Tutorial: Implementing the *HelloWorld* component

This tutorial shows how to create a very simple component *`HelloWorld`*, using jQuery and following the guidelines defined by the Biojs specification.

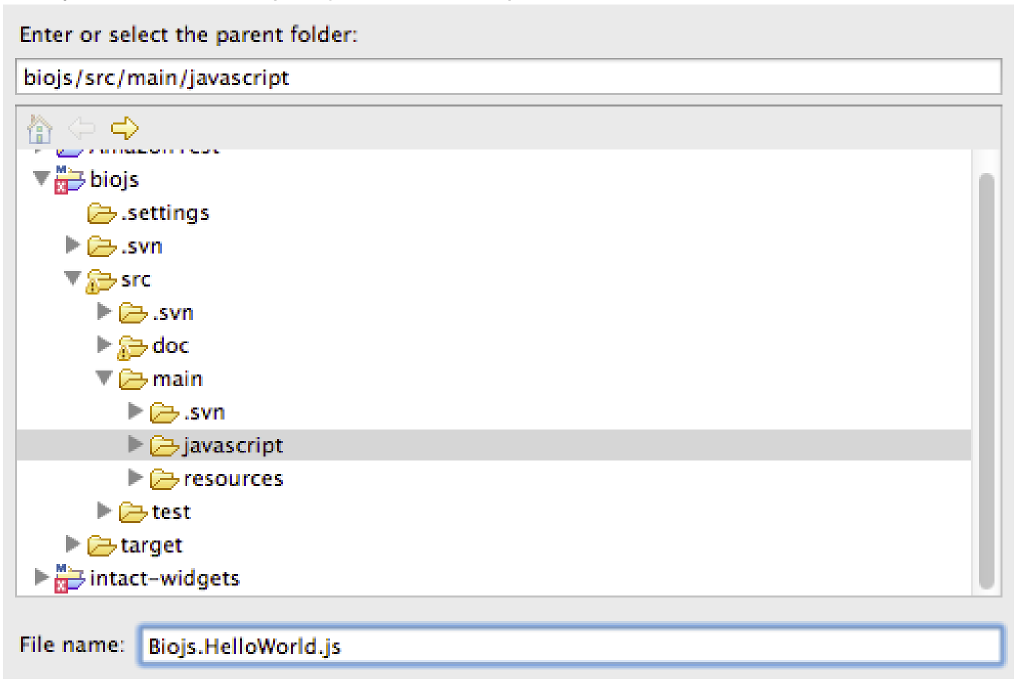
Requirements and recommendations

To follow this tutorial you will need to have the source code downloaded already (if you don't have it, refers to the tutorial Install the registry locally). A web server like Apache is not needed in this tutorial but it is recommended. We suggest you to get XAMPP, an easy to install Apache Distribution for Linux, Windows, Mac OS X, and Solaris.

Steps:

1. In Biojs we like to collect a list of requirements for a component before to start any development. This is the template we normally use, and here you have an example of the requirements we collected for the *`HelloWorld`* component.

2. Create an empty file named *`Biojs.MyHelloWorld.js`* into directory *`biojs/src/main/javascript`*. *Note: Biojs.HelloWorld.js already exists in the repository as an example, please create your own file* *`Biojs.MyHelloWorld.js`*



Enter or select the parent folder:

biojs/src/main/javascript

File name: Biojs.HelloWorld.js

4. Create another file for style settings named *biojs.MyHelloWorld.css* into directory *biojs/src/main/resources/css* containing a class named *MyHelloWorld.* Paste the following style settings in there:

```
/* Use this class to apply all the style settings you need for your component
   In this case, will be applied following settings for the main div container
*/

div.MyHelloWorld {
   font-size: 36px,
   text-align: center,
   vertical-align:middle,
   display: table-cell,
   width: 597px,
   height: 300px
}

/* Any other element inside your component, should be styled by trough
   filtering up desired children of main container.
   For example, lets filter all the span tags inside MyHelloWorld:
*/

div.MyHelloWorld span {
   /* Your style settings here */
}
```

NOTE: This step is highly recommended as a good practice to separate the style settings from the JS source code. In this sense, deployers could adapt the style of the component with the target Web page/site in a easy way.

