**Log4j 2.x version**

Log4j API that can be used to create logging infra structure in your tests.

**Why logging is important in any application?**

Logging is very important to any application. It helps us is quick debugging, easy maintenance by collecting information about the execution.

**Advantages of Log4j**

* Log4j allows you to have a very good logging infrastructure with minimal efforts.
* Allows categorizing logs at different logging levels (Trace, Debug, Info, Warn, Error and Fatal).
* Provides control to format the output of the logs.
* It has multiple appenders styles, which allows to direct logs to  different outputs styles like a file, console or a database.
* Logging can be set at runtime using configuration files.

Log4j consists of three main components

**Logger**  This is a class, which helps you log information at different logging levels.

**Appenders**  Appenders are objects, which help Logger objects write logs to different outputs. Appenders can specify a file, console or a database as the output location.

**Layouts**  Layout class helps us define how the log information should appear in the outputs.

<https://logging.apache.org/log4j/2.0/download.html>

<https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core/2.7>

<https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api/2.7>

Log4j will check the system property “log4j.configurationFile” for the configuration file path.

In case **no system property is defined** the configuration order takes below precedence:

* Property ConfigurationFactory will look for log4j2-test.properties in the classpath.
* YAML ConfigurationFactory will look for **log4j2-test.yaml** or **log4j2-test.yml** in the classpath.
* JSON ConfigurationFactory will look for **log4j2-test.jsn**or **log4j2-test.json**in the classpath.
* XML ConfigurationFactory will look for **log4j2-test.xml**in the classpath.
* Property ConfigurationFactory will look for log4j2.properties on the classpath
* YAML ConfigurationFactory will look for **log4j2.yml** or **log4j2.yaml** in the classpath.
* JSON ConfigurationFactory will look for **log4j2.jsn** or **log4j2.json** in the classpath.
* XML ConfigurationFactory will look for **log4j2.xml** in the classpath.
* If no configuration file was provided, the **DefaultConfiguration**takes place and that would lead you for set of default behaviors:
  + Root logger will be used.
  + Root logger level will be set to **ERROR**.
  + Root logger will propagate logging messages into console.
  + PatternLayout is set to be %d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n
* **ALL**
* **TRACE**
* **DEBUG**
* **INFO**
* **WARN**
* **ERROR**
* **FATAