digits to display.

Methods

Name	Para-meters	Return Type	Description
+ getValue()		Float	Standard getter for the **value** attribute.
+ setValue(value)	value: Float	Void	Standard setter for the **value** attribute.

Examples

The following example will display a LED within a `<div id = ”led”></div>` element.

```
var led = new Rj.Widget.Led(100.54, 3, 2);
led.renderTo(“led”);
```