5. Paste the following skeleton code on it (it was taken from [Biojs specification]() section 6.3.1):

```
Biojs.MyHelloWorld = Biojs.extend ({

  constructor: function (options) {
     /* Your constructor code here

        Note: options provided on instantiation time overrides the
        default values in this.opt, automatically; i.e. ‘options’
        argument refers to the provided values and ‘this.opt’
        refers to the  the overridden options. For more details,
        go to section 6.3.2 in the spec. doc. */
  },

  opt: {
  /* Target DIV
     This mandatory parameter is the identifier of the DIV tag where the
```

```
   component should be displayed. Use this value to draw your
   component into. */
   target: "YourOwnDivId",

/* Component Options
   These options defines the input data for your component.
   Must have a default value for each one. Note that, either some or
   all of values might be replaced by the constructor using the values
   provided in instantiation time.

   Define your own options here following the next syntax:
      <option1>: <defaultValue1>,
      <option2>: <defaultValue2>,
      :
      .
      <optionN>: <defaultValueN> */
},

eventTypes: [

   /* Event Names
      The parent class Biojs build the event handlers automatically
      with the names defined here. Use this.raiseEvent(<eventName>,
      <eventData>) for triggering an event from this component. Where,
      <eventName> is a string (defined in eventTypes) and <eventData> is
      an object which should be passed to the registered listeners.

      Define your event names following the syntax:
        “<eventName1>”,
        “<eventName2>”,
           :
           .
        “<eventNameN>”
    */
],

/* Your own attributes

   _<attrName1>: <defaultValueAttr1>,
   _<attrName2>: <defaultValueAttr2>,
      :
      .
   _<attrNameN>: <defaultValueAttrN>,

   Example:
   _PI: 3.1415, */

/* Your own ‘PUBLIC’ methods

   <methodName1>: function (<argsMethod1>) {<codeOfMethod1>},
   <methodName2>: function (<argsMethod2>) {<codeOfMethod2>},
      :
        .
    <methodNameN>: function (<argsMethodN>) {<codeOfMethodN>}

    Example:
    square: function(number) { return number*number }

    Your own 'PROTECTED' methods

    Javascript doesn't provides visibility mechanism for class members.
    Use character '_' to identify the private members of your component.
    For example: '_initialize'.

    NOTE: use this.base(arguments) to invoke parent's method if apply.

*/
});
```

5. Give the name MyHelloWorld to the component

```
 // Syntax: Biojs.<ComponentName> = Biojs.[<ParentComponentName>.]extend ({
Biojs.MyHelloWorld = Biojs.extend ({
```

6. Declare the options according to the defined ones in requirements. Set both name and default value for each option in the ***opt*** object.

```
 opt: {
    target: "YourOwnDivId",
    fontFamily: '"Andale mono", courier, monospace',
    fontColor: "white",
    backgroundColor: "#7BBFE9",
    selectionFontColor: "black",
    selectionBackgroundColor: "yellow"
  },
```

7. Declare the events according to the defined ones in requirements. Set the event name list in the array ***eventTypes***. In this case, we have just one event named *"onHelloSelection"*. Note that you need do nothing additional to define the events because the event handlers are built automatically on instantiation.

```
 eventTypes: [
     "onClick"
  ],
```

8. Lets build the component into the *div* container (remember that the identifier is the value of *opt.target*) by means of implementing the ***constructor***. Biojs ables to use any library to manage the content of the container. *MyHelloWorld* will use jQuery[1]; the dollar symbol indicates that library is being used.

```
constructor: function (options) {
   // In JavaScript “this” always refers to the “owner” of the function
   // we're executing (http://www.quirksmode.org/js/this.html)
   // Let's preserve the reference to 'this' through the variable self. In
   //this way, we can invoke/execute
   // our component instead of the object where 'this' is being
   // invoked/executed.
  var self = this;
  // For practical use, create an object with the main DIV container
  // to be used in all of the code of our component
  this._container = jQuery("#"+self.opt.target);
  // Apply options values
  this._container.css({
    'font-family': self.opt.fontFamily, // this is an example of the use
                                        // of self instead of this
    'background-color': self.opt.backgroundColor,
    'color': self.opt.fontColor
  });

  // Add CSS class to the main container
  // Go to step 4 of this tutorial to see the CSS settings
  this._container.addClass('MyHelloWorld');

  // Disable text selection and
  // Change the selection mouse pointer
  // from text to hand.
  this._container.css({
    '-moz-user-select':'none',
    '-webkit-user-select':'none',
    'user-select':'none'
  });

  // Set the content
  text = 'Hello World!';

  for( i=0; i< text.length; i++ ) {
    this._container.append('<span>' + text[i] + '</span>');
  }

  // Internal method to initialize the event of select 'Hello'
  this._addSelectionTrigger();
  // Internal method to set the onClick event
  this._addSimpleClickTrigger();
}
```

[1] Library to manage the DOM easily across browsers. more info: http://jquery.com/

9. *MyHelloWorld* has a **dependency** with *jQuery* library. Copy the file http://code.jquery.com/jquery-1.6.4.min.js into the local dependencies repository of *Biojs* *biojs/src/main/dependencies/jquery*

10. Add the **method** *setSize* as member of *MyHelloWorld*. As stated in requirements, the method receives the size as argument. The new size is applied by changing the value of *font-size* from *css* style.

```
setSize: function(size) {
   jQuery("#"+this.opt.target).css('font-size', size);
},
```

11. Raising **events** can happening anywhere in the component code. In this case, we need raise the event *onHelloSelection* whenever the user select the word 'Hello' (more information in requirements). An invocation to the following method must be added to constructor. Note the usage of '_' as first letter in the method name indicating that it will be for internal use only.

This method registers the mouse events to figure out when the selection is happening. Invoke this method inside constructor. Whenever the user mouse clicks a span element, the event 'onClick' will be raised by the component.

```
_addSimpleClickTrigger: function () {
    var self = this;
      // Add the click event to each character in the content
      this._container.find('span')
        .click( function(e) {
                // A letter was clicked!
                // Let's discover which one was it
                // TIP: e.target contains the clicked DOM node
                var selected = jQuery(e.target).text();
                // Create an event object
                var evtObject = { "selected": selected };
                // We're ready to raise the event onClick of our component
                self.raiseEvent('onClick', evtObject);
        });
  },
```

The event "onHelloSelection" might be raised with the following code:

```
_addSelectionTrigger: function() {

   var self = this;
   var isMouseDown = false;
   // Create the CSS class called selected to change both background and color

   jQuery('<style> .selected { '+
        'background-color:' + self.opt.selectionBackgroundColor + ';' +
            'color:' + self.opt.selectionFontColor +'; }</style>'
            ).appendTo('head');
    //
    // Add the click event to each character in the content
    // But remember, we must to figure out when 'Hello' is selected only
    this._container.find('span')
     .mousedown(function() {
            // Turn on the flag
            isMouseDown = true;

            // A new selection is starting
            // Reset all by removing the CSS "selected" class if already applied
            self._container.children('span').removeClass('selected')

            // Apply the class for this span/character
            // NOTE: "this" refers to the internal object 'span'
            // NOT to the component's instance
            jQuery(this).addClass('selected');

     }).mouseover(function() {
            // Check if the mouse is being dragged
            if (isMouseDown) {
                    jQuery(this).addClass('selected');
            }
     })
     .mouseup(function() {

            /// Turn off the flag
            isMouseDown = false;

            var textSelected = '';

            // Get the entire selected word
            self._container.children('span.selected')
                    .each(function(){
                            textSelected += jQuery(this).text();
                    });

            // Since requirements, only "Hello" word should be selected
            // to raise the event
            if (textSelected == 'Hello') {
                    self.raiseEvent('onHelloSelected', {
                            textSelected : textSelected
                    })
            }
     });
  }
```

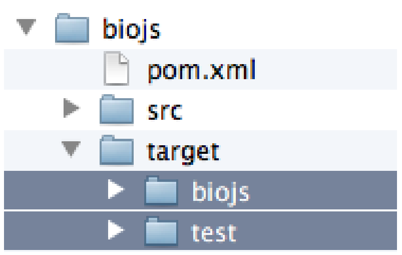
10. Create a test file. Test the component *Biojs.MyHelloWorld.js* by creating a file in your local repository (path: *biojs/src/test/javascript*) called *MyHelloWorld.html,* with this content:

```
<html>
<head>
    <title>Biojs.MyHelloWorld test</title>
    <script language="JavaScript" type="text/javascript" src="../biojs/Biojs.js"></script>
    <script language="JavaScript" type="text/javascript" src="../biojs/Biojs.MyHelloWorld.js"></script>
    <script language="JavaScript" type="text/javascript" src="../biojs/dependencies/jquery/jquery-1.6.4.min.js"></script>
    <script language="JavaScript" type="text/javascript">
      window.onload = function() {
          var instance = new Biojs.MyHelloWorld({
                  target : "YourOwnDivId",
                  selectionBackgroundColor : '#99FF00'
          });
      };
    </script>
    </head>
    <body>
      <div id="YourOwnDivId" ></div>
    </body>
</html>
```

11. Publish the test file in your Web server. Let's biojs put all of the content in the htdocs by executing the command:

```
mvn compile
```

Result:

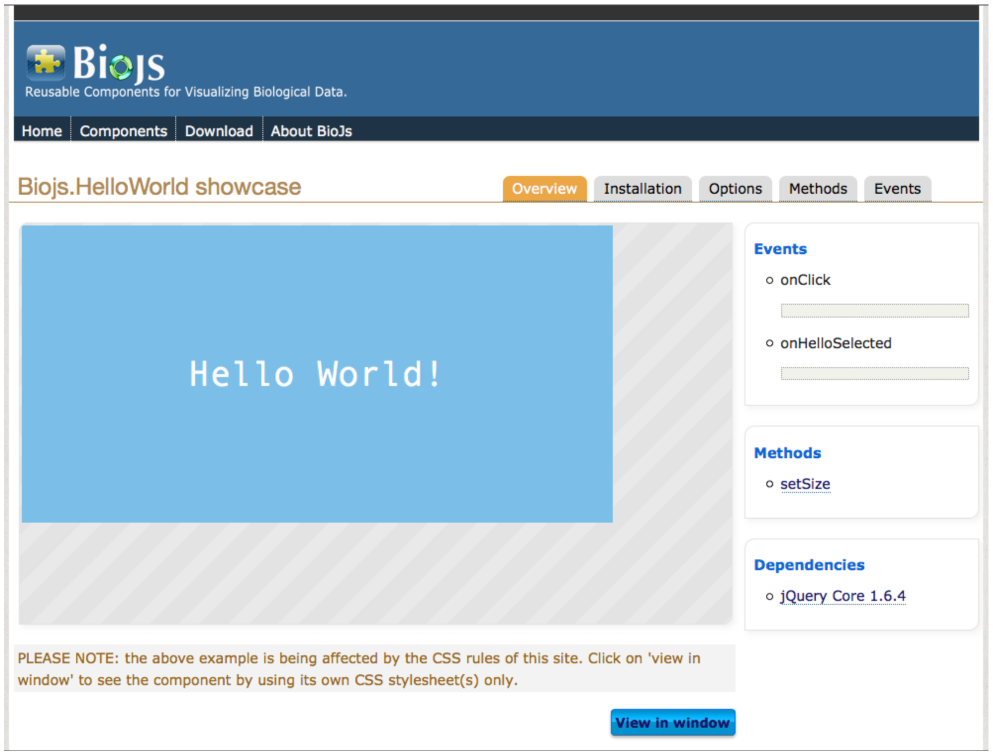


12. Open MyHelloWorld in your browser: http://localhost/biojs/test/MyHelloWorld.html

NOTE: you will need to publish the "target" directory to be able to see the test in your Web Server. Refers to the step 5 in the tutorial [Installing the Biojs repository in local](#) for more details.

13. Entire source code is available [here](#).

14. Almost done, the next task is adding the documentation by tagging the code as is shown in the Tutorial 2: Documenting the *HelloWorld* component. It enables *Biojs* to generate both *registry* and *jsdoc* automatically. It will seems like this:



Biojs.HelloWorld showcase

Hello World!

Events
- onClick
- onHelloSelected

Methods
- setSize

Dependencies
- jQuery Core 1.6.4

PLEASE NOTE: the above example is being affected by the CSS rules of this site. Click on 'view in window' to see the component by using its own CSS stylesheet(s) only.