Syntax Reference

Syntax

> ✅ IOC = inversion of control
> DI = dependency injection

Jetty.xml IOC/DI has long been available even before spring sprung 🙂 (springframework)
Almost anything you can do programmatically can be done on jetty.xml, so long as it does not require loops.

The entire functionality is covered by two classes:
1. org.mortbay.xml.XmlConfiguration
2. org.mortbay.xml.XmlParser

Attributes

id

This simply means that you want to reference an object for later use.
Jetty persists the object in a *java.util.HashMap<String,Object>*

Most common use cases:

```
<New id="foo" class="com.acme.Foo"/>
// equivalent to:
// com.acme.Foo foo = new com.acme.Foo();
// instantiates with the default constructor
then persists the object

<Ref id="foo"/>
// references that object

<Call id="bar"
class="com.acme.MyStaticObjectFactory"
name="createObject"/>
// equivalent to:
//
com.acme.MyStaticObjectFactory.createObject(
);
// call the static method and the returned
object is persisted

<Configure id="Server"
class="org.mortbay.jetty.Server">
// ..
// configure this object
// tags like Set, Get, and Call are
addressed to this object
</Configure>
/** The id is persisted in a hashmap so that
more than one jetty.xml can configure the
same object.
    e.g java -jar start.jar etc/jetty.xml
etc/jetty-ssl.xml etc/jetty-plus.xml */
```

class

The value is the fully qualified class name of an object.

Used in three tags:
1. #Configure
2. #New
3. #Call

name

Generally used to specify the name of the property/method of an object.
Also used to specify the name of a System property

Used on the ff tags:
1. #Set
2. #Get
3. #Call
4. #SystemProperty

type

Used to specify the type of an #Array tag.
The value is the fully qualified class name of an object.

Used only in one tag:
1. #Array

default

Used to specify the default value of a System property if it does not exist.

Used only in one tag:
1. #SystemProperty

---

configure.dtd

This is the document type descriptor which specifies a straightforward mapping of xml elements to the java api.

The first lines in a jetty xml configuration file must reference this dtd as follows:

```
<?xml version="1.0"  encoding="ISO-8859-1"?>
<!DOCTYPE Configure PUBLIC "-//Mort Bay
Consulting//DTD Configure//EN"
"http://jetty.mortbay.org/configure.dtd">
```

Java objects are configured by a sequence of <New>, <Set>, <Put> and <Call> elements. Here is a small example to give you the flavour of it:

<Set name="Test">value</Set>	==	obj.setTest("value");
<Put name="Test">value</Put>	==	obj.put("Test","value");
<Call name="test"><Arg>value</Arg></Call>	==	obj.test("value");
<New class="com.acme.MyStuff"><Arg/></New>	==	new com.acme.MyStuff();

Values are coerced to match method arguments on a best effort approach, but explicit types may also be specified.

Jetty xml files are parsed by the org.mortbay.util.XmlConfiguration class using the configure.dtd descriptor.

<Configure> Element

This is the root element that specifies the class of object that is to be configured:

```
<Configure class="org.mortbay.jetty.Server>
. . . </Configure>
```

Alternatively, if the object already exists and has an *id*, then it can be referenced in the Configure element like so:

```
<Configure id="Server"
class="org.mortbay.jetty.Server">
```

The Configure element can contain the *New*, *Set, _Put*, *Call* or *Ref* elements.

More Examples

```
<Configure id="Server"
class="org.mortbay.jetty.Server">
// ..
// configure this object
// tags like Set, Get, and Call are
addressed to this object
</Configure>
/** The id is persisted in a hashmap so that
more than one jetty.xml can configure the
same object.
    e.g java -jar start.jar etc/jetty.xml
etc/jetty-ssl.xml etc/jetty-plus.xml */

<Configure id="Server"
class="org.mortbay.jetty.Server">
  <Set name="port">8080</Set>
</Configure>
// equivalent to:
// org.mortbay.jetty.Server server = new
org.mortbay.jetty.Server();
// server.setPort(8080);
```

<Set> Element

Sets an object property. One argument
Used in conjunction with #Configure, #New, #Ref, #Call.

This element maps to a call to a set method on the current object. The name and optional *type* attributes are used to select the set method. If the name given is xxx, then a setXxx method is used, or the xxx field is used of setXxx cannot be found.

A *Set* element can contain value text and/or the value elements *Call, New* and *SystemProperty.* If no value type is specified, then white space is trimmed out of the value. If it contains multiple value elements they are added as strings before being converted to any specified type.

A *Set* with a class attribute is treated as a static set method invocation.

```
<Set name="xyz">
 . . .
</Set>
```

More Examples

```
<Configure id="Server"
class="org.mortbay.jetty.Server">
  <Set name="port">8080</Set>
</Configure>
// equivalent to:
// org.mortbay.jetty.Server server = new
org.mortbay.jetty.Server();
// server.setPort(8080);

<New class="com.acme.Employee">
  <Set name="department">IT</Set>
</New>
// equivalent to:
// com.acme.Employee employee = new
com.acme.Employee();
// employee.setDepartment("IT");
```

<Get> Element

Returns the object's property.

