# Jetty Components

# 2 Feb 2011

# Jetty Components

*2 Feb 2011*

## 1 Introduction

Jetty component provides the functionality of starting a Jetty Server with and without default configurations of listeners, webapplications and servlets. It also provides the feature of dynamically configuring a running Jetty HttpServer with additional listeners, web applications and servers.

It provides instantiable extensions to the www components, which must also be on the classpath of this component's deployment.

## 2 Installation

The Jetty components are packaged in the sf-jetty JAR file. They have the following dependencies at run time

| Artifact | Organisation | Version | Description |
|---|---|---|---|
| sf-www | | (in sync) | www base classes |
| servletapi | javax.servlet | 2.5 | Servlet API. Jetty now requires this. |
| jetty | org.mortbay.jetty | 6.1.5 | |
| jetty-util | org.mortbay.jetty | 6.1.5 | |
| | | | |

To pick up the JSP artifacts from the ivy.xml file, use the "runtime" configuration

### 2.1 JSP support

For JSP Support, the following artifacts are required.

| Artifact | Organisation | Version | Description |
|---|---|---|---|
| commons-logging | commons-logging | 1.0.4 | |
| core | org.eclipse.jdt | 3.1.1 | Eclipse javac compiler |
| ant | org.apache.ant | 1.6.5 or later | Bridge to JDK compiler |
| jsp-2.1 | org.mortbay.jetty | 6.1.5 | |
| jsp-api-2.1 | org.mortbay.jetty | 6.1.5 | |

We are not convinced that the ant artifact is needed, given that the Eclipse JDT compiler is a javac compiler. However, we do currently include it in our ivy metadata.

To pick up the JSP artifacts from the ivy.xml file, use the "jsp" configuration

## 3 Components

The Jetty package has different components for the core jetty server, jetty server with admin privileges, listeners, webapplications, servletcontexts, handlers and servlets.

These are all declared in the SmartFrog file /org/smartfrog/services/jetty/components.sf,

```
/org/smartfrog/services/jetty/components.sf
```

### 3.1.1 CoreJettyServer

| Attribute Name | Description | Optional/Mandatory |
|---|---|---|
| jettyhome | Jetty home path.<br>This is essentially the base directory for operations; | Mandatory |
| host | Host on which the Jetty server runs | Default "localhost"; |
| enableLogging | Boolean to keep logging on or off | Default false |
| maxThreads extends Integer | Maximum number of worker threads | Mandatory, but a default is declared |
| minThreads extends Integer | Minimum number of worker threads | Mandatory, but a default is declared |
| maxIdleTime extends Integer | How long in milliseconds should an idle thread wait before being terminated | Mandatory, but a default is declared |
| sendDateHeader | Should a date header be included in responses? | Default true |
| sendServerVersion | should the server version be sent in a response? | Default true |
| stopAtShutdown | set this to add a flag to shut the server down cleanly during app termination. Not normally needed, as SmartFrog has its own shutdown routines. | Default false |
| logAppend | Should the log be appended to any existing files? | Default true |
| logDir | Directory for log files. If a relative directory path is supplied, it is resolved relative to jettyhome. | Default is declared as log/ |
| logExtended | Should an extended format log be created? | Default true |
| logKeepDays | Days to keep the log | Default 7 |
| logIgnorePaths | List of paths to not log | Default [] |
| logTimezone | Time zone for events in the log | Default "GMT" |
| logPattern | pattern for log files | Default "yyyy_mm_dd.request.log" |

The three thread attributes, maxThreads, minThreads and maxIdleTime,control how many threads are available in a worker thread pool. The maximum number of simultaneous requests that can be processed is limited by these values. Furthermore, if socket connectors are used, extra threads are needed to listen on the sockets for incoming requests. These threads come from the same pool of threads. If a Jetty-hosted web site appears to hang, it may be that there are not enough threads available.

### *3.1.2 HttpServer*

# 4 Logging

Jetty6 uses *Simple Logging 4 Java*, SLF4J, which is bound to a logging implementation by the inclusion of a specific back end JAR for the chosen logging framework. As Ivy 2.0.0-alpha-2 does not parse the SLF4J metadata files (this will be fixed in the release), we do not yet include this log in our release, so jetty logs to the console instead.

We do provide a class that directly bridges from the Jetty logging APIs to the SmartFrog logger. This can be loaded by including the specific SmartFrog file `/org/smartfrog/services/jetty/log/components.sf`,

```
#include "/org/smartfrog/services/jetty/log/components.sf"
```

This descriptor is automatically loaded in the `/org/smartfrog/services/jetty/components.sf` file.

The log descriptor defines a component that sets up the JVM property `org.mortbay.log.class` to the value `org.smartfrog.services.jetty.log.JettyLogger`:

```
IntegrateJettyLogging extends SystemProperties {
   properties [
    ["org.mortbay.log.class","org.smartfrog.services.jetty.log.JettyLogger"]
   ];
   setOnEarlyDeploy true;
   setOnDeploy false;
   unsetOnTerminate false;
}
```

When Jetty starts up, it reads this property (once) to create a system log and a log factory; the factory is used for creating all further logs. The component must be deployed before any Jetty components; the `setOnEarlyDeploy` attribute ensures that the properties are set very early on in the deployment process, before any sibling components have been deployed or started.

An alternative way to configure Jetty is to set the same system property when the JVM is started. This can be done in the SmartFrog root process by editing `default.ini`.

Once set, the logging is routed through SmartFrog for the duration of the life of the JVM. New deployments to the same JVM will retain the original binding, because the

# 5 Changes

In SmartFrog 3.12.008, Jetty support moved from Jetty4 to Jetty6. This is a fundamental change in the core codebase, but offers many advantages

- An up to date servlet API
- A new handler architecture
- Better JSP support

It did require some major changes internally, some of which are visible

- All predefined 'demo' servlets, servlets that came with a full Jetty installation, have been removed.
- The Jetty administrator component, which brought up a secured administration page, has been removed. This feature is no longer in Jetty.

The code update triggered further changes to the components

- Request logging can now be configured more completely
- Thread pool sizes can be defined for the jetty engine, and for the socket listener
- A `JettySelectChannelConnector` component offers access to the threadless nio-based socket. This is

better for listening for new connections.

- Some of the components have been renamed to match the renamings of Jetty internals. Specifically, the `JettySocketConnector` is the name of the component that listens on sockets; its attributes for port and host are `port` and `host` respectively. A `JettyListener` component is defined that extends this class, retaining the old name and old attributes, `listenerPort` and `serverHost`.
- A `JettySSLSocketConnector` component is added to allow SSL Connections to be declared
- The servlet context no longer automatically supports contexts relative to jettyhome. If you want to deploy a directory relative to another location, use `FileComponent` declarations to define the hierarchy.
- Any attempt to deploy a nonexistent file or directory will result in a failure to start the web application or servlet application.
- Bindings from the servlet context and web application context to specific filenames are postponed to the start phase, instead of during the deploy period of the component lifecycle. This eliminates race conditions between peer components.

Changes that may break existing code

- Jetty no longer serves up directory trees. An empty directory results in an HTTP Error code.
- You must have a welcome files section in the `WEB-INF/web.xml` file to make index pages visible. As an example. here is the file list at the end of our test war

```
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```