

# Annotations

## "Beta Feature"

This is a new feature slated to be delivered in Jetty 6.1.2rc0, but can be used by checking out and locally building svn trunk.

## Servlet 2.5 Annotations, Resource Injections and Callbacks

The [2.5 Servlet Specification](#) adds the ability to inject JNDI resources into fields and methods of servlets, filters and listeners, and also to perform certain callbacks at various points in the lifecycle of a web application.

JNDI resource injection and the lifecycle callbacks can be specified entirely within the web.xml file, or alternatively marked up as [annotations](#) in your source code. You can even use a combination of annotations and web.xml declarations.

One important thing to be aware of is that the 2.5 Servlet Specification has introduced a new attribute into the <web-app> element, the **metadata-complete** attribute. If **true**, then the web container will NOT search the webapp for source code annotations, and your web.xml file must contain all resources required. If **false** or not specified, then jetty is required to examine all servlets, filters and listeners in the webapp for annotations. Therefore, you can save startup time by using this attribute correctly - if you don't want to use annotations then ensure you mark **metadata-complete="true"**, otherwise you will pay the penalty of the code examination.

```
<web-app
  xmlns="http://java.sun.com/xml/ns/j2ee"

  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"

  xsi:schemaLocation="http://java.sun.com/xml/
  ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_5.
  xsd"
  metadata-complete="false"
  version="2.5">
```

Jetty supports the following injections and callback types:

- **@Resource** equivalent to <resource-ref>, <resource-env-ref>, <env-entry> and <message-destination-ref> in web.xml
- **@Resources** declares java:comp/env name linkages for reference by JNDI lookups

- **@PostConstruct** equivalent to **<post-construct>** in web.xml
- **@PreDestroy** equivalent to **<pre-destroy>** in web.xml
- **@RunAs** equivalent to **<run-as>** in web.xml

Lets look first at how you can use web.xml to declare injections and callbacks without needing to use [JDK1.5 annotations](#).

## Resource Injections and Callbacks specified in web.xml

### Introduction

The Servlet 2.5 web.xml schema has been augmented to support the declaration of resource injections and lifecycle callbacks. We'll look at a couple of small examples of how to declare them in web.xml, but for more in-depth information, you should consult the [Common Annotations for the Java Platform Specification \(JSR250\)](#), the [Servlet 2.5 Specification \(JSR154\)](#) and the [JavaEE Specification v5 \(JSR244\)](#).

Here is a small example of the injection of a DataSource resource:

```
<resource-ref>

<res-ref-name>jdbc/mydatasource</res-ref-name>

<res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <injection-target>

<injection-target-class>com.acme.MyServlet</injection-target-class>

<injection-target-name>myDatasource</injection-target-name>
  </injection-target>
</resource-ref>
```

This example shows that the resource named `java:comp/env/jdbc/mydatasource` will be injected by Jetty into the the field named `myDatasource` or the method named `setMyDatasource(javax.sql.DataSource)` in the instance of the class `com.acme.MyServlet` before it goes into service.

Similarly, values for <env-entry>s can be injected:

```
<env-entry>

<env-entry-name>maxAmount</env-entry-name>
  <env-entry-value>99.99</env-entry-value>

<env-entry-type>java.lang.Double</env-entry-
type>
  <injection-target>

<injection-target-class>com.acme.MyServlet</
injection-target-class>

<injection-target-name>maxAmount</injection-
target-name>
  </injection-target>
</env-entry>
```

This example shows the value 99.99 being injected into the field named `maxAmount` (or the method named `setMaxAmount(Double)`) in the instance of the class `com.acme.MyServlet` before it goes into service.