This element maps to a call to a get method of field on the current object. The *name* attribute is used to select the get method. If the name given is xxx, then a getXxx method is used, or the xxx field is used of setXxx cannot be found.

A *Get* element can contain *Set, Put* and/or *Call* elements which act on the object returned by the get call.

```
<Get name="xyz">
  <Set name="xyzMethod"><New
class="org.mortbay.jetty.xyz"/></Set>
</Get>
```

A *Get* with a class attribute is treated as a static get method or field.

More Examples

```
<Configure id="Server"
class="org.mortbay.jetty.Server">
  <Get id="p" name="port"/>
  <Call class="com.acme.Environment"
name="setPort">
    <Arg>
      <Ref id="p"/>
    </Arg>
  </Call>
</Configure>
// equivalent to:
// org.mortbay.jetty.Server server = new
org.mortbay.jetty.Server();
//
com.acme.Environment.setPort(server.getPort(
));

<Configure id="Server"
class="org.mortbay.jetty.Server">
  <Call id="s" name="toString"/>
  <Call class="org.mortbay.log.Log"
```

```
name="info">
    <Arg>
      <Ref id="s"/>
    </Arg>
  </Call>
</Configure>
// equivalent to:
// Server server = new Server();
// String str = server.toString();
// org.mortbay.log.Log.info(str);

<New class="java.io.File">
  <Arg>.</Arg>
  <Get id="path" name="getAbsolutePath"/>
</New>
<Call class="org.mortbay.log.Log"
name="info">
  <Arg>
    <Ref id="path"/>
  </Arg>
</Call>
// equivalent to:
```

```
// java.io.File file = new java.io.File();
// String path = file.getAbsolutePath();
// org.mortbay.log.Log.info(path);
```

<Put> Element

This element maps to a call to a put method on the current object, which must implement the Map interface. The *name* attribute is used as the put key and the optional *type* attribute can force the type of the value.

A *Put* element can contain value text and/or the value elements *Call, New* and *SystemProperty.* If no value type is specified, the white space is trimmed out of the value. If it contains multiple value elements, they are added as strings before being converted to any specified type.

```
<Put name="admin">admin</Put>
```

<Call> Element

Calls an object method.
The method could be an instance method or a static method.

This element maps to an arbitrary call to a method on the current object. The *name* attribute and *Arg* elements are used to select the method.

A *Call* element can contain a sequence of *Arg* elements followed by a sequence of *Set*, *Put* and/or *Call* elements which act on any object returned by the original call:

```
<Call name="test">
  <Arg>value1</Arg>
  <Set name="Test">Value2</Set>
</Call>
```

This is equivalent to:

```
Object o2 = o1.test("value1");
o2.setTest("value2");
```

A Call with a *class* attribute is treated as a static call:

```
<Call class="com.acme.Foo"
name="setString">
  <Arg>somestring</Arg>
</Call>
```

Is equivalent to:

```
com.acme.Foo.setString("somestring");
```

More Examples

```
<Call class="com.acme.SomeObject"
name="init"/>
// equivalent to:
// com.acme.SomeObject.init()
// calls a static method

<Configure id="Server"
class="org.mortbay.jetty.Server">
  <Call name="getPort">
  <Call class="com.acme.Environment"
name="setPort">
    <Arg>
      <Ref id="p"/>
    </Arg>
  </Call>
</Configure>
// equivalent to:
// org.mortbay.jetty.Server server = new
org.mortbay.jetty.Server();
//
com.acme.Environment.setPort(server.getPort(
));
```

<Arg> Element

An argument of a method or a constructor.
Used in conjunction with #New and #Call.

This element defines a positional argument for the *Call* element. The optional *type* attribute can force the type of the value.

An *Arg* element can contain value text and/or the value elements *Call*, *New* and *SystemProperty*. If no value type is specified, then white space is trimmed out of the value. If it contains multiple value elements they are added as

strings before being converted to any specified type.

```
<Arg><Ref id="Server"/></Arg>
<Arg>./webapps</Arg>
<Arg>org/mortbay/jetty/webapp/webdefault.xml
</Arg>
<Arg type="boolean">True</Arg>
<Arg type="boolean">False</Arg>
```

The *Ref* tag here points to a previously created object with an *id* parameter.

More Examples

```
<Arg>foo</Arg> // String

<Arg>true</Arg> // Boolean

<Arg>1</Arg> // int, long, short, float,
double

<Arg><Ref id="foo"></Arg>  // any object
// reference a persisted object and pass it
as a paramter

<Call class="com.acme.Environment"
name="setFoo">
  <Arg>
    <New class="com.acme.Foo">
      <Arg>bar</Arg>
    </New>
  </Arg>
</Call>
```

```
// equivalent to:
com.acme.Environment.setFoo(new
com.acme.Foo("bar"));

<New class="com.acme.Baz">
  <Arg>
    <Call id="bar"
class="com.acme.MyStaticObjectFactory"
name="createObject">
      <Arg>2</Arg>
    </Call>
  </Arg>
</New>
```

```
// equivalent to: new
com.acme.Baz(com.acme.MyStaticObjectFactory.
createObject(2));
```

<New> Element

Equivalent to the "new" in java.
Instantiates an object.

