
# Rasdaman Web Client Toolkit
## Developer Guide

# Preface

## *Introduction*

The main purpose of this toolkit is to allow developers to create user interfaces for displaying data from a raster database.
The toolkit is developed in Javascript and uses popular libraries like jQuery to achieve its goals.

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

The next pages will describe how you can create widgets and modify them to suit your purposes. At the end of each widget description an example of use will be 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.

## Table of contents

# Query Namespace

## *1.1 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 | Parameters | Return Type | Description |
|---|---|---|---|
| + getId() | | Int | Standard getter for the **id** attribute. |
| + getQuery() | | String | Standard getter for the **query** attribute. |
| + setQuery(query) | query: string | BaseQuery | Sets a new value to the query and returns the BaseQuery object on which the operation has been performed. |

**Examples**

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 js to

enforce this any child class **should implement a transport method**(see QueryExecutor for more details)

### *1.2 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 | Parameters | Return Type | Description |
|---|---|---|---|
| + getVariable(variable) | String | String | Returns the value attached to the given variable. |
| + setVariable(variable, value) | variable: string, value: mixed | SelectQuery | Assigns a value to a variable and returns the query object the action is being performed on. |
| + replaceVariablesInQuery() | | 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 would output SELECT jpeg(x.red > 35, x.green > 50) FROM collection as x. Please note that in most cases **you will not need to use** the replaceVariablesInQuery method as all transport methods call it automatically.

### *1.3 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 | Parameters | 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, param3).

### *1.4 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 | Parameters | 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){//We have the response from the
server here
        console.log(response);
});
```

Please 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.

# Widget Namespace

## *2.1 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.VisualizationChart |
| - query | BaseQuery | The query that the widget wants to manipulate, 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 | Parameters | Retu rn Type | Description |
|---|---|---|---|
| + renderTo(nod | DOMObject node | none | Renders the widget in the node provided. Can be anything ranging from body to a specific |

| | | | |
|---|---|---|---|
| e) | | | 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 eventName, 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 eventName, function handler | none | Registers a new handler for a specific event |
| + removeListener(eventName) | String eventName | 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.

### *2.2 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 | Parameters | 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.

### *2.3 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>
...
```

## *2.4 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 | Parameters | Return Type | Description |
|---|---|---|---|
| + getSlideLevel() | | int | Standard getter for the **slideLevel** attribute. |
| + setSlideLevel(slideLevel) | slideLevel: int | Void | Standard setter for the **slideLevel** attribute. |