The [Servlet 2.5 Specification \(JSR154\)](#) also introduces the concept of lifecycle callbacks. These are of two types: a post-construction callback, and a pre-destruction callback. The former will be invoked after all resource injections have been performed on an instance of a managed class (eg servlet, filter or listener) but *before* it goes into service. The latter is invoked just before the container removes the instance from service. Let's look at how you might declare these callbacks in a `web.xml` file:

```
<post-construct>
```

```
<lifecycle-callback-class>com.acme.MyServlet  
</lifecycle-callback-class>
```

```
<lifecycle-callback-method>myPostConstructCa  
llback</lifecycle-callback-method>  
</post-construct>
```

```
<pre-destroy>
```

```
<lifecycle-callback-class>com.acme.MyServlet  
</lifecycle-callback-class>
```

```
<lifecycle-callback-method>myPreDestroyCallb  
ack</lifecycle-callback-method>  
</pre-destroy>
```

The above example would invoke the method `myPostConstructCallback()` on the instance of the class `com.acme.MyServlet` before the servlet is put into service, and would invoke the method `myPreDestroyCallback()` on the instance before it goes out of service.

## How to Configure in Jetty

To use injections and callbacks declared in `web.xml`, you simply follow the instructions for setting up [JNDI](#). Once your webapp is configured for JNDI access (ie you have followed the instructions in the first section of the [JNDI](#) page), and you have defined the resources you want to access in a jetty configuration file (such as a `jetty.xml` or a `WEB-INF/jetty-env.xml` file as described on the [JNDI](#) page), you can reference them in your `web.xml` file.

## Example Web Application

The `examples/test-jndi-webapp` webapp in the Jetty distribution shows you how to go about this.

## Resource Injections and Callbacks with Annotations

### Introduction

The [Servlet 2.5 Specification \(JSR154\)](#) allows the use of annotations instead of/in addition to declarations in the web.xml file.

Let's see what the equivalent annotations would be for the examples we gave for the web.xml markup.

Here's the equivalent of the injection of the DataSource:

```
public class MyServlet
{
    private DataSource myDS;

    @Resource(mappedName="jdbc/mydatasource")
    public void setMyDatasource(DataSource
ds)
    {
        myDS=ds;
    }

    ...

    public Connection getConnection ()
    {
        return myDS.getConnection();
    }
}
```

In this case, we've chosen to put the annotation on a setter method, but equally we could have chosen to put it onto a field instead. As you can see, at runtime, you can simply use the data member `myDS` to obtain a `Connection` because Jetty will have injected the field with the `DataSource` you configured named `jdbc/mydatasource` in a `jetty.xml` or `WEB-INF/jetty-env.xml` file.

Here's the equivalent for the `EnvEntry`:

```
public class MyServlet
{
    @Resource(mappedName="maxAmount")
    private Double wiggle;
}
```

Finally, here are the lifecycle callback equivalents:

```
public class MyServlet
{
    @PostConstruct
    private void myPostConstructCallback ()
    {
        System.err.println("PostConstruct
callback done");
    }

    @PreDestroy
    private void myPreDestroyCallback()
    {
        System.err.println("PreDestroy
callback done");
    }
}
```

Refer to the [Common Annotations for the Java Platform Specification \(JSR250\)](#) and the [JavaEE Specification v5 \(JSR244\)](#) for more extensive examples, and for precise rules on what source constructs can be marked with annotations.

## How to Configure in Jetty

Firstly, read the information on setting up [JNDI](#). Then apply the following configuration to instruct Jetty to search for annotations in the source code of your web application:



```
<Array id="annotationConfig"
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.annotations.Configur
ation</Item> <!-- Enables annotation support
-->

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

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

...

<New id="xyzWebAppContext"
class="org.mortbay.jetty.webapp.WebAppContex
t">
    ...
    <Set name="ConfigurationClasses"><Ref
id="annotationConfig/></Set>
    ...
</New>
```

Once you've enabled annotation support for your webapp, and followed the instructions in [JNDI](#) to set up references to resources you want to access at runtime (eg `org.mortbay.jetty.plus.naming.EnvEntry` for environment values, `org.mortbay.jetty.plus.naming.Resource` for DataSources, Queues, Topics etc) you can go ahead and mark up your code.

## **Example Web Application**

The `examples/test-annotations` web application can be built and deployed to demonstrate how to use annotation markup in your code.