This element allows the creation of a new object as part of a value of a *Set*, *Put* or *Arg* element. The *class* attribute determines the type of the new object and the contained *Arg* elements are used to select the constructor for the new object. A *New* element may have an *id* parameter which gives a unique name to the object which can be referenced later by *Ref* elements.

A *New* element can contain a sequence of *Arg* elements followed bya sequence of *Set*, *Put* and/or *Call* elements which act on the new object:

```
<New id="myobject"
class="com.acme.MyClass">
  <Arg>value1</Arg><Set
name="Test">Value2</Set>
</New>
```

This is equivalent to:

```
Object o = new
com.acme.MyClass("value1");
o.setTest("value2");
```

More Examples

```
<New class="com.acme.Foo"/>
// equivalient to:
// com.acme.Foo foo = new com.acme.Foo()
// instantiates the object with the default
constructor

<New class="com.acme.Foo">
  <Arg>bar</Arg>
</New>
// equivalient to:
// com.acme.Foo foo = new
com.acme.Foo("bar")
// instantiates the object with a
one-string-argument constructor.
```

<Ref> Element

This element allows a previously created object to be referenced by unique name.

A *Ref* element can contain a sequence of *Set*, *Put* and/or *Call* elements which act on the referenced object:

```
<Ref id="myobject" />
  <Set name="Test">Value2</Set>
</Ref>
```

Alternatively, the *Ref* element can be used as the value of an *Arg* or *Set* element:

```
<Set name="someMethod"><Ref
id="myobj"/></Set>
```

<Array> Element

An object array.

This element allows the creation of a new array as part of a value of a *Set*, *Put* or *Arg* element. The *type* attribute determines the type of the new array and the contained *Item* elements are used for each element of the array.

```
<Array type="java.lang.String">
  <Item>value0</Item>
  <Item><New
class="java.lang.String"><Arg>value1</Arg
></New></Item>
</Array>
```

This is equivalent to

```
String[] a = new String[]{"value0", new
String("value1")};
```

More Examples

```
<Array type="org.mortbay.jetty.Handler">
  <Item>
    <New id="Contexts"
class="org.mortbay.jetty.handler.ContextHand
lerCollection"/>
  </Item>
  <Item>
    <New id="DefaultHandler"
class="org.mortbay.jetty.handler.DefaultHand
ler"/>
  </Item>
  <Item>
```

```
    <New id="RequestLog"
class="org.mortbay.jetty.handler.RequestLogH
andler"/>
  </Item>
</Array>

<Array id="plusConfig"
type="java.lang.String">

<Item>org.mortbay.jetty.webapp.WebInfConfigu
ration</Item>

<Item>org.mortbay.jetty.plus.webapp.EnvConfi
guration</Item>

<Item>org.mortbay.jetty.plus.webapp.Configur
ation</Item>

<Item>org.mortbay.jetty.webapp.JettyWebXmlCo
nfiguration</Item>
```

```
<Item>org.mortbay.jetty.webapp.TagLibConfigu
ration</Item>
</Array>
```

<Map> Element

This element allows the creation of a new HashMap and to populate it with (key,value) pairs.

```
<Map>
  <Entry>
   <Item>keyName</Item>
   <Item><New
class="java.lang.String"><Arg>value1</Arg
></New></Item>
  </Entry>
</Map>
```

This is equivalent to

```
java.util.Map map = new
java.util.HashMap();
map.put("keyName", "value1"};
```

<Item> Element

An element of an array.
Used only in conjuction with #Array

This element defines an entry for the *Array* or *Map Entry* elements. The *optional* type attribute can force the type of the value.

An Item element can contain value text and/or the value elements *Call*, *New* and *SystemProperty*. If no value type is specified, then white space is trimmed out of the value. If it contains multiple value elements they are added as strings before being converted to any specified type.

```
<Item>
  <New
class="org.mortbay.jetty.nio.SelectChannelCo
nnector">
    <Set name="port">8080</Set>
    <Set name="maxIdleTime">30000</Set>
    <Set name="Acceptors">10</Set>
 </New>
</Item>
```

<SystemProperty> Element

Gets the value of a system property.

This element allows JVM System properties to be retrieved as part of the value of a *Set*, *Put* or *Arg* element. The name attribute specifies the property name and the optional default argument provides a default value.

```
<SystemProperty name="Test"
default="value"/>
```

This is equivalent to:

```
System.getProperty("Test","value");
```

More Examples

```
<SystemProperty name="jetty.port"
default="8080"/>
// equivalent to:
System.getProperty("jetty.port", "8080");
// means that if "jetty.port" is not set,
"8080" will be used.
```