**Examples**

The following example will dipslay 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 the changevalue event to detect modification to the slider
thresholdSlider.addListener("myAppName", "changevalue",
function(value){
        console.log("This is the new value: " + value.toString();
});
```

## *2.5 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 | Parameters | Return Type | Description |
|---|---|---|---|
| + refresh() | | | Stub method for the extending classes. |

## 2.6 Map Widget

### Description

Defines an a widget used for displaying maps composed of several layers.
The implementation is based on the OpenLayers library <http://http://openlayers.org/>

### Attributes

| Name | Type | Description |
|---|---|---|
| - map | Object | The raw OpenLayers map. |

### Methods

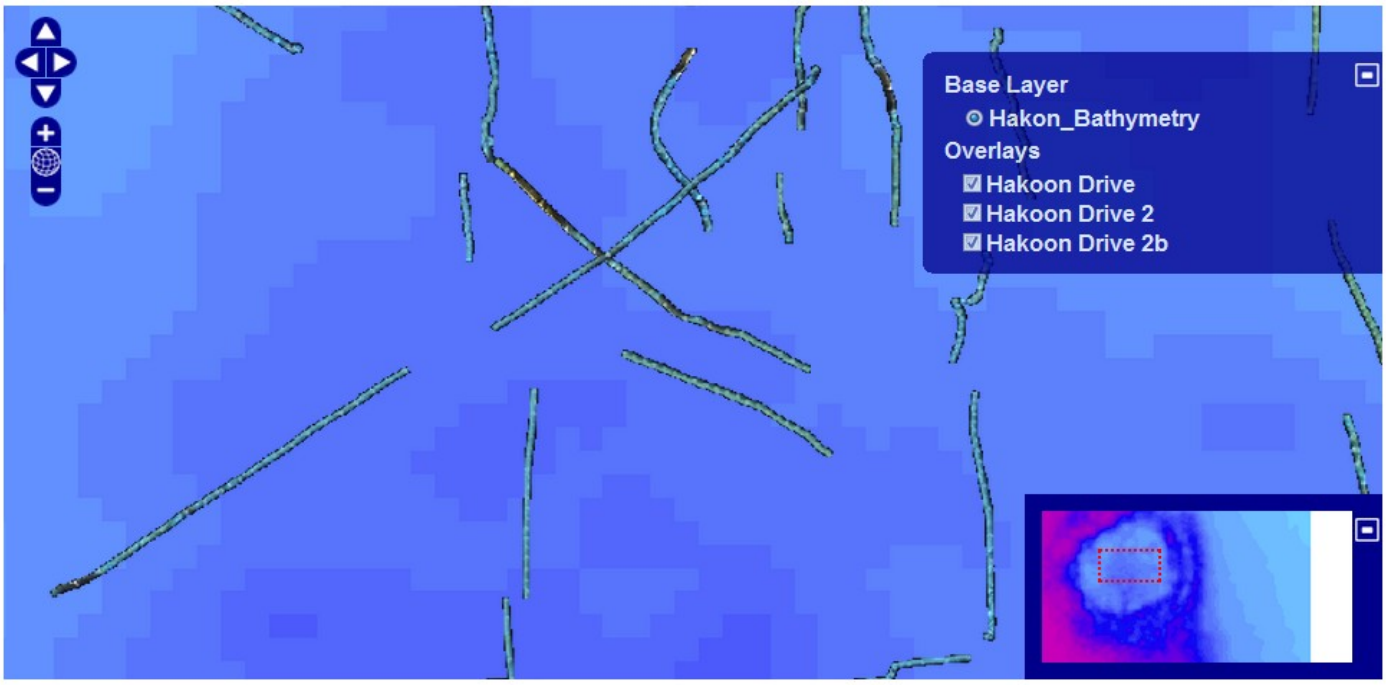
| Name | Parameters | Return Type | Description |
|---|---|---|---|
| + getRawMap() | | Object | Returns the raw OpenLayers Map. |
| + addLayers() | | Void | Adds layers to the map. |

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




### *2.7 Diagram Widget*

**Description**

Defines a widget used as a base for all charts.

**Attributes**

| Name | Type | Description |
|---|---|---|
| - query | Object | The Query that is used to retrieve the data. |
| - data | Array | If static data needs to be used, it can be initialized here. |
| - selector | String | Any valid CSS3 or xPath selector that will identify the div in which the graph is placed. |
| - 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 | Parameters | Return Type | Description |
|---|---|---|---|
| + setData | data: Array | Boolean | Sets the data attribute and fires two events: datapreload - before the data is loaded<br>datapostload - fired once the data is loaded |
| + getData | | Array | Returns the data assigned to the widget |
| + loadData | render: Boolean | Void | Loads the data by executing the query. If render is true it will render the collected data |
| + 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. |

**Examples**

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

### *2.8 Linear Diagram*

**Description**

Defines a widget used for displaying linear graphs.

**Methods**

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

**Examples**

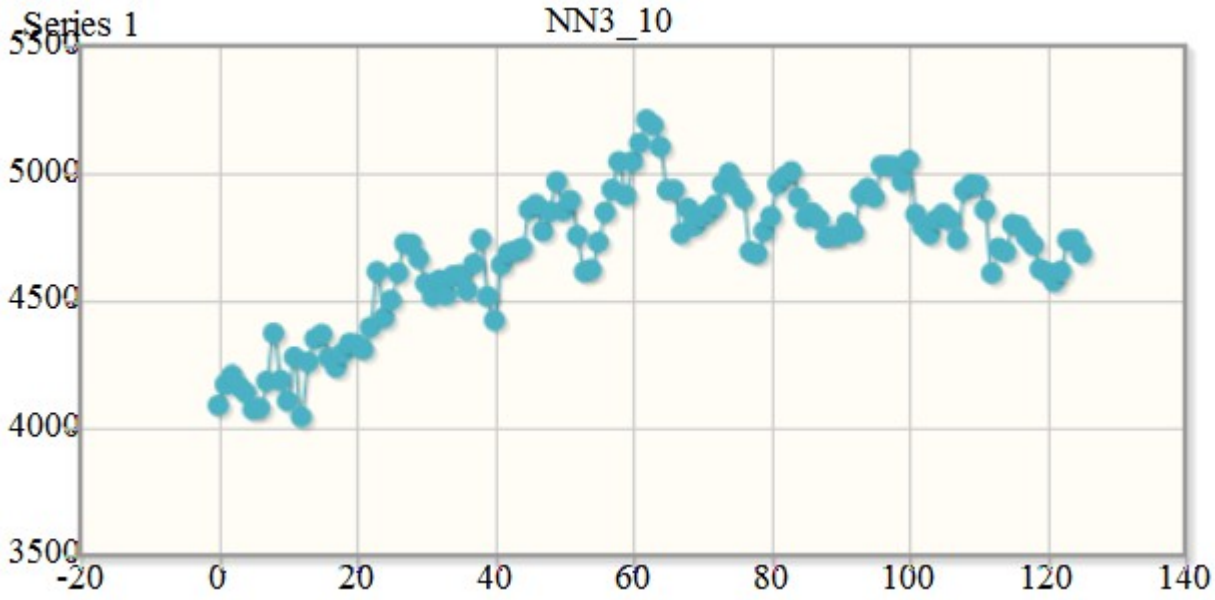
JS Code:

```
//Initialize the query - we are using an URL Query object
   var source = “NN3_10”;
   var query = new Rj.Query.UrlQuery("wcpsParser.php", 'GET', {
     'coverageId': source
   });
   //Create the widget
   var diagram = new Rj.Widget.LinearDiagram(query, "#chartPlace",
source);

   // Get the diagram axis and labels before the 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 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>'
```




### *2.9 Area Diagram*

**Description**

Defines a widget used for displaying area graphs.

**Methods**

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

**Examples**

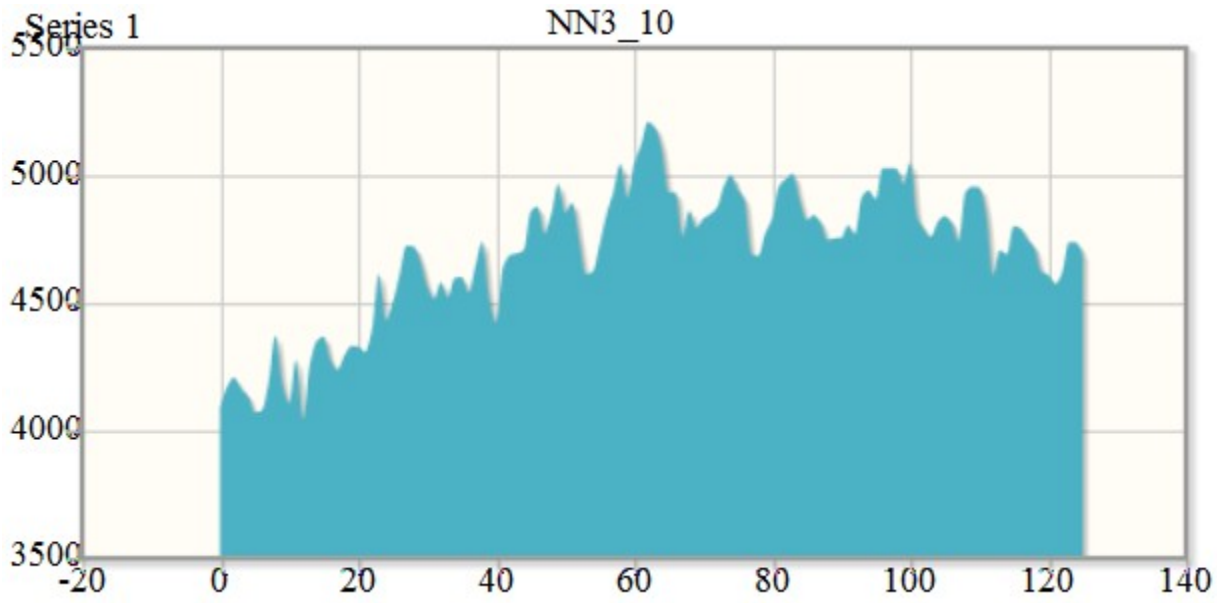
JS Code:

```
var source = "NN3_10"
//Initialize the query - we are using an URL Query object
```

```
    var query = new Rj.Query.UrlQuery("wcpsParser.php", 'GET', {
      'coverageId': source
    });
    //Create the widget
    var diagram = new Rj.Widget.AreaDiagram(query, "#chartPlace",
source);

    // Get the diagram axis and labels before the 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>'
```




### *2.10 Scatter Diagram*

**Description**

Defines a widget used for displaying scattered graphs.

**Methods**

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

**Examples**

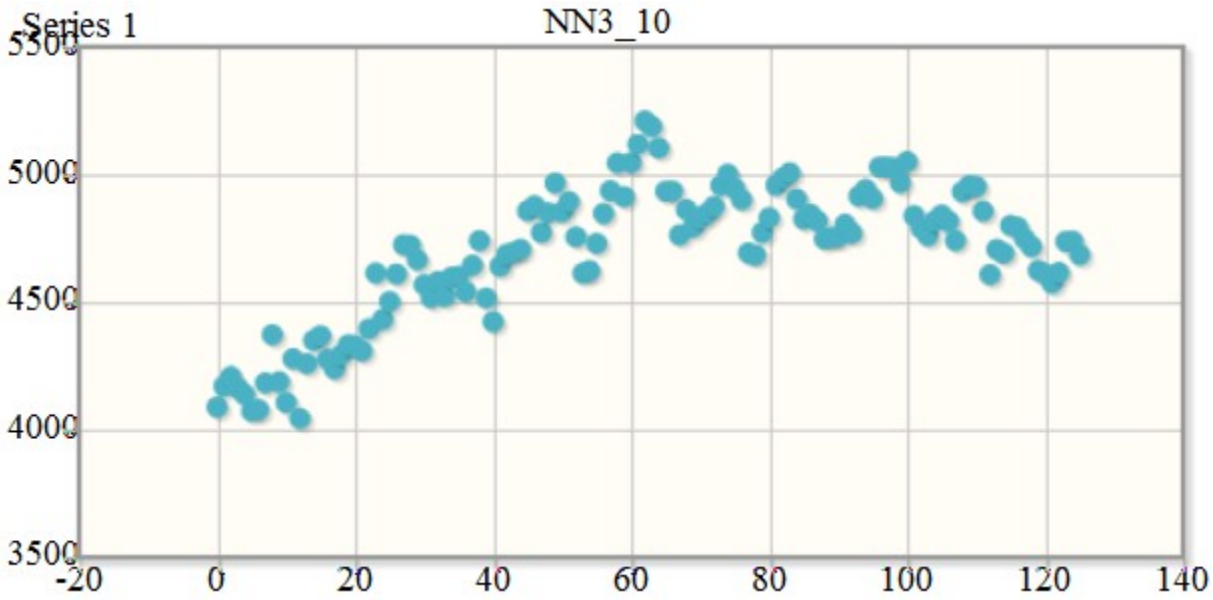
JS Code:

```
//Initialize the query - we are using an URL Query object
    var source = “NN3_10”
```

```
    var query = new Rj.Query.UrlQuery("wcpsParser.php", 'GET', {
      'coverageId': source
    });
    //Create the widget
    var diagram = new Rj.Widget.ScatterDiagram(query, "#chartPlace",
source);

    // Get the diagram axis and labels after the data is loaded by listening
to the 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 the 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>'
```


Series 1
NN3_10
5500
5000
4500
4000
3500
-20
0
20
40
60
80
100
120
140