How to run jetty

*

*

Basically you can run jetty in two different modes:

1. Interactively: use s.th. like `buildr yourproject:run-jetty` and have the jetty running within this shell, so that you see the output of jetty and your app
2. In the background: use `buildr jetty:start` and `buildr yourproject:deploy-app` to start jetty in the background and deploy your app. The build is not blocking in this case and you can use your shell for different things.

Run jetty interactively

In your buildfile you define the jetty task within your project (here it's "webapp"):

```
require 'buildr/jetty'
require 'readline'

define "webapp" do

  ....

  task("jetty"=>[package(:war), jetty.use]) do |task|
    jetty.deploy("http://localhost:8080", task.prerequisites.first)
    Readline::readline('[Type ENTER to stop Jetty]')
  end

end
```

If you prefer to use `CTRL-C` to stop jetty instead of hitting `ENTER` you can replace the `Readline::...` line with this:

```
puts 'Press CTRL-C to stop Jetty'
trap 'SIGINT' do
  jetty.stop
end
Thread.stop
```

Then you can run your webapplication inside jetty:

```
$ buildr webapp:jetty
```

That's all

Start jetty in the background

In your buildfile you define the "deploy-app" task within your project (here it's "webapp"):

```
require 'buildr/jetty'

define "webapp" do

  ....

  task("deploy-app"=>[package(:war), jetty.use]) do |task|
    class << task ; attr_accessor :url, :path ; end
    task.url = "http://localhost:8080"
    task.path = jetty.deploy(task.url, task.prerequisites.first)
  end

end
```

Now, in one console do:

```
$ buildr jetty:start
```

Switch to a different console and do:

```
$ buildr webapp:deploy-app
```

Now the app is running and you can access it from the browser. The first console is effectively the Jetty log, and you kill Jetty at any time by going there and hitting CTRL-C. The second console gives you a new prompt and you can run buildr deploy-app again, test the app using curl, etc.

Alternatively, in a single console:

```
$ buildr jetty:start &
$ buildr webapp:deploy-app
```

And again this free up the console so you can do more work while your app is running inside Jetty.

To stop the server:

```
$ buildr jetty:stop
```

Run several jetty instances in different modules

If you have a multi module project with several web applications (producing a `war`) you might want to start them in different jetty instances (on different ports).

To achieve this you cannot use the default jetty singleton instance (as shown in the examples above) but you must create a new jetty instance for the different web applications (or at least starting with the second 🙂). The following example demonstrates this:

```
define "myproj" do

  define "subprojA" do
    ...
    task("jetty"=>[package(:war), jetty.use]) do |task|
      jetty.deploy("http://localhost:8080", task.prerequisites.first)
      Readline::readline('[Type ENTER to stop Jetty]')
    end

  end

  define "subprojB" do

    myJetty = Buildr::Jetty.new("webapp", "http://localhost:8090")
    task("jetty"=>[package(:war), myJetty.use]) do |task|
      myJetty.deploy("http://localhost:8090", task.prerequisites.first)
      Readline::readline('[Type ENTER to stop Jetty]')
    end

  end

end
```

Now you can start both jetty instances in parallel.
Starting jetty for the first subproject:

```
$ buildr myproj:subprojA:jetty &
```

And after the subprojA jetty was started (or in a different console):

```
$ buildr myproj:subprojB:jetty
```

Run embedded jetty on working directory

The suggestions above will always have you run through a complete compile -> package -> deploy cycle, even if you are only editing Javascript files or did a minor small change in a sub project, which gets annoying pretty fast. Use the embedded server and run it directly on your working directory instead to save all that packaging time and to edit HTML in place.

Add a source file to start embedded jetty using your webapp configuration (Scala code here, but you can change that to Java easily):

```
import org.mortbay.jetty._
import org.mortbay.jetty.webapp.WebAppContext

object MyServer {
        def main(args: Array[String]) = {
                val server = new Server(8080)
                val webAppContext = new WebAppContext("src/main/webapp", "/")
                webAppContext.setConfigurationClasses(Array[String](
                        "org.mortbay.jetty.webapp.WebInfConfiguration",
                        "org.mortbay.jetty.webapp.WebXmlConfiguration"
                ))
                server.addHandler(webAppContext)
                server.start
                server.join
        }
}
```

Next add a run task inside of your project:

```
define "myproject" do

  desc 'Run application with embedded jetty'
  task :run => :compile do
          Java::Commands.java('MyServer', :classpath => compile.dependencies + [compile.target.to_s])
  end

end
```

This will set up your class path with the compile dependencies and run the java command with your server class.

Starting the application:

```
$ buildr myproject:run
```

That's it.