Version: 1.1
Last revision: 13/12/2011

BioJS

A library of JavaScript components to represent biological data

John Gómez Carvajal
johncar@gmail.com

Rafael Jimenez
rafael@ebi.ac.uk

Glen van Ginkel
glen@ebi.ac.uk

Leyla Jael García
ljgarcia@ebi.ac.uk

Table of Contents

Introduction

Representing bio-oriented data on the web is a common practice in sites related with bio-oriented services. In those sites, similar ad-hoc implementations displaying the same concept (i.e.: sequence, ontology, protein, etc.) can be found. Most of them use different user controls and presentations, which could degrade the user experience across services.

This library aims to provide a common baseline specification to create web components, making it easy to maintain, develop, reuse, extend and integrate bio-oriented web applications regardless of the server-side programming language used.

1. Overview

BioJs is a library of components that are easy to reuse, maintain and deploy on the web. The library is developed using well-established methodologies as well as object-oriented design with inheritance that facilitates rapid development, reuse, extension, integration and deployment in the web applications. The primary focus of BioJs is to facilitate the development of bio-oriented applications that represent biological data in a consistent and user friendly way. However, since the library includes its own framework, it could also be used in domains outside biology.

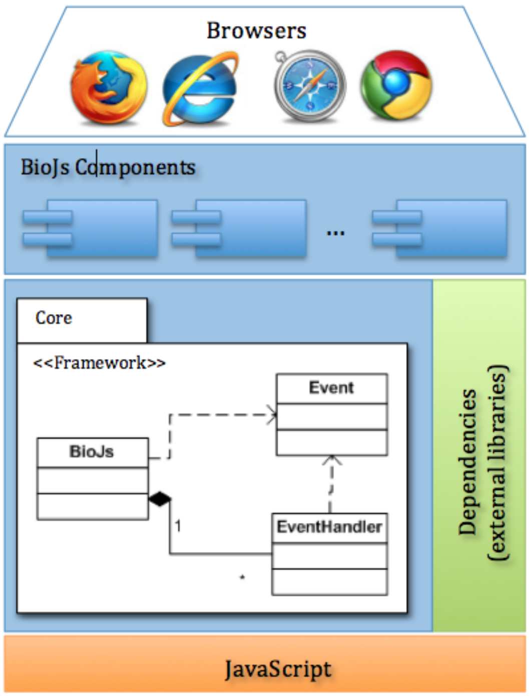


Figure 1. Architecture of BioJs

Section 2 describes the basic concepts underlying the framework so that the sections that follow are better understood. Sections 3 and 4 discuss the library extension and the reuse of components respectively. Sections 5 and 6 are useful for reference, documentation and examples.

2. Description of the Library

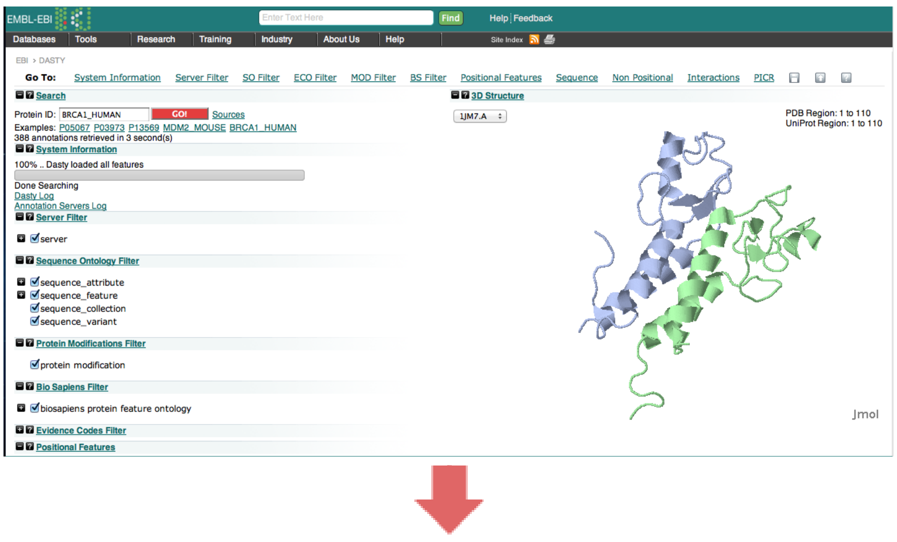
BioJs provides its own framework developed in a object-oriented way with inheritance. The library contains a base class called *Biojs* from which all *components* in the BioJs library inherit including all methods and attributes. This ensures that all components use a common mechanism for event handling, initialization and namespacing.

2.1 Component

A *component* is the basic element in the library. In different contexts, the word component can mean different things. For being more precise, it refers to ***web component*** to distinguish from the more technical and deeply means in the software engineering. In this sense, a component is a chunk or building block for a web page.

For example, consider the search engine for proteins Dasty[1] (figure 1). The page can be divided into six possible components, each one serving a different purpose such as a search box, a header, a filter control and even more complexes such as the 3D structure.

Components are essential to reuse designs in different places, and maintain a consistent user experience. High consistency improves productivity and reduces subsequent implementation time. However, it needs some considerations and agreements in order to build a baseline for both development and deployment.



388 annotations retrieved in 3 second(s)

100% .. Dasty loaded all features

Done Searching

Dasty Log

Annotation Servers Log

[1] http://www.ebi.ac.uk/dasty

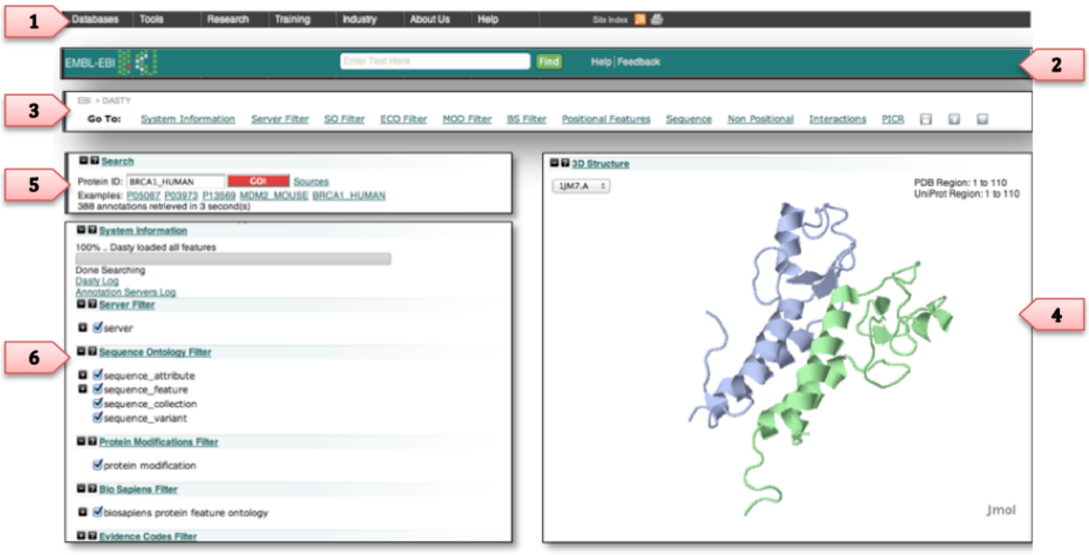
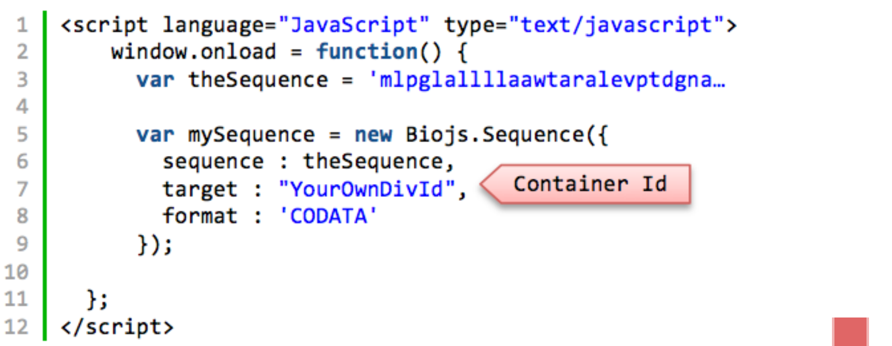


Figure 1. The Dasty web page using the keyword BRCA1_HUMAN. Top: displayed as an entire page, Above: page divided into six possible web components (Dasty is a protein DAS client copyrighted by the European Bioinformatics Institute).

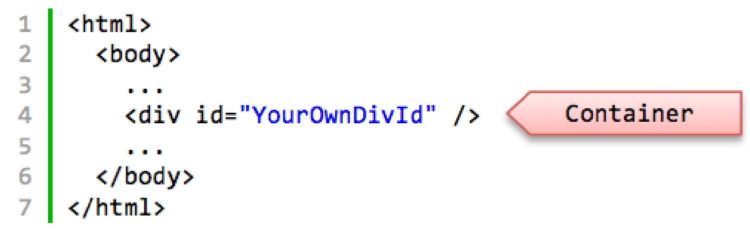
2.1.1 Container

In our context, a component is a chunk of a page which contains simpler elements inside. In this sense, it is necessary to know where and how to group the elements. On the one hand, a natural way to do that is by using the HTML *div* tag, because it defines a division or a section in an HTML document. On the second hand, the DOM[2] allow scripts to dynamically access and update the content, structure and style of elements in HTML documents.



```
<script language="JavaScript" type="text/javascript">
    window.onload = function() {
      var theSequence = 'mlpglallllaawtaralevptdgna…

      var mySequence = new Biojs.Sequence({
        sequence : theSequence,
        target : "YourOwnDivId",
        format : 'CODATA'
      });

    };
</script>
```

Component will be drawn here



```
<html>
  <body>
    ...
    <div id="YourOwnDivId" />
    ...
  </body>
</html>
```

The first consideration is to identify the target *div* to place the component's pieces. For this, the *id* attribute of the *div* element is an expected input value. For example, the above code shows

[2] Document Object Model. More info: http://www.w3.org/DOM/

an incomplete code of the HTML document where a sequence component is being instantiated. There are two input attributes the *sequence* and the *target*, and their corresponding values *'mlpgvptdgnagllaepqiamfcg...'* and *'YourOwinDiv'*. The first one is the string value of the sequence and the second one is the identifier of the *div* element into the HTML document.

