26. Logging

Spring Boot uses Commons Logging for all internal logging, but leaves the underlying log implementation open. Default configurations are provided for Java Util Logging, Log4J, Log4J2 and Logback. In each case loggers are pre-configured to use console output with optional file output also available.

By default, If you use the 'Starter POMs', Logback will be used for logging. Appropriate Logback routing is also included to ensure that dependent libraries that use Java Util Logging, Commons Logging, Log4J or SLF4J will all work correctly.

There are a lot of logging frameworks available for Java. Don't worry if the above list seems confusing. Generally you won't need to change your logging dependencies and the Spring Boot defaults will work just fine.

26.1 Log format

The default log output from Spring Boot looks like this:

```
2014-03-05 10:57:51.112  INFO 45469 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine:
2014-03-05 10:57:51.253  INFO 45469 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embe
2014-03-05 10:57:51.253  INFO 45469 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationConte
2014-03-05 10:57:51.698  INFO 45469 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean        : Mapping servlet: 'dispat
2014-03-05 10:57:51.702  INFO 45469 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  : Mapping filter: 'hiddenH
```

The following items are output:

- Date and Time — Millisecond precision and easily sortable.
- Log Level — `ERROR`, `WARN`, `INFO`, `DEBUG` or `TRACE`.
- Process ID.
- A `---` separator to distinguish the start of actual log messages.
- Thread name — Enclosed in square brackets (may be truncated for console output).
- Logger name — This is usually the source class name (often abbreviated).
- The log message.

Logback does not have a `FATAL` level (it is mapped to `ERROR`)

26.2 Console output

The default log configuration will echo messages to the console as they are written. By default `ERROR`, `WARN` and `INFO` level messages are logged. You can also enable a "debug" mode by starting your application with a `--debug` flag.

```
$ java -jar myapp.jar --debug
```

you can also specify `debug=true` in your `application.properties`.

When the debug mode is enabled, a selection of core loggers (embedded container, Hibernate and Spring) are configured to output more information. Enabling the debug mode does *not* configure your application log all messages with `DEBUG` level.

26.2.1 Color-coded output

override the auto detection.

Color coding is configured using the `%clr` conversion word. In its simplest form the converter will color the output according to the log level, for example:

```
%clr(%5p)
```

The mapping of log level to a color is as follows:

Level	Color
`FATAL`	Red
`ERROR`	Red
`WARN`	Yellow
`INFO`	Green
`DEBUG`	Green
`TRACE`	Green

Alternatively, you can specify the color or style that should be used by providing it as an option to the conversion. For example, to make the text yellow:

```
%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){yellow}
```

The following colors and styles are supported:

- `blue`
- `cyan`
- `faint`
- `green`
- `magenta`
- `red`
- `yellow`

26.3 File output

By default, Spring Boot will only log to the console and will not write log files. If you want to write log files in addition to the console output you need to set a `logging.file` or `logging.path` property (for example in your `application.properties`).

The following table shows how the `logging.*` properties can be used together:

Table 26.1. Logging properties

`logging.file`	`logging.path`	Example	Description
(none)	*(none)*		Console only logging.
Specific file	*(none)*	`my.log`	Writes to the specified log file. Names can be an exact location or relative to the current directory.
(none)	Specific directory	`/var/log`	Writes `spring.log` to the specified directory. Names can be an exact location or relative to the current directory.

The logging system is initialized early in the application lifecycle and as such logging properties will not be found in property files loaded via `@PropertySource` annotations.

Logging properties are independent of the actual logging infrastructure. As a result, specific configuration keys (such as `logback.configurationFile` for Logback) are not managed by spring Boot.

26.4 Log Levels

All the supported logging systems can have the logger levels set in the Spring `Environment` (so for example in `application.properties`) using 'logging.level.*=LEVEL' where 'LEVEL' is one of TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF. The `root` logger can be configured using `logging.level.root`. Example `application.properties`:

```
logging.level.root=WARN
logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=ERROR
```

By default Spring Boot remaps Thymeleaf `INFO` messages so that they are logged at `DEBUG` level. This helps to reduce noise in the standard log output. See `LevelRemappingAppender` for details of how you can apply remapping in your own configuration.

26.5 Custom log configuration

The various logging systems can be activated by including the appropriate libraries on the classpath, and further customized by providing a suitable configuration file in the root of the classpath, or in a location specified by the Spring `Environment` property `logging.config`.

Since logging is initialized **before** the `ApplicationContext` is created, it isn't possible to control logging from `@PropertySources` in Spring `@Configuration` files. System properties and the conventional Spring Boot external configuration files work just fine.)

Depending on your logging system, the following files will be loaded:

Logging System	Customization
Logback	`logback-spring.xml`, `logback-spring.groovy`, `logback.xml` or `logback.groovy`
Log4j	`log4j-spring.properties`, `log4j-spring.xml`, `log4j.properties` or `log4j.xml`
Log4j2	`log4j2-spring.xml` or `log4j2.xml`
JDK (Java Util Logging)	`logging.properties`

When possible we recommend that you use the `-spring` variants for your logging configuration (for example `logback-spring.xml` rather than `logback.xml`). If you use standard configuration locations, Spring cannot completely control log initialization.

There are known classloading issues with Java Util Logging that cause problems when running from an 'executable jar'. We recommend that you avoid it if at all possible.

To help with the customization some other properties are transferred from the Spring `Environment` to System properties:

Spring Environment	System Property	Comments

`logging.exception-conversion-word`	`LOG_EXCEPTION_CONVERSION_WORD`	The conversion word that's used when logging exceptions.
`logging.file`	`LOG_FILE`	Used in default log configuration if defined.
`logging.path`	`LOG_PATH`	Used in default log configuration if defined.
`logging.pattern.console`	`CONSOLE_LOG_PATTERN`	The log pattern to use on the console (stdout). (Not supported with JDK logger.)
`logging.pattern.file`	`FILE_LOG_PATTERN`	The log pattern to use in a file (if LOG_FILE enabled). (Not supported with JDK logger.)
`logging.pattern.level`	`LOG_LEVEL_PATTERN`	The format to use to render the log level (default `%5p`). (The `logging.pattern.level` form is only supported by Logback.)
`PID`	`PID`	The current process ID (discovered if possible and when not already defined as an OS environment variable).

All the logging systems supported can consult System properties when parsing their configuration files. See the default configurations in `spring-boot.jar` for examples.

If you want to use a placeholder in a logging property, you should use Spring Boot's syntax and not the syntax of the underlying framework. Notably, if you're using Logback, you should use `:` as the delimiter between a property name and its default value and not `:-`.

You can add MDC and other ad-hoc content to log lines by overriding only the `LOG_LEVEL_PATTERN` (or `logging.pattern.level` with Logback). For example, if you use `logging.pattern.level=user:%X{user} %5p` then the default log format will contain an MDC entry for "user" if it exists, e.g.

```
2015-09-30 12:30:04.031 user:juergen INFO 22174 --- [  nio-8080-exec-0] demo.Controller Handling authenticated request
```

26.6 Logback extensions

Spring Boot includes a number of extensions to Logback which can help with advanced configuration. You can use these extensions in your `logback-spring.xml` configuration file.

You cannot use extensions in the standard `logback.xml` configuration file since it's loaded too early. You need to either use `logback-spring.xml` or define a `logging.config` property.

26.6.1 Profile-specific configuration

The `<springProfile>` tag allows you to optionally include or exclude sections of configuration based on the active Spring profiles. Profile sections are supported anywhere within the `<configuration>` element. Use the `name` attribute to specify which profile accepts the configuration. Multiple profiles can be specified using a comma-separated list.

```
<springProfile name="staging">
    <!-- configuration to be enabled when the "staging" profile is active -->
</springProfile>

<springProfile name="dev, staging">
```

```
</springProfile>

<springProfile name="!production">
    <!-- configuration to be enabled when the "production" profile is not active -->
</springProfile>
```

26.6.2 Environment properties

The `<springProperty>` tag allows you to surface properties from the Spring `Environment` for use within Logback. This can be useful if you want to access values from your `application.properties` file in your logback configuration. The tag works in a similar way to Logback's standard `<property>` tag, but rather than specifying a direct `value` you specify the `source` of the property (from the `Environment`). You can use the `scope` attribute if you need to store the property somewhere other than in `local` scope.

```
<springProperty scope="context" name="fluentHost" source="myapp.fluentd.host"/>
<appender name="FLUENT" class="ch.qos.logback.more.appenders.DataFluentAppender">
    <remoteHost>${fluentHost}</remoteHost>
    ...
</appender>
```

The `RelaxedPropertyResolver` is used to access `Environment` properties. If specify the `source` in dashed notation (`my-property-name`) all the relaxed variations will be tried (`myPropertyName`, `MY_PROPERTY_NAME` etc).