

# Porting to jetty6

## Porting from Jetty [345].x to Jetty 6

For many developers, there should be little or no porting to be done to make a web application work on Jetty 6. If the application adheres to the standards, it should simply be a matter of configuration.

However, if the application uses non standard APIs from earlier version of Jetty or from other containers, then some porting work will be required. This page gives a brief overview of the changes needed to port to Jetty 6.

## Architectural Changes

There have been two major changes in the architecture of Jetty 6, both intended to improve the modularity and allow interceptor style extension.

### Merged HTTP Request API and Servlet Request API

In Jetty <=5, there was an API for pure Jetty HTTP requests and responses. The Jetty requests and responses were wrapped by the ServletHandler to provide the Servlet API for requests and responses. This architecture allowed for handlers to be written without servlet dependencies. However, given the re-entrant nature of Request Dispatches and the increasing involvement of the servlet spec in all common handlers (resources, security, etc.). This Jetty/Servlet duality became an unwanted complexity when extending the server.

In Jetty 6, all requests and responses are based on the Servlet APIs requests and responses. This imposes a 136k JAR dependency on all jetty runtimes, but allows for significant reduction in the Jetty jar itself, as a parallel API is not required and much complexity is removed. **Importantly** this does not mean that all requests are servlet requests and must be delivered by servlets. It simply means that jetty reuses the Servlet API as its own API for common methods such as `getParameter(String)`, `getOutputStream()` and `getQueryString()`.

All Servlet facility, such as contexts, session, security, filters and servlets themselves are options. Only when the appropriate handlers are configured will the relevant parts of the servlet request/response API be active. For example:

- Without a SessionHandler the `getSession()` will always return null.
- Without a ContextHandler the `getContext()` path will always return null and the complete URI path will be available via `getPathInfo()`

### Handlers are interceptors/filters.

In Jetty <=5, handlers were called in sequence, one after the other, until the request is marked as handled. This allowed for shallow calling stacks, but did not allow for simple and safe before/after handling using `try {} finally {}` constructs.

In Jetty 6, All handlers are formed into a strict containment tree rooted to the server and the normal arrangement is for handlers to implement their aspect of handling and then delegate the request to a contained handler. For a full configuration, the handling typically is as follows:

1. The [Server](#) delegates the request to the its configured handler, which is normally an instance of [HandlerCollection](#).
2. The [HandlerCollection](#) acts like Jetty 5, and calls each of its contained handlers in turn. Typically it is configured with a [ContextHandlerCollection](#), a [DefaultHandler](#) and a [RequestLogHandler](#). This allows a request to be handled

- by a context or the default handler, and then be passed to the request log handler.
3. The [ContextHandlerCollection](#) maintains a map of context path to lists of [ContextHandlers](#). For a given request, each the URI is used to find matching context paths, and each is called in turn until the request is handled.
  4. If the ContextHandler accepts a request (it may reject it due to virtual hosts), it will delegate the request to a nested handler and for the duration of that call it will set:
    - the request context path
    - the current thread context classloader
    - the resource base
    - the error handler
    - context attributes and init parameters.
  5. Typically the ContextHandler will be an instance of [WebAppContext](#) and will thus contain a nested chain of SessionHandler, SecurityHandler and ServletHandler. The request is delegated to the first handler in this chain, a [SessionHandler](#)
  6. The [SessionHandler](#) will examine the request for any session ID to be activated and will activate the mechanism for creating new sessions before delegating the request to the [SecurityHandler](#).
  7. The [SecurityHandler](#) will check any constraints and authentication before delegating the request to the [ServletHandler](#)
  8. The [ServletHandler](#) implements the dispatch to Filters and Servlets to handle the request according to the servlet specification.

## Servlet 2.5 and JSP 2.1

Jetty 6 implements the 2.5 servlet specification. There is nothing revolutionary in this update of the API and mostly represents API cleanups and corrections. The upgrade to JSP 2.1 is slightly more significant as this requires JAVA 1.5 (hence jetty retains JSP 2.0 as an option).

## Renaming of classes and packages.

Jetty 6 took the opportunity to clean up the package hierarchy and to remove some long deprecated methods. Several classes and packages have been renamed:

JETTY 5 CLASS	MATCHING JETTY 6 CLASS	SIMILAR JETTY 6 CLASS
org.mortbay.util.Resource	org.mortbay.resource.Resource	
org.apache.commons.logging		org.mortbay.log.*
org.mortbay.http.*	org.mortbay.jetty.*	
org.mortbay.http.HttpServletRequest		org.mortbay.jetty.Request
org.mortbay.http.HttpServletResponse		org.mortbay.jetty.Response
org.mortbay.http.HttpHandler		org.mortbay.jetty.Handler
org.mortbay.http.HttpContext		org.mortbay.jetty.handler.ContextHandler

org.mortbay.http.HttpListener	org.mortbay.jetty.Connector	
org.mortbay.http.HttpServer		org.mortbay.jetty.Server
org.mortbay.util.ThreadPool		org.mortbay.thread.ThreadPool

The pain that is commons logging has been removed and Jetty now has no hard dependency on any logging mechanism. If an SLF4J jar is found, it will be used, otherwise any logging is simply sent to stderr.

## Build and packaging

Jetty 6 is built with maven, which has changed the way the jars are bundled.

**MORE DETAIL HERE**