2.1.2 Options

The next thing to consider is the style, the parameters, the inputs and any other relevant information to initialize the component. The mechanism to pass parameters to any component is the same: *options*. That is, a set of pairs with both *name* and *value* where each one corresponds to one option in the component. The syntax for the options is:

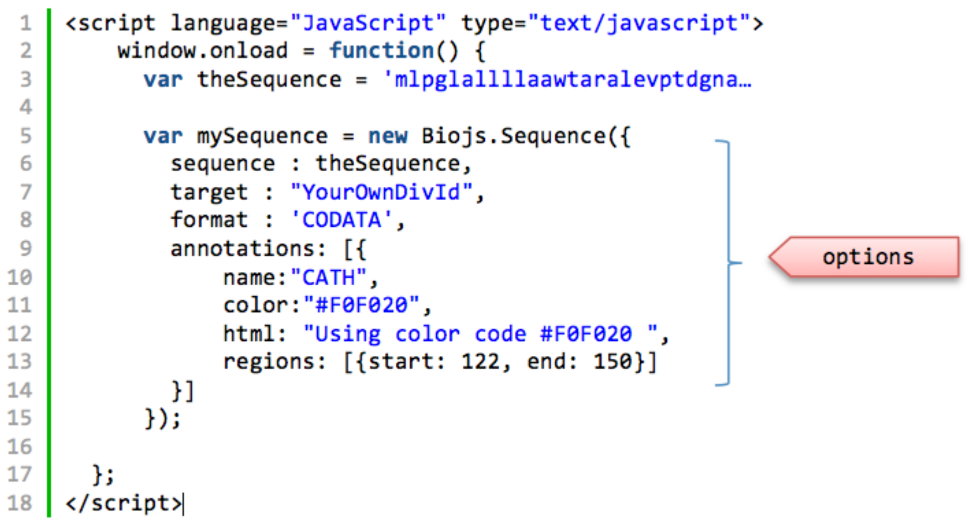
```
var instance = new Biojs.<ComponentName>({
   <optionName1>: <optionValue1>,
   <optionName2>: <optionValue2>,
   ...
   <optionNameN>: <optionValueN>
});
```

Where, *ComponentName* is the name of the component to be instantiated, pairs *optionName* and *optionValue* corresponds to any option from the supported option list by the component.

An option can be either mandatory or not depending on the existence of a default value. If the option has a default value then is optional, otherwise is mandatory. The following table shows an example of the option list for the *Biojs.Sequence* component

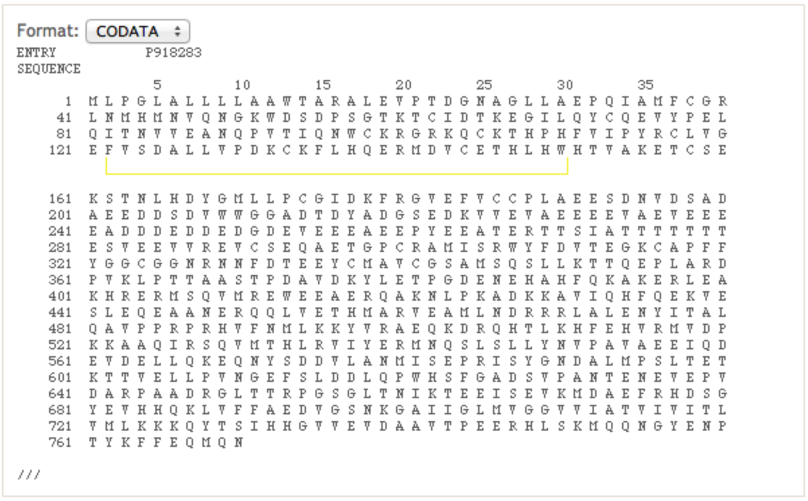
Option Name	Type	Mandatory	Description
target	string	Yes	Identifier of the *div* element where the component will be displayed.
sequence	string	Yes	The sequence.
format	string	No	Display format. One of: "RAW", "FASTA", "PRIDE", "CODATA".
hideFormatSelector	boolean	No	'true' value hides the format's combo box.
selectionColor	string	No	HTML color code for coloring a selected region.

The following source code shows how to initialize a *Biojs.Sequence* component. Note that, a plain Javascript object with four members has been passed as argument. The members sequence, target and format are options supported by the component such as shown in the former table.



```
<script language="JavaScript" type="text/javascript">
    window.onload = function() {
      var theSequence = 'mlpglallllaawtaralevptdgna…

      var mySequence = new Biojs.Sequence({
        sequence : theSequence,
        target : "YourOwnDivId",
        format : 'CODATA',
        annotations: [{
            name:"CATH",
            color:"#F0F020",
            html: "Using color code #F0F020 ",
            regions: [{start: 122, end: 150}]
        }]
      });

    };
</script>
```

result:



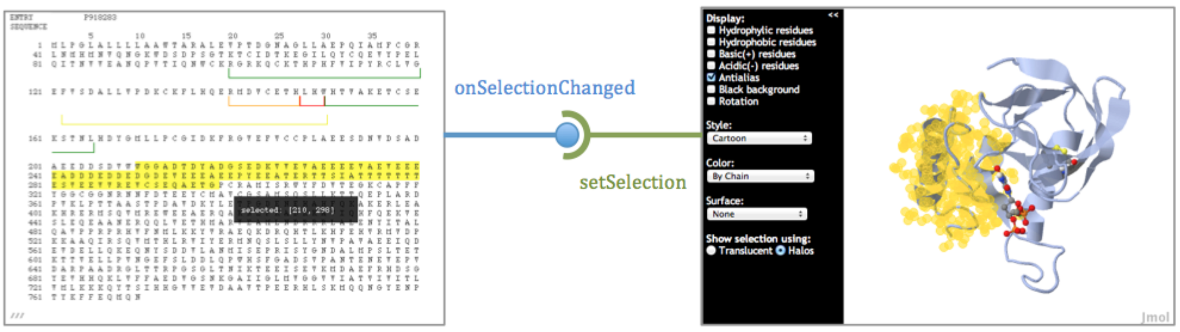
Format: CODATA

```
ENTRY           P918283
SEQUENCE
                  5         10        15        20        25        30        35
     1  M L P G L A L L L L A A W T A R A L E V P T D G N A G L L A E P Q I A M F C G R
    41  L N M H M N V Q N G K W D S D P S G T K T C I D T K E G I L Q Y C Q E V Y P E L
    81  Q I T N V V E A N Q P V T I Q N W C K R G R K Q C K T H P H F V I P Y R C L V G
   121  E F V S D A L L V P D K C K F L H Q E R M D V C E T H L H W H T V A K E T C S E

   161  K S T N L H D Y G M L L P C G I D K F R G V E F V C C P L A E E S D N V D S A D
   201  A E E D D S D V W W G G A D T D Y A D G S E D K V V E V A E E E E V A E V E E E
   241  E A D D D E D D E D G D E V E E E A E E P Y E E A T E R T T S I A T T T T T T T
   281  E S V E E V V R E V C S E Q A E T G P C R A M I S R W Y F D V T E G K C A P F F
   321  Y G G C G G N R N N F D T E E Y C M A V C G S A M S Q S L L K T T Q E P L A R D
   361  P V K L P T T A A S T P D A V D K Y L E T P G D E N E H A H F Q K A K E R L E A
   401  K H R E R M S Q V M R E W E E A E R Q A K N L P K A D K K A V I Q H F Q E K V E
   441  S L E Q E A A N E R Q Q L V E T H M A R V E A M L N D R R R L A L E N Y I T A L
   481  Q A V P P R P R H V F N M L K K Y V R A E Q K D R Q H T L K H F E H V R M V D P
   521  K K A A Q I R S Q V M T H L R V I Y E R M N Q S L S L L Y N V P A V A E E I Q D
   561  E V D E L L Q K E Q N Y S D D V L A N M I S E P R I S Y G N D A L M P S L T E T
   601  K T T V E L L P V N G E F S L D D L Q P W H S F G A D S V P A N T E N E V E P V
   641  D A R P A A D R G L T T R P G S G L T N I K T E E I S E V K M D A E F R H D S G
   681  Y E V H H Q K L V F F A E D V G S N K G A I I G L M V G G V V I A T V I V I T L
   721  V M L K K K Q Y T S I H H G V V E V D A A V T P E E R H L S K M Q Q N G Y E N P
   761  T Y K F F E Q M Q N

///
```

¿how to change this data dynamically? even more ¿how to trigger an action into the component? the next section answers these questions introducing the *methods*.

2.1.3 Methods

At the first stage, the component is displayed using the options described in the previous section. However, in most cases, the component is a dynamic element into the web page. Think in an hypothetical scenario where there are two components, a `Sequence` and a `3DStructure` as shown in figure 1 and figure 3, respectively. So, whenever the user selects a 3D structure corresponding to a region in the sequence, the sequence component should highlight that region (see figure 4). In this case, the `3DStructure` component is modifying the sequence component status. In general, any component can change the internal state of any other component by means of a *method.*



Display:
- Hydrophylic residues
- Hydrophobic residues
- Basic(+) residues
- Acidic(-) residues
- ✓ Antialias
- Black background
- Rotation

Style: Cartoon

Color: By Chain

Surface: None

Show selection using: ● Translucent ○ Halos

Figure 3. The Sequence component at left is invoking the *`setSelection`* method of the Protein3D component at right whenever raises the *`onSelectionChanged`* event.

A *method* is a public interface to interact with a component. A method receive a set of parameters as input data depending of its implementation. In the above example, *`highlight`* is a method that receives both the initial and final positions in order to indicate the region to be highlighted in red color. Usually, each component of the library have a set of those methods. Detailed information can be found in the component documentation, there can be figured out which methods it has implemented.

These methods define the behavior that is triggered by an external component. Next section describes those triggered by the component itself.

2.1.4 Events

The *events* are used to register behaviors that corresponds to user interactions and to further manipulate those registered behaviors. List of events supported are specific for each component, refer to documentation for more detailed information. Here is given a basic concept of event handling by the library.

Each component has the capability to announce to any registered *listener* what is happening inside, by sending a *message*. The message is a JavaScript object which contains the information of the event raised and the listener is a function that implements an action. Every component in BioJS can register listeners by using either *`addListener`* method or *`<eventName>`* method.

- **Method addListener**

In this case, the listener is a function that receives the event object as argument and executes an actions in response to the event. The figure 4 shows that N events are supported for Biojs component and and a function is being registered by means of method *`addListener(event, function)`*. Where, the first parameter is a string containing the event name and the second one is a *function* that will execute the action.

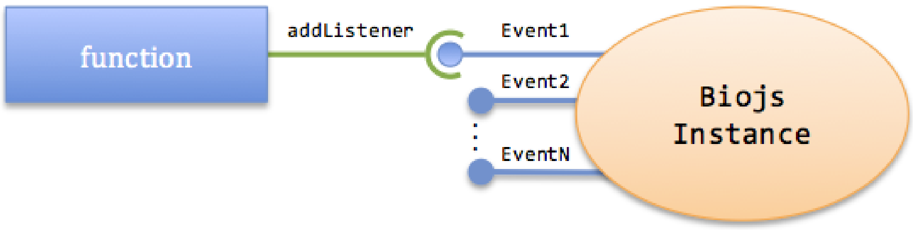


Figure 4. Left function is registered as listener of the event 1 from the component. For its part, the component can raise N types of events to registered functions.

For example, consider a sequence component instance called `mySequence` that register a function under the event *'onSelectionChanged'*. It means, whenever `mySequence` do triggering that event, an alert window is shown with the data of the event.

```
mySequence.addListener( 'onSelectionChanged',
    function( objEvent ) {
        alert("Selected: " + objEvent.start + ", " + objEvent.end );
    }
);
```

- **Method <*eventName*>**

In order to simplify the registering of an action, the library provides an alternative method for each event. The name of method is the same of event name. Figure 5 shows a component *A* connected to another component *B* by means of *B.Event1(A.method(evObj))*. Where, *Event1* is the name of event raised by *B*, *method* is a member of A receiving the event object *evObj* as argument. Note that the argument of the method and the object e must match. In other case, use a function instead of a method.

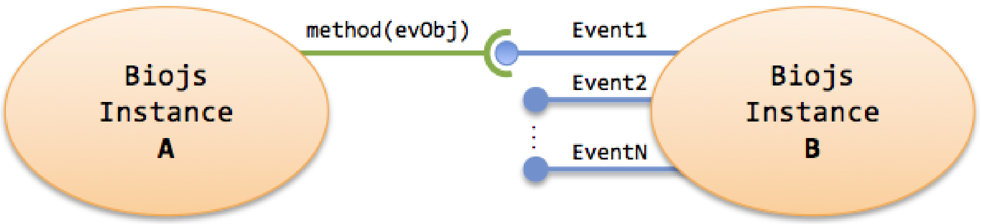


Figure 5. Left component *A* is listening the *Event1* from *B* by one of its methods. For its part, *B* can raise N types of events to who is connected to it.

For the following example, imagine the scenario of the figure 3. The component *mySequence* will invoke *setSelection* of *myProtein3D* whenever the event *onSelectionChanged* is raised. In this case, the event object matches with the arguments of *setSelection*.

```
mySequence.onSelectionChanged( myProtein3D.setSelection );
```

The other way to do the same when the object does not match with arguments, is by using a function as shown in the example:

```
mySequence.onSelectionChanged(
   function( eventObj ) {
      myProtein3D.setSelection({
           start: eventObj.start, end: eventObj.end
      });
   }
);
```

The next sections will describe the library on depth, as well as how to extend and use the components.

3. Library Structure

The library is composed of a set of core classes in JavaScript language and a set of components which extends the main class *Biojs.* Core classes *Biojs, Biojs.Event* and *Biojs.EnventHandler* are the basic building blocks for the namespacing, event handling and inheritance.

3.1 Namespace

As a general rule, since all components inherit from the *BioJs* class, they exist within the Biojs namespace, so *Biojs* is the namespace of the library. In this way, each component can be referenced using the *Biojs.<ComponentName>* syntax, where *ComponentName* is the name of the component.

The general syntax to create a new component extending the main class is:

```
var Biojs.<ComponentName> = Biojs.extend(  {  /* members */  }  );
    ^^^^^  ^^^^^^^^^^^^^^   ^^^^^ ^^^^^^    ^^^^^^^^^^^^^^^^^
      |           |           |     |           |_ class body
      |           |           |     |_ inheritance mechanism
      |           |           |_ the main class
      |           |_ component name
      |_ namespace
```

Each component resides in a separated Javascript file. Also, the library provides documentation and examples of usage. For detailed information about how to extend the library, refer to section 5.

3.2 Directories and files

There are three main directories *biojs*, *jsdoc* and *registry* containing the components code, the documentation and the registry, respectively.

The *biojs* directory contains a set of javascript files, dependencies and resources. There are a base file Biojs.js with core functionality and files of components. A convention is used to naming the files in order to correspond with the name of the component. The following example shows the files `Biojs.Sequence.js` and `Biojs.Protein3D.js` which refers to the components *Sequence* and *Protein3D*, respectively.

```
biojs
  |---Biojs.Sequence.js
  |---Biojs.Protein3D.js
...
  |---[+] dependencies
  |-----[+] jmol-12.0.48
  |-----[+] jquery
  |-----proxy.php
 ...
  |---[+] resources
  |-----[+] images
  |-----[+] css
```

The dependencies are required libraries by the components and resources are additional files (images, css files, and more).

The *`jsdoc`* directory contains the documentation in *`javadoc`* fashion. The components are self-documenting and self-testing. In one hand, the documentation will be generated automatically by using jsDoc[3] maven plugin from documented javascript code. In the other hand, the functional examples will be generated in the same way to build the *`registry`*. Both directories are shown in the following example.

```
jsdoc
  |---symbols
  |-----Biojs.Sequence.html
  |-----Biojs.Protein3D.html
 …
  |---[+] data
  |---[+] scripts
  |---[+] src
  |---[+] style

registry
  |-----Biojs.Sequence.html
  |-----Biojs.Sequence.events.html
  |-----Biojs.Sequence.options.html
  |-----Biojs.Sequence.methods.html
  |-----Biojs.Protein3D.html
  |-----Biojs.Protein3D.events.html
 ...
  |---[+] data
  |---[+] scripts
  |---[+] src
  |---[+] style
```

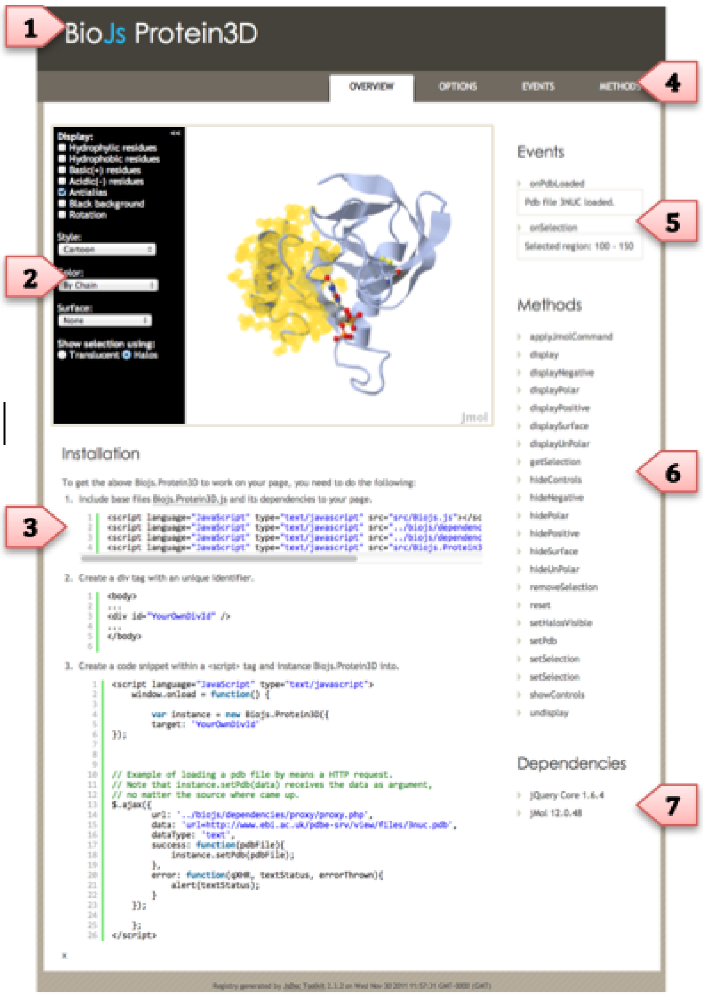
[3] JsDoc Toolkit is an application, written in JavaScript, for automatically generating template-formatted, multi-page HTML. (http://code.google.com/p/jsdoc-toolkit/).

3.3 Dependencies

Dependencies are external libraries required to execute the component. Those libraries must be included in the HTML document and depends on the component itself. The information related to dependencies must be in the component documentation. Core objects do not use external libraries, and so, dependencies is not required.

4 Documentation

The documentation provided by Biojs consists of both jsdoc and registry. By using the jsDoc application, documentation is automatically generated from commented JavaScript source code by means of templates-formatted HTML. Biojs already have the templates required by jsDoc in order to generate both `jsdoc` and `registry`. This section will describe the subset of jsDoc comment tags[4] used by Biojs to generate it.



BioJs Protein3D

Display:
- Hydrophylic residues
- Hydrophobic residues
- Basic(+) residues
- Acidic(-) residues
- Antialias
- Black background
- Rotation

Style: Cartoon

Color: By Chain

Surface: None

Show selection using: Translucent Halos

Installation

To get the above Biojs.Protein3D to work on your page, you need to do the following:

1. Include base files Biojs.Protein3D.js and its dependencies to your page.

```
<script language="JavaScript" type="text/javascript" src="src/Biojs.js"></sc
<script language="JavaScript" type="text/javascript" src="../biojs/dependenc
<script language="JavaScript" type="text/javascript" src="../biojs/dependenc
<script language="JavaScript" type="text/javascript" src="src/Biojs.Protein3
```

2. Create a div tag with an unique identifier.

```
<body>
...
<div id="YourOwnDivId" />
...
</body>
```

3. Create a code snippet within a <script> tag and instance Biojs.Protein3D into.

```
<script language="JavaScript" type="text/javascript">
    window.onload = function() {

        var instance = new Biojs.Protein3D({
        target: 'YourOwnDivId'
});



// Example of loading a pdb file by means a HTTP request.
// Note that instance.setPdb(data) receives the data as argument,
// no matter the source where came up.
$.ajax({
        url: '../biojs/dependencies/proxy/proxy.php',
        data: 'url=http://www.ebi.ac.uk/pdbe-srv/view/files/3nuc.pdb',
        dataType: 'text',
        success: function(pdbFile){
            instance.setPdb(pdbFile);
        },
        error: function(qXHR, textStatus, errorThrown){
            alert(textStatus);
        }
    });

    };
</script>
```

Events
- onPdbLoaded

Pdb file 3NUC loaded.

- onSelection

Selected region: 100 - 150

Methods
- applyJmolCommand
- display
- displayNegative
- displayPolar
- displayPositive
- displaySurface
- displayUnPolar
- getSelection
- hideControls
- hideNegative
- hidePolar
- hidePositive
- hideSurface
- hideUnPolar
- removeSelection
- reset
- setHalosVisible
- setPdb
- setSelection
- setSelection
- showControls
- undisplay

Dependencies
- jQuery Core 1.6.4
- jMol 12.0.48

Registry generated by JsDoc Toolkit 2.1.2 on Wed Nov 30 2011 11:57:21 GMT-0000 (GMT)

Figure 6. Example of **registry** of the Protein3D generated with the ad-hoc template *biojs*.

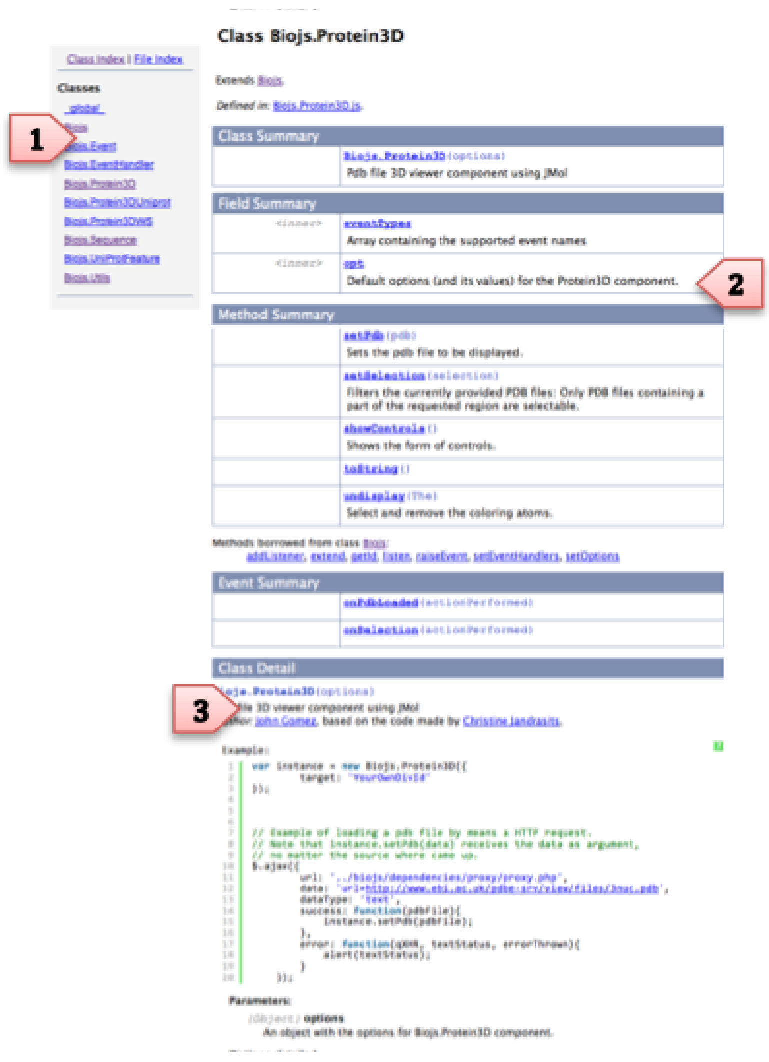
Title
Functional sample
Installation steps
Menu
Events
Methods
Links to libraries

The figure 6 shows an example of registry for the Protein3D component that was generated with the template `biojs` provided by means of Biojs's repository. Basic idea is to give a functional example of component as well as to obtain a summary. Biojs template generates 4 pages Overview, Options, Events and methods.

A more detailed view of the components is provided by the `jsdoc` template in javadoc fashion. This template generates a main page containing the list of classes and the detailed view of class like shown in the figure 7.

[4] Refers to http://code.google.com/p/jsdoc-toolkit/ for more detailed information.

How was stated before, generating the documentation requires some special tag comments in line with the javascript code. A tag is a mark starting with *'@'* character followed by its name. Biojs uses a subset of tags available from jsDoc in order to documenting classes, methods, events and options as well as the examples for the *registry*.



Class Biojs.Protein3D

Class Index | File Index

Classes
global
ojs
ojs.Event
Biojs.EventHandler
Biojs.Protein3D
Biojs.Protein3DUniprot
Biojs.Protein3DWS
Biojs.Sequence
Biojs.UniProtFeature
Biojs.Utils

Extends Biojs.

Defined in: Biojs.Protein3D.js.

Class Summary

	Biojs.Protein3D (options) Pdb file 3D viewer component using JMol

Field Summary

<inner>	**eventTypes** Array containing the supported event names
<inner>	**opt** Default options (and its values) for the Protein3D component.

Method Summary

	setPdb (pdb) Sets the pdb file to be displayed.
	setSelection (selection) Filters the currently provided PDB files: Only PDB files containing a part of the requested region are selectable.
	showControls () Shows the form of controls.
	toString ()
	undisplay (The) Select and remove the coloring atoms.

Methods borrowed from class Biojs:
addListener, extend, getId, listen, raiseEvent, setEventHandlers, setOptions

Event Summary

	onPdbLoaded (actionPerformed)
	onSelection (actionPerformed)

Class Detail

ojs.Protein3D (options)
ile 3D viewer component using JMol
hor: John Gomez, based on the code made by Christine Jandrasits.

Example:

```
var instance = new Biojs.Protein3D({
        target: 'YourOwnDivId'
});



// Example of loading a pdb file by means a HTTP request.
// Note that instance.setPdb(data) receives the data as argument,
// no matter the source where came up.
$.ajax({
        url: '../biojs/dependencies/proxy/proxy.php',
        data: 'url=http://www.ebi.ac.uk/pdbe-srv/view/files/3nuc.pdb',
        dataType: 'text',
        success: function(pdbFile){
            instance.setPdb(pdbFile);
        },
        error: function(qXHR, textStatus, errorThrown){
            alert(textStatus);
        }
    });
```

Parameters:
{Object} **options**
An object with the options for Biojs.Protein3D component.

Figure 7. Example of ***jsdoc*** of the *Protein3D* generated with the default template *jsdoc*.

1:
2:
3:

Classes index
Summary of class members
Detail of class members

4.1 Documenting the Component

A component in Biojs refers to a class which extends either main class *Biojs* or one of its children. The js file of a component extending the main class must contain a header like this:

```
/**
 * Description of the component
 *
 * @class
 * @extends Biojs
 *
 * @author <a href="mailto:johncar@gmail.com">John Gomez</a>
 */
```

In case of a component extending one of *Biojs*'s children, must write the name of its parent instead of the grand parent for *@extends* tag. In addition, a component could have a set dependencies and option as well as one example of instantiation at least.

In addition, another tag *@lends* must be included next to the class declaration to tell jsdoc the members of the object are the class members. In this case, the members *member1* (a method) and *member2* (an array) are members of the class *Biojs.Protein3D* because the *@lends* tag:

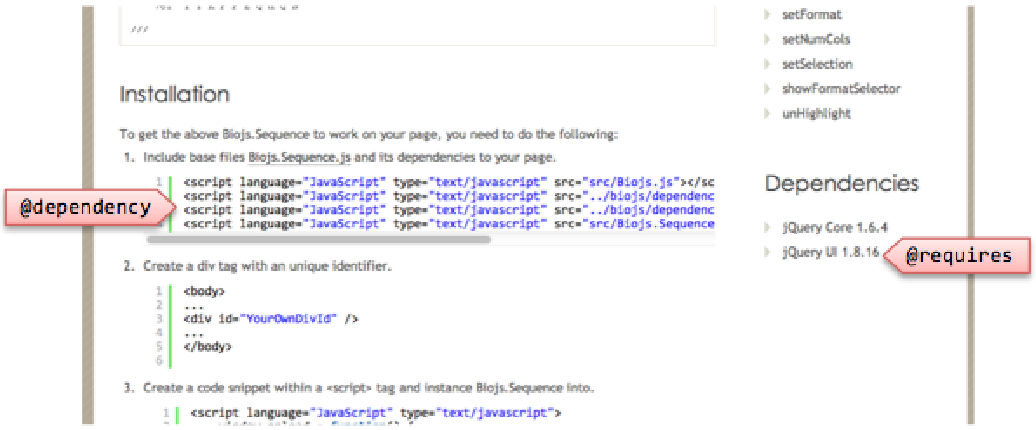
```
Biojs.Protein3D = Biojs.extend(
/** @lends Biojs.Protein3D# */
{
   member1: function (options) { … },
   member2: []
}
```

4.2 Documenting the Dependencies

There are two tags for documenting dependencies *@requires* and *@dependency*. The first one is used to put a link to the dependency on both registry and jsdoc. The second one, for showing out how to install on the HTML by means of registry. So, the following example shows a link to download jQuery[5] and the corresponding HTML header tag.

```
 * @requires <a href='http://jqueryui.com/download'>jQuery UI 1.8.16</a>
 * @dependency <script language="JavaScript" type="text/javascript" ...
```

In this way, will be generate a link in both registry and jsdoc as well as the tag in the HTML example's header:



Installation

To get the above Biojs.Sequence to work on your page, you need to do the following:

1. Include base files Biojs.Sequence.js and its dependencies to your page.

```
<script language="JavaScript" type="text/javascript" src="src/Biojs.js"></sc
<script language="JavaScript" type="text/javascript" src="../biojs/dependenc
<script language="JavaScript" type="text/javascript" src="../biojs/dependenc
<script language="JavaScript" type="text/javascript" src="src/Biojs.Sequence
```

2. Create a div tag with an unique identifier.

```
<body>
...
<div id="YourOwnDivId" />
...
</body>
```

3. Create a code snippet within a <script> tag and instance Biojs.Sequence into.

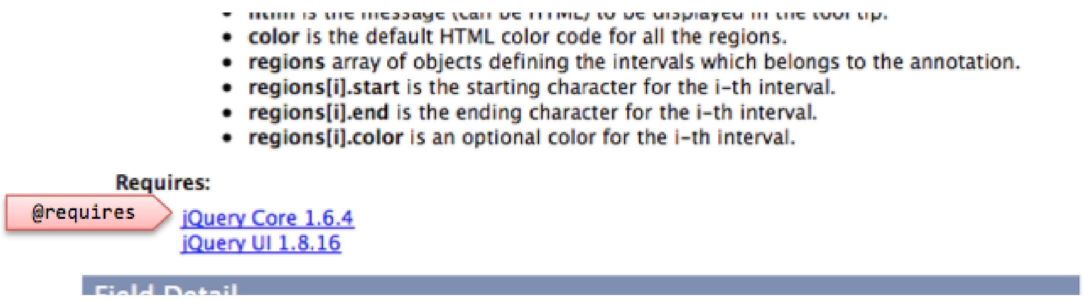
```
<script language="JavaScript" type="text/javascript">
```

Dependencies

- jQuery Core 1.6.4
- jQuery UI 1.8.16

Figure 8.a) Partial view of a registry showing tags result.

[5] A library to manipulate the DOM easily.



- color is the default HTML color code for all the regions.
- regions array of objects defining the intervals which belongs to the annotation.
- regions[i].start is the starting character for the i-th interval.
- regions[i].end is the ending character for the i-th interval.
- regions[i].color is an optional color for the i-th interval.

Requires:

@requires

jQuery Core 1.6.4
jQuery UI 1.8.16

Figure 8.b) Partial view of class detail of a jsdoc.

4.3 Documenting the Options

Options are set of configurable parameters supported by a component. Set of available options could be different for one component to other. The general syntax to document is :

```
* @option {type} name description
```

where, type is the expected data type, name and description are text describing the option. Basic types are *int, doube, string, bool* or *Object*. For arrays use *'[]'* notation, for example *Object[]* would refers to an array of objects. By default, an option is mandatory, in opposite case the name must me enclosed by *'[]'* and provide a default value following the syntax *[name=<defaultValue>]*. Try to describe the option as detailed as possible using HTML tags. Explain each member in case of an object. This example shows the declaration of mandatory option *target* and another optional one called *format* with a value by default *"FASTA":*

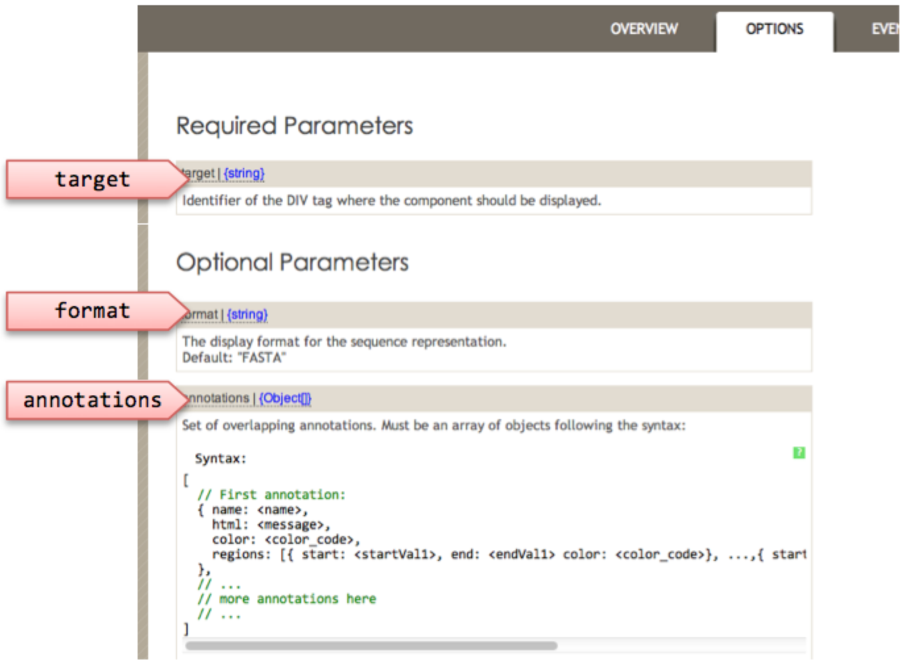
```
* @option {string} target
*    Identifier of the DIV tag where the component should be displayed.
*
* @option {string} [format="FASTA"]
*    The display format for the sequence representation.
```

Here is an example of an option array of objects named *annotations* and optional:

```
* @option {Object[]} [annotations]
*    Set of overlapping annotations. Must be an array of objects following the
*    syntax:
*     <pre class="brush: js" title="Syntax:">
*            [
*              // First annotation:
*              { name: &lt;name&gt;,
*                html: &lt;message&gt;,
*                color: &lt;color_code&gt;,
*              },
*              // ...
*              // more annotations here
*              // ...
*            ]
*             </pre>
*    where:
*      <ul>
```

```
 *        <li><b>name</b> is the unique name for the annotation</li>
 *        <li><b>html</b> is the message to be displayed in the tool tip.</li>
 *        <li><b>color</b> is the default HTML color code for all the regions.</li>
 *      </ul>
```

From the above example is obtained:



Required Parameters

arget | {string}

Identifier of the DIV tag where the component should be displayed.

Optional Parameters

rmat | {string}

The display format for the sequence representation.
Default: "FASTA"

nnotations | {Object[]}

Set of overlapping annotations. Must be an array of objects following the syntax:

```
 Syntax:
[
  // First annotation:
  { name: <name>,
    html: <message>,
    color: <color_code>,
    regions: [{ start: <startVal1>, end: <endVal1> color: <color_code>}, ...,{ start
  },
  // ...
  // more annotations here
  // ...
]
```

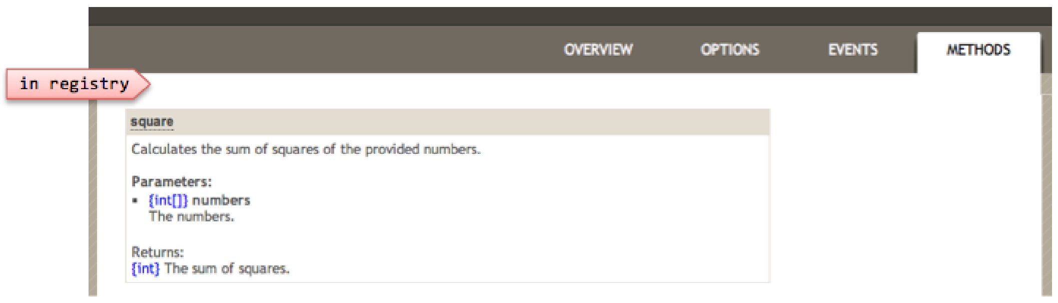
4.4 Documenting the Mehods

Every function that being declared inside the class body is considered as a method member of it. Basically, methods has a description, zero or more input parameters and a return value. The general syntax is:

```
/**
* Description of the method.
* @param {type} name1 Description of the argument name1.
* @param {type} name2 Description of the argument name2.
  ...
* @param {type} nameN Description of the argument nameN.
*
* @returns {object} A Clone of the object passed as argument.
*/
methodName: function( name1, name2,..., nameN ) {...}
```

Arguments can be declared using the same syntax of options. The following example describe how to document a simple method which calculates the sum of squares from a array of integers:

```
/**
* Calculates the sum of squares of the provided numbers.
* @param {int[]} numbers The numbers.
* @returns {int} The sum of squares.
*/
square: function(numbers) {
   var sum = 0;
   for ( i in numbers ) {
      sum += numbers[i]*numbers[i];
   }
   return sum;
}
```

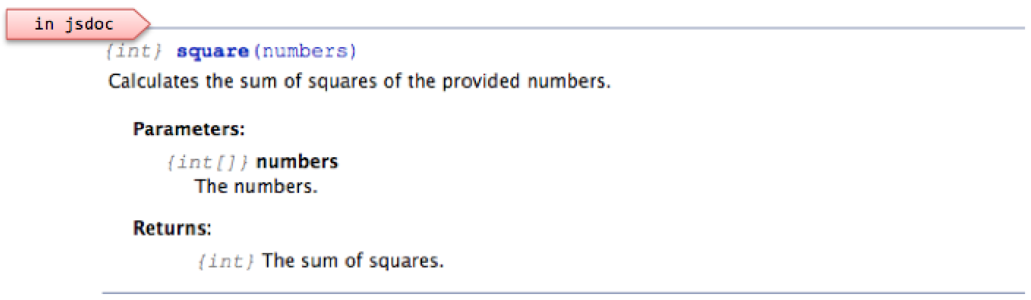


square

Calculates the sum of squares of the provided numbers.

Parameters:
- {int[]} numbers
  The numbers.

Returns:
{int} The sum of squares.



{int} **square**(numbers)

Calculates the sum of squares of the provided numbers.

Parameters:

{int[]} **numbers**
The numbers.

Returns:

{int} The sum of squares.

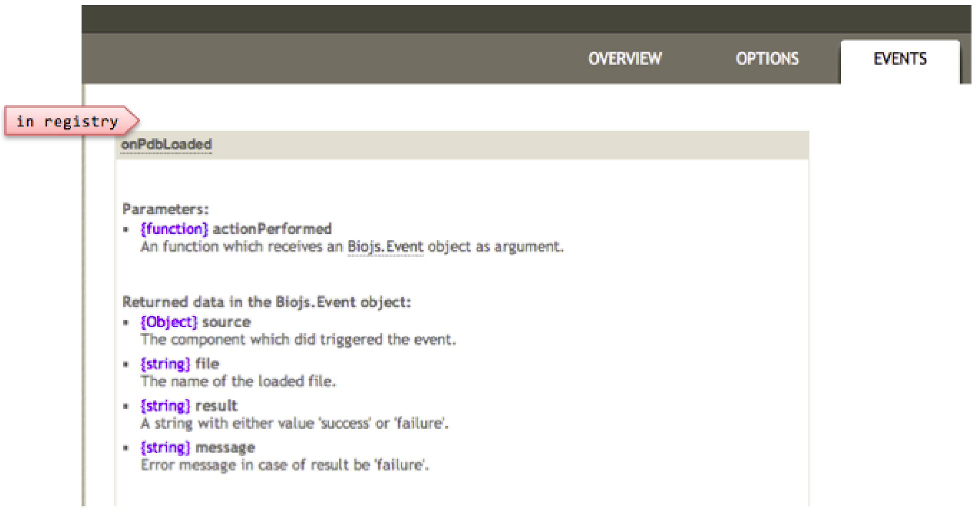
4.5 Documenting the Events

Array *eventTypes* is a special member of any component where the event names are declared. Each value on it must be documented by using the syntax:

```
eventTypes: [
   /**
   * @event
   * @name Biojs.<ComponentName>#<eventName>
   * @param {function} actionPerformed The function to execute...
   * @eventData {<type>} eventData1 The data1 received as member of event object.
   * @eventData {<type>} eventData2 The data2 received as member of event object.
     ...
   * @eventData {<type>} eventDataN The dataN received as member of event object.
   */
   "<eventName>",
   ...
]
```

Here is an example documenting the event *onPdbLoaded* member of *Biojs.Protein3D*. The event object passed to *actionPerformed* has three members *source*, *file* and *result*:

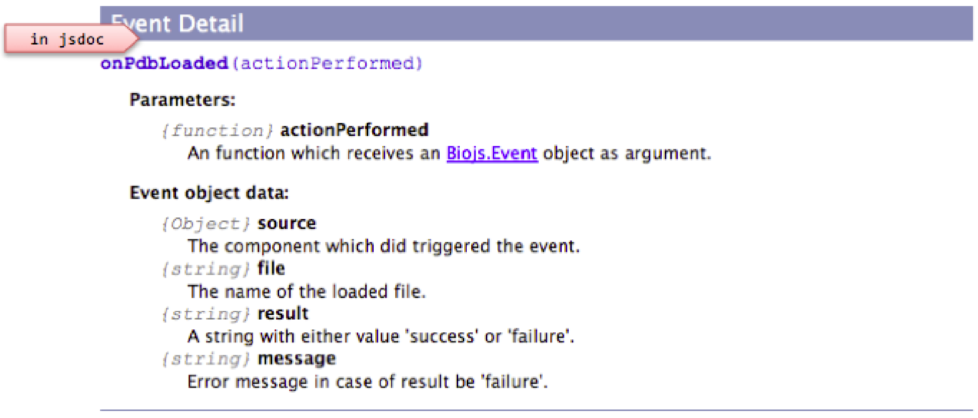
```
eventTypes: [
   /**
   * @event
   * @name Biojs.Protein3D#onPdbLoaded
   * @param {function} actionPerformed The function to execute.
   * @eventData {Object} source The component which did triggered the event.
   * @eventData {string} file The name of the loaded file.
   * @eventData {string} result A string with either value 'success' or 'failure'.
   */
   "onPdbLoaded"
]
```



onPdbLoaded

Parameters:
- {function} actionPerformed
  An function which receives an Biojs.Event object as argument.

Returned data in the Biojs.Event object:
- {Object} source
  The component which did triggered the event.
- {string} file
  The name of the loaded file.
- {string} result
  A string with either value 'success' or 'failure'.
- {string} message
  Error message in case of result be 'failure'.



Event Detail

onPdbLoaded(actionPerformed)

Parameters:

{function} **actionPerformed**
An function which receives an Biojs.Event object as argument.

Event object data:

{Object} **source**
The component which did triggered the event.
{string} **file**
The name of the loaded file.
{string} **result**
A string with either value 'success' or 'failure'.
{string} **message**
Error message in case of result be 'failure'.

4.6 Working Examples

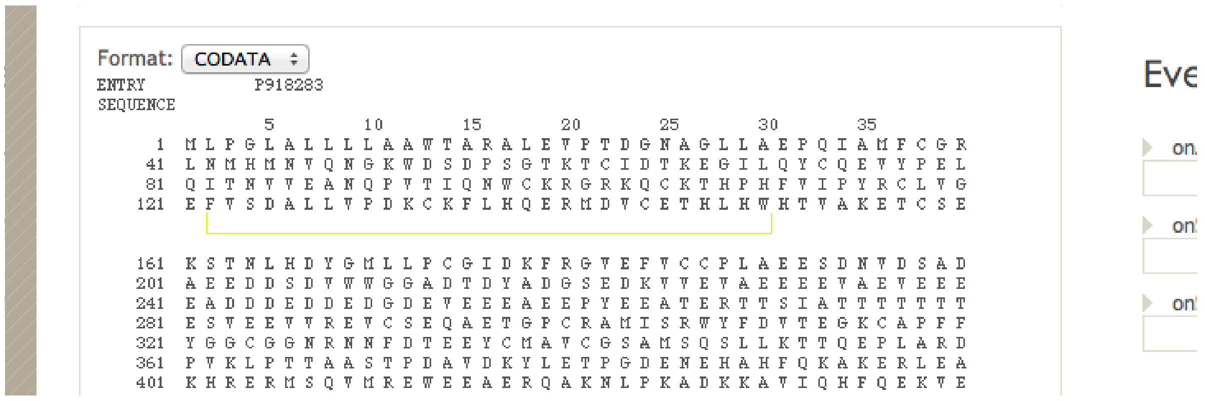
In addition to the documentation, Biojs uses a special comments to build working examples by means the tag *@example*. An example contains javascript code that will be included in both registry and jsdoc. It can be declared in methods, events and classes.

Following examples shows the instantiation of *Biojs.Sequence* under the *instance* variable a well as the usage of methods and events. Setting *'instance'* as the name of the instance is recommended for being consistent with the examples along the resulting javascript code. So, the examples can be integrated by Biojs.

Example of instantiation (must be included in the class header, see 3.3.1):

```
 * @example
 * var theSequence = 'mlpglallllaawtaralevptdgnagllaepqiamfcgrlnmhmnvqngkwdsdpsg...
 * var instance = new Biojs.Sequence({
 *            sequence : theSequence,
 *            target : "YourOwnDivId",
 *            format : 'CODATA',
 *            id : 'P918283',
 *            annotations: [{
 *                   name:"CATH",
 *                   color:"#F0F020",
 *                      html: "Using color code #F0F020 ",
 *                      regions: [{start: 122, end: 150}]
 *              }]
 * });
```

It will generate the working example:

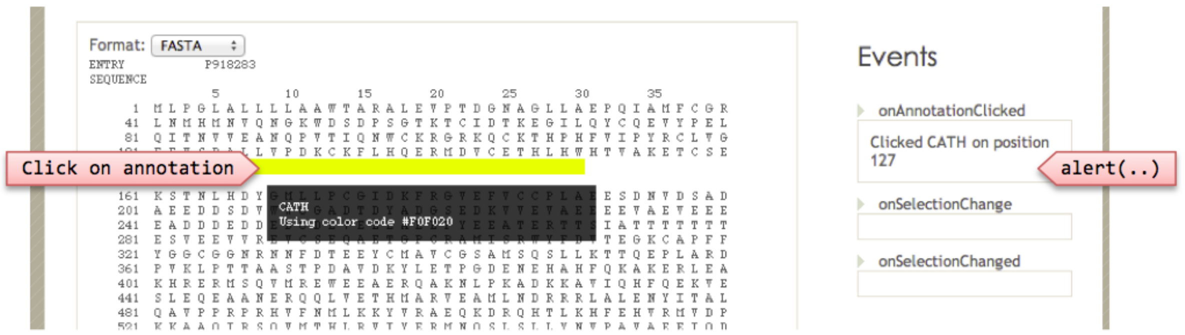


```
Format: CODATA
ENTRY          P918283
SEQUENCE
               5         10        15        20        25        30        35
    1  M L P G L A L L L L A A W T A R A L E V P T D G N A G L L A E P Q I A M F C G R
   41  L N M H M N V Q N G K W D S D P S G T K T C I D T K E G I L Q Y C Q E V Y P E L
   81  Q I T N V V E A N Q P V T I Q N W C K R G R K Q C K T H P H F V I P Y R C L V G
  121  E F V S D A L L V P D K C K F L H Q E R M D V C E T H L H W H T V A K E T C S E

  161  K S T N L H D Y G M L L P C G I D K F R G V E F V C C P L A E E S D N V D S A D
  201  A E E D D S D V W W G G A D T D Y A D G S E D K V V E V A E E E E V A E V E E E
  241  E A D D D E D D E D G D E V E E E A E E P Y E E A T E R T T S I A T T T T T T T
  281  E S V E E V V R E V C S E Q A E T G P C R A M I S R W Y F D V T E G K C A P F F
  321  Y G G C G G N R N N F D T E E Y C M A V C G S A M S Q S L L K T T Q E P L A R D
  361  P V K L P T T A A S T P D A V D K Y L E T P G D E N E H A H F Q K A K E R L E A
  401  K H R E R M S Q V M R E W E E A E R Q A K N L P K A D K K A V I Q H F Q E K V E
```

Example of event (must be included with the event tags, see 4.5):

```
* @example
* instance.onAnnotationClicked(
*    function( objEvent ) {
*       alert("Clicked " + objEvent.name + " on position " + objEvent.pos );
*    }
* );
```

The event *onAnnotationClicked* is included on the *registry*. Whenever the event is raised, the *alert* text is written on a *div*. Note that the example have an alert invocation, however, when annotation is clicked does not show the alert window. It is right because Biojs do replace the alert by a writing in a *div*.



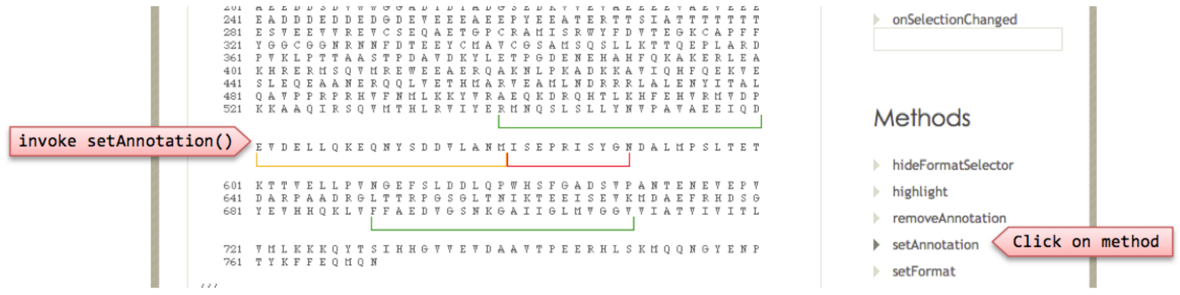
```
Format: FASTA
ENTRY          P918283
SEQUENCE
               5         10        15        20        25        30        35
    1  M L P G L A L L L L A A W T A R A L E V P T D G N A G L L A E P Q I A M F C G R
   41  L N M H M N V Q N G K W D S D P S G T K T C I D T K E G I L Q Y C Q E V Y P E L
   81  Q I T N V V E A N Q P V T I Q N W C K R G R K Q C K T H P H F V I P Y R C L V G
                     L L V P D K C K F L H Q E R M D V C E T H L H W H T V A K E T C S E

  161  K S T N L H D Y G M L L P C G I D K F R G V E F V C C P L A E E S D N V D S A D
  201  A E E D D S D V W W G G A D T D Y A D G S E D K V V E V A E E E E V A E V E E E
  241  E A D D D E D D E D G D E V E E E A E E P Y E E A T E R T T S I A T T T T T T T
  281  E S V E E V V R E V C S E Q A E T G P C R A M I S R W Y F D V T E G K C A P F F
  321  Y G G C G G N R N N F D T E E Y C M A V C G S A M S Q S L L K T T Q E P L A R D
  361  P V K L P T T A A S T P D A V D K Y L E T P G D E N E H A H F Q K A K E R L E A
  401  K H R E R M S Q V M R E W E E A E R Q A K N L P K A D K K A V I Q H F Q E K V E
  441  S L E Q E A A N E R Q Q L V E T H M A R V E A M L N D R R R L A L E N Y I T A L
  481  Q A V P P R P R H V F N M L K K Y V R A E Q K D R Q H T L K H F E H V R M V D P
  521  K K A A Q T R S Q V M T H L R V T V E R M N Q S L S L L V N V P A V A E E T Q D

CATH
Using color code #F0F020

Events
onAnnotationClicked
Clicked CATH on position 127
onSelectionChange
onSelectionChanged
```

Example of method (must be included in the method tags, see 4.4):

```
* @example
* // Annotations using regions with different colors.
* instance.setAnnotation({
```

```
*     name: "UNIPROT",
*     html: "&lt;br&gt; Example of &lt;b&gt;HTML&lt;/b&gt;"
*     color: "green",
*     regions: [
*        {start: 540, end: 560},
*        {start: 561, end:580, color: "#FFA010"},
*        {start: 581, end:590, color: "red"},
*        {start: 690, end:710}]
* });
```

The method *setAnnotation* is included in the registry and invoked whenever the user clicks on it.



```
241  E A D D D E D D E D G D E V E E E A E E P Y E E A T E R T T S I A T T T T T T T
281  E S V E E V V R E V C S E Q A E T G P C R A M I S R W Y F D V T E G K C A P F F
321  Y G G C G G N R N N F D T E E Y C M A V C G S A M S Q S L L K T T Q E P L A R D
361  P V K L P T T A A S T P D A V D K Y L E T P G D E N E H A H F Q K A K E R L E A
401  K H R E R M S Q V M R E W E E A E R Q A K N L P K A D K K A V I Q H F Q E K V E
441  S L E Q E A A N E R Q Q L V E T H M A R V E A M L N D R R R L A L E N Y I T A L
481  Q A V P P R P R H V F N M L K K Y V R A E Q K D R Q H T L K H F E H V R M V D P
521  K K A A Q I R S Q V M T H L R V I Y E R M N Q S L S L L Y N V P A V A E E I Q D

     E V D E L L Q K E Q N Y S D D V L A N M I S E P R I S Y G N D A L M P S L T E T

601  K T T V E L L P V N G E F S L D D L Q P W H S F G A D S V P A N T E N E V E P V
641  D A R P A A D R G L T T R P G S G L T N I K T E E I S E V K M D A E F R H D S G
681  Y E V H H Q K L V F F A E D V G S N K G A I I G L M V G G V V I A T V I V I T L

721  V M L K K K Q Y T S I H H G V V E V D A A V T P E E R H L S K M Q Q N G Y E N P
761  T Y K F F E Q M Q N
```

Note that the former examples of the event *setAnnotation* and the method *onAnnotationClicked* are connected by means of *instance* variable. It is very important take this in account to build a working example and the component can be tested properly.

5. Reusing a component

Biojs components can be used in many web pages. It is very easy to install by following three basic steps dependencies installation, setting the container and component instantiation.

5.1 Dependencies installation

As it was stated in section 3.2, Biojs has a local repository of dependencies to make easy the installation process. However, you are free on using any source to get the dependencies of a particular component. Only be sure about using a suitable version of dependency.

The installation consist on adding a *script* tag in the HTML header. Following example shows the dependencies for *Biojs.Protein3D*: jQuery, jMol, Biojs and the component code.

```
<html>
   <head>
     <script language="JavaScript" type="text/javascript" src="src/Biojs.js"></script>
     <script language="JavaScript" type="text/javascript" src="biojs/dependencies/jquery/jquery-1.4.2.min.js">
     <script language="JavaScript" type="text/javascript" src="biojs/dependencies/jmol-12.0.48/Jmol.js"></script>
     <script language="JavaScript" type="text/javascript" src="src/Biojs.Protein3D.js"></script>
...
```

5.2 Setting the container

The container is a HTML *div* element inside of the web page where the component will be installed. Just provide the *id* in order to be used by component. Its position in the page is not relevant but the style yes it is. So, use a style-plain container to do not conflict with the component style settings.

For being consistent with the former example, the container with *id="YourOwnDivId"* is added to the page in this way:

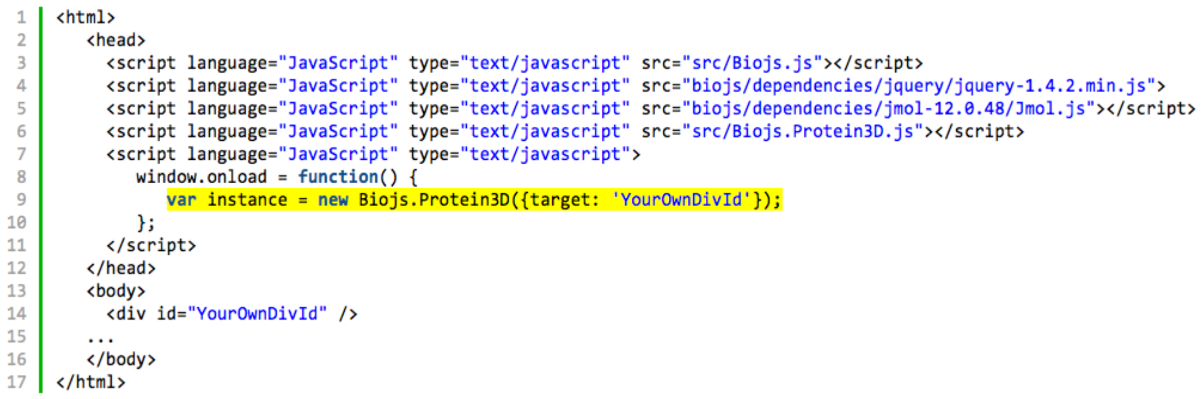
```
<html>
   <head>
     <script language="JavaScript" type="text/javascript" src="src/Biojs.js"></script>
     <script language="JavaScript" type="text/javascript" src="biojs/dependencies/jquery/jquery-1.4.2.min.js">
     <script language="JavaScript" type="text/javascript" src="biojs/dependencies/jmol-12.0.48/Jmol.js"></script>
     <script language="JavaScript" type="text/javascript" src="src/Biojs.Protein3D.js"></script>
   </head>
   <body>
     <div id="YourOwnDivId" />
   ...
   </body>
</html>
```

For more information about the container, refers to section 2.1.1.

5.3 Component instantiation

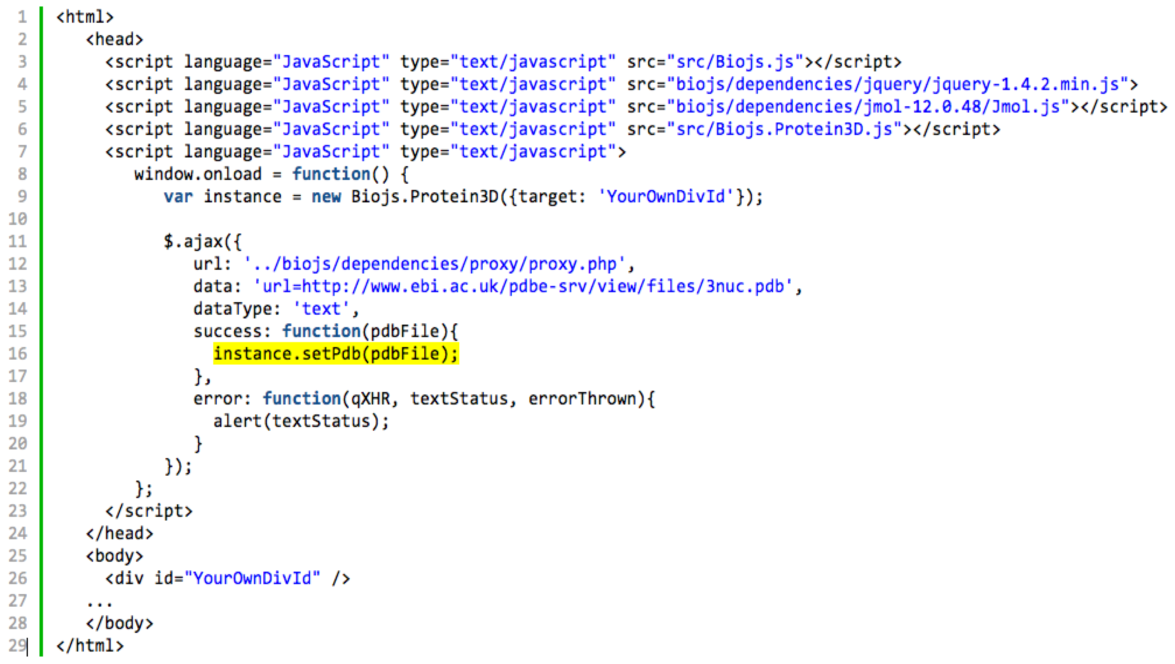
Before creating some component, review the option list supported by it. As was stated in section 2.1.2, there are some mandatory options as well as others not mandatory. Mandatory ones must be provided to ensure a properly component initialization.

Since the component needs a container on instantiation time, use the *onLoad* document event to ensure the existence of the *div* element. Completing the former example, a *Biojs.Protein3D* is being instantiated when the page is loaded:



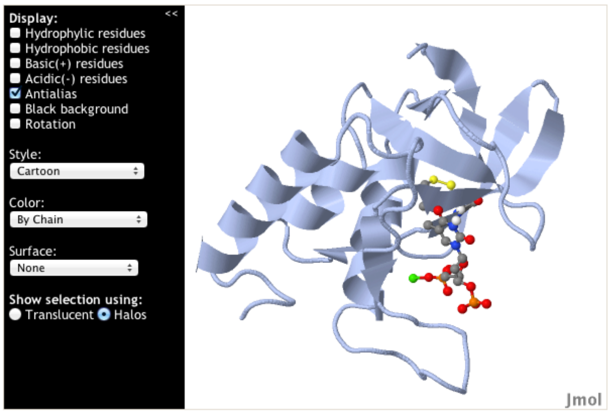
```
<html>
   <head>
     <script language="JavaScript" type="text/javascript" src="src/Biojs.js"></script>
     <script language="JavaScript" type="text/javascript" src="biojs/dependencies/jquery/jquery-1.4.2.min.js">
     <script language="JavaScript" type="text/javascript" src="biojs/dependencies/jmol-12.0.48/Jmol.js"></script>
     <script language="JavaScript" type="text/javascript" src="src/Biojs.Protein3D.js"></script>
     <script language="JavaScript" type="text/javascript">
        window.onload = function() {
           var instance = new Biojs.Protein3D({target: 'YourOwnDivId'});
        };
     </script>
   </head>
   <body>
     <div id="YourOwnDivId" />
   ...
   </body>
</html>
```

Almost complete, however this component needs some pdb file to visualize it. Lets get the pdb file of *3nuc* from *url=http://www.ebi.ac.uk/pdbe-srv/view/files/3nuc.pdb* by means of jQuery Ajax request and then applying *setPdb* method with retrieved data as argument:



```
<html>
   <head>
     <script language="JavaScript" type="text/javascript" src="src/Biojs.js"></script>
     <script language="JavaScript" type="text/javascript" src="biojs/dependencies/jquery/jquery-1.4.2.min.js">
     <script language="JavaScript" type="text/javascript" src="biojs/dependencies/jmol-12.0.48/Jmol.js"></script>
     <script language="JavaScript" type="text/javascript" src="src/Biojs.Protein3D.js"></script>
     <script language="JavaScript" type="text/javascript">
        window.onload = function() {
           var instance = new Biojs.Protein3D({target: 'YourOwnDivId'});

           $.ajax({
              url: '../biojs/dependencies/proxy/proxy.php',
              data: 'url=http://www.ebi.ac.uk/pdbe-srv/view/files/3nuc.pdb',
              dataType: 'text',
              success: function(pdbFile){
                instance.setPdb(pdbFile);
              },
              error: function(qXHR, textStatus, errorThrown){
                alert(textStatus);
              }
           });
        };
     </script>
   </head>
   <body>
     <div id="YourOwnDivId" />
   ...
   </body>
</html>
```

Note the usage of a proxy instead of the *url*. It is doing due to cross-domain security limitations on browsers. By using a proxy is possible to overcome these limitations. So, the proxy is a dependency for this component not installed in HTML header due it is a *php* script. The resulting component would be:


Display:
- ☐ Hydrophylic residues
- ☐ Hydrophobic residues
- ☐ Basic(+) residues
- ☐ Acidic(-) residues
- ☑ Antialias
- ☐ Black background
- ☐ Rotation

Style: Cartoon

Color: By Chain

Surface: None

Show selection using: Translucent [✓ Halos]

6 Extending the Library

If you plan to contribute or modify the library, don't hesitate to become a member in our googlecode project at http://code.google.com/p/biojs/. In the following sections will be described the required concepts to extending the library.

6.1 Repository

If you plan to make changes, use this command to check out the code as yourself using HTTPS:

```
# Project members authenticate over HTTPS to allow committing changes.
svn checkout https://biojs.googlecode.com/svn/trunk/ biojs --username <your_username>
```

Use this command to anonymously check out the latest project source code:

```
# Non-members may check out a read-only working copy anonymously over HTTP.
svn checkout http://biojs.googlecode.com/svn/trunk/ biojs-read-only
```

Project directory structure:

```
<project-root>/
  |
  [+] pom.xml
  |
  [+] src/
  |   |
  |   [+] main/
  |   |   |
  |   |   [+] javascript/ (source location for js files)
  |   |   [+] resources/ (source location for any static resources)
  |   |       |
  |   |       [+] dependencies/
  |   |
  |   [+] test/
  |   |    |
  |   |    [+] javascript/ (source location for (jsunit) test sources)
  |   |
  |   [+] doc/ (source location for documentation files)
  |        |
  |        [+] biojsTemplate/ (template for jsdoc)
  |
  [+] target/ (location for compiled files)
```

The *src* directory contains the all the library files spread in four sub-directories: *main, test* and *doc*. Where, *main* contains the source files of the components, *test* contains the Jsunit scripts and *doc* contains the Jsdoc templates for generating both documentation and registry.

6.2 Project Lifecycle

The Javascript Maven project provides maven plug-ins and extensions dedicated to javascript development, either as standalone pure javascript libraries or as part of web application. Its goal is to promote best development practices with a standardized project organization, so that developers can be quickly productive on a project and take advantage of the best tools available for javascript development, documentation or quality assurance.

Project Definition

The *pom.xml* is configured to include some extensions to enable the javascript support:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>uk.ac.ebi</groupId>
  <artifactId>biojs</artifactId>
  <packaging>javascript</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>BioJS</name>

  <plugins>
      <plugin>
        <groupId>nl.windgazer</groupId>
        <artifactId>jsdoctk-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
               <template>${basedir}/src/doc/biojsTemplate</template>
               <directory>${project.build.directory}</directory>
               <recurse>1</recurse>
               <ext>js</ext>
               <srcDir>${basedir}/src/main/javascript</srcDir>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>javascript-maven-plugin</artifactId>
        <extensions>true</extensions>
        <version>1.0-alpha-1-SNAPSHOT</version>
        <executions>
            <execution>
                <goals>
                    <goal>compress</goal>
                </goals>
                <phase>compile</phase>
            </execution>
        </executions>
        <configuration>
            <sourceDirectory>src/main/javascript</sourceDirectory>
            <webappDirectory>${basedir}/target</webappDirectory>
            <outputDirectory>${basedir}/target/biojs</outputDirectory>
            <compressor>jsmin</compressor>
            <scripts>biojs</scripts>
        </configuration>
      </plugin>
  </plugins>
```

```
<project>
```

Version handling

/**/

6.3 Creation of a New Component

Before coding anything in javascript, a detailed component description is required. This gathering requirements questionnaire can help you in both collect and identify relevant information about your component as well as how should be displayed. Options, methods, events, style and mock-up will help you to visualize the functionality on the paper. Even it able to do user tests with the mock-up on paper (or another media) before coding.

6.3.1 Inheritance model

Creating a new component with Biojs consist on extending out the main class `Biojs` or any existent component. Biojs supports a simple inheritance model which ables to create a new component from another by means of a static method *`extends`*.

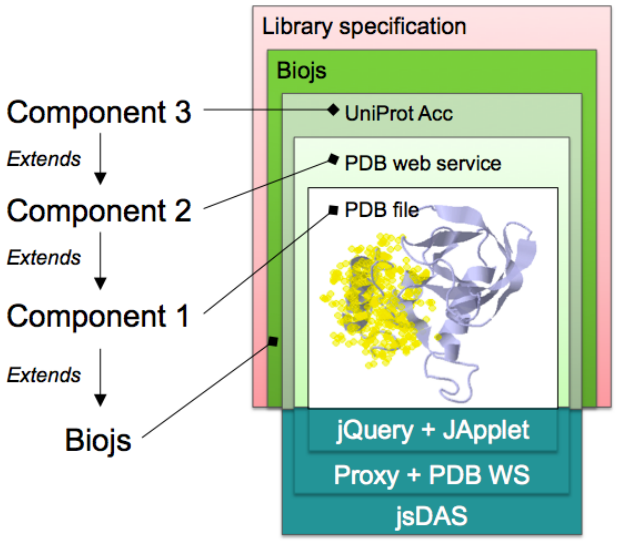


Figure 9. Example of inheritance chain.

The example in the figure 9 describes an inheritance chain of 3 components. *`Component1`* is extending from the base class *`Biojs`*, then *`Component2`* from *`Component1`* and finally *`Component3`* from *`Component2`*. Note that dependencies in the bottom not rely on the Biojs scope. It means that *`Biojs`* main class does not depends on any library, however the components do it.

6.3.1 Writing the code

The first stage in creating a new component is assign it a new file in the *`src`* directory (see section 6.1). A convention is used to name files such as *`Biojs.<ComponentName>.js`*.

Following template shows the syntax which must have the file content:

```
Biojs.<ComponentName> = Biojs.[<ParentComponentName>.]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.
 */
 });
```

Where *<ComponentName>* is the name for the component will be created and *<ParentComponentName>*. is the name of parent just in case of extending an existent component. For creating a component from scratch use *Biojs.extend* or *Biojs.<ParentComponentName>.extend* in other case.

Note that there are 3 special members *constructor*, *eventTypes* and *opt*. The first one is the method that will be used to build new instances. The second one is a string array containing the name of the events supported. The last one contains the default values for the options. These members will be managed in a different way by the inheritance.

6.3.2 Defining the Options

There is a special member in the component code called *opt*. It is an javascript plain object which contains a set of attributes and default values. The biojs inheritance model merge the options from both the child and its parents to preserve all supported options. In case of the same option be present in both parent and child, default value from child will override the parent's option value.

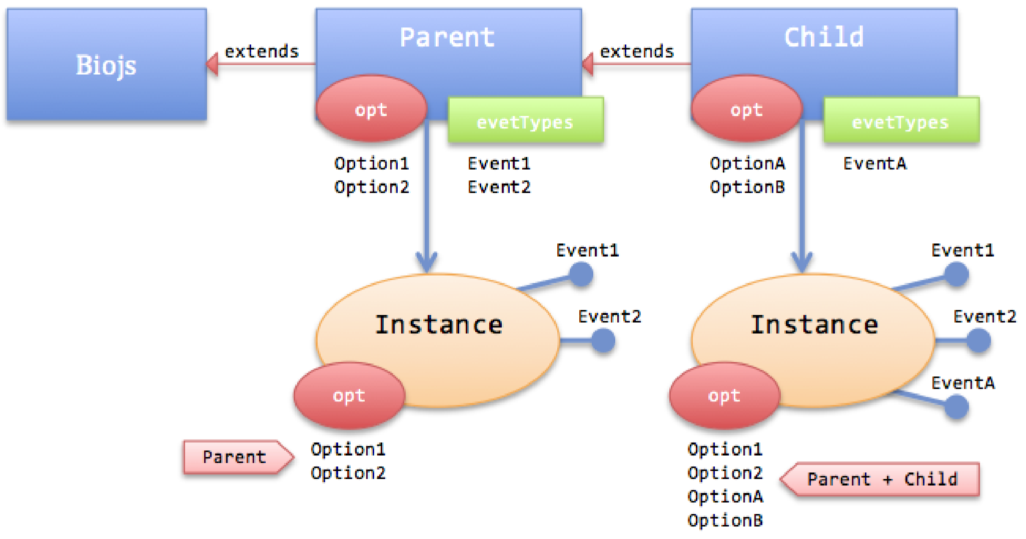


Figure 10. Example of inheritance on special members opt and eventTypes.

Lets consider a component *Parent* extending from *Biojs* and another *Child* extending from *Parent* as shown on the figure 10. *Parent* does not inherit the special members from *Biojs* because it does not exist into it. *Child* would get the *Parent*'s members, but preserving its own as they will be combined on instantiation time.

6.3.2 Defining the Methods

6.3.3 Event Handling

The events are processes originated by a component instance. They have a *name*, *listeners* and *data*. In one hand, the event names are defined in the static member class named *eventTypes*. In the other hand, both *listeners* and *data* are created dynamically; the first one by means of *addListener* method (or alias) and the last one whenever the event is raised using *raiseEvent* method.

The figure 11 shows a component with 3 events defined in *eventTypes* and its instance containing the corresponding event handlers. Biojs build these handlers automatically on instantiation time. In this way, the instance can execute *raiseEvent* method to trigger the event by passing the *data* object as argument.

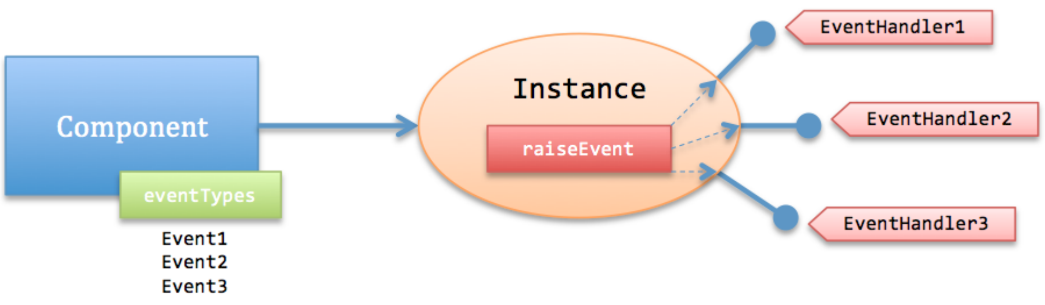


Figure 11. Event handlers are created automatically with the defined names in *eventTypes*.

At any time from any component method, an event can be triggered by following the syntax:

```
...
<methodName>: function (<argsMethod>) {
   //
   // your code before event triggering
   //
   this.raiseEvent(<eventType>, <data>);
   //
   // your code after event trigered
   //
},
...
```

Where, `<eventType>` is the name of the event to be triggered and `<data>` is the object containing the data which will be passed to the listener(s).

6.4 Compiling the code

Javascript is a interpreted language, so the code is not compilable to get some binary file. Compilation here refers to the processing based on javascript code; removing out comments and spaces, compressing variable names and generating the docummentation. Biojs uses maven plug-ins to process the source code and it is already configured in the *pom.xml*.

So, just executing the following maven command will be obtained the compiled version of the library in the *target* directory (see section 6.2):

```
mvn compile
```

Registry and *jsdoc* can be generated from the already commented code (see section 4) with the following command:

```
mvn jsdoctk:jsdoc
```

7. Glossary

Component is a chunk of a page design. A component contains generic, atomic elements (like text, links, buttons, checkboxes, and images) combined into a meaningful building block used—and reused—in the inter-face design of an entire page. Other common terms you may have heard to describe a page chunk include module, portlet, widget, or even molecule.

Dependency is a required library, program, script or other piece of code required to ensure the properly functionality of a component. (example: jQuery, YIU, jMol, PHP proxy script, etc.).