

Jetty Coding Standards

Java Code Formatting

Jetty uses the code formatting as specified in the following file:

- <http://dev.eclipse.org/svnroot/rt/org.eclipse.jetty/admin/jetty-eclipse-java-format.xml>

Java Code Templates

Jetty specifies the following code templates for use by the project developers:

- <http://dev.eclipse.org/svnroot/rt/org.eclipse.jetty/admin/jetty-eclipse-codetemplates.xml>

Java Code Conventions

The following is an example of the Java formatting and naming styles to be applied to Jetty:

```
import some.exact.ClassName;           // GOOD
import some.wildcard.package.*;       // BAD!

package org.always.have.a.package;

/*
-----
----- */
/** Always have some javadoc
 */
class MyClassName
{
    // indent by 4 spaces.
    // use spaced to indent
    // The code must format OK with default
    tabsize of 8.

    private static final int
    ALL_CAPS_FOR_PUBLIC_CONSTANTS=1;
```

```
    // Field prefixed with __ for static of  
_ for normal fields.
```

```
    // This convention is no longer  
mandatory, but any given
```

```
    // class should either consistently use  
this style or not.
```

```
    private static String __staticField;  
    private Object _privateField;
```

```
    // use getters and setters rather than  
public fields.
```

```
    public void setPrivateField(Object  
privateField)
```

```
    {  
        _privateField=privateField;  
    }
```

```
    public Object getPrivateField()
```

```
    {  
        return _privateField;  
    }
```

```
    public void doSomething()  
        throws SomeException
```

```
    {  
        Object local_variable =  
_privateField;  
        if (local_variable==null)  
        {
```

```
// do something
```

```
}  
  }  
}
```

Javascript coding standards

Javascript is used as a first class programming language, not just as a value-add to html mark up. Thus it is very important that we establish good coding practises. The following are some key points to how we wish to use javascript in the jetty project:

Formatting

The formatting should be as for javascript, with the exception that tabs may be used for indents. If tabs are used for indents, then spaces should not be used. ie all spaces or all tabs - no mixed.

Use objects rather than functions.

traditional javascript makes extensive use of functions. Where ever possible these functions should be grouped together as an Object. Thus instead of writing:

```
function asFloat(t)  
{  
    return parseFloat(t.innerHTML);  
}  
  
var dftDigits=2;  
function fixedDigits(t, digits)  
{  
    if (digits)  
        return (t.toFixed) ?  
t.toFixed(digits) : t;  
    return (t.toFixed) ?  
t.toFixed(dftDigits) : t;  
}
```

we should write:

```

var numUtil =
{
  _digits: 2,

  asFloat: function(t)
  {
    return parseFloat(t.innerHTML);
  },

  fixedDigits: function(t,digits)
  {
    if (digits)
      return (t.toFixed) ? t.toFixed(digits)
: t;
    return (t.toFixed) ?
t.toFixed(numUtil._digits) : t;
  }
}

```

Private fields and functions should have names starting with '_'.

Note that this is a very light weight form of object and we are not using any concept of class. It is equivalent to a javaclass of static members and simply groups related functions together for name space clarity.

Javascript also supports class via the prototype mechanism. This should also be used when appropriate but is not covered by the example above.

JS files as packages

Just as Javascript functions should be grouped into objects, javascript objects should be grouped into files, which can be considered the equivalent of java packages.

Use semicolon to end statements

I know it is optional, but it makes it easier for old C-hacks to parse as well as making it easier for the interpreters to spot errors if a typo is made.

Unique Id's, multiple classes

The ID of a DOM element must be unique, while multiple DOM elements can have the same class. A frequent mistake is to use a non unique id, for example on a cell in a table (if the table has multiple rows then a cell id is not unique). A class should be used if there are visual ramifications, otherwise normal attributes or external associative arrays should be used.

Understand this

The `this` automatic field in js works differently to java. If a member function of an object is called as a callback, then the `this` does not refer to the object but the calling scope. For example in the following code:

```
var myhandler =
{
  _handle: function(message)
  {
    alert(this);
  },

  register: function()
  {
    $('mybutton').onclick=this._handle;
  }
};
myhandler.register();
```

the `_handle` method is called when the `mybutton` is clicked, but the `this` will not refer to the `myhandler` object, but rather the DOM element of the button! The object `var myhandler` can be used instead of `this` in these circumstances.

Use 'single' instead of "double" quotes

Strings in js and html can be quoted with single or double quotes. We should use single quotes in the javascript and double quotes in markup. It is then simpler for js to quote double quotes when generating markup.

The exception to this is in JSON, which should always use double quotes as that is what is defined by the standard.

Use var keyword for variables inside for loop

I know this is supposed to be optional but IE does not seem to handle well for loops with variables that are not

declared with var. Thus instead of writing:

```
for (i = 0; i < tokens.length; i++) {  
    //process  
}
```

we should write:

```
for (var i = 0; i < tokens.length; i++) {  
    //process  
}
```

Use behaviour

Many libraries have a concept of behaviours, not least the behaviour library: [behaviour](#).

This is a good paradigm for separating concerns and avoiding excessive use of javascript within markup.

Behaviours should

be used where possible/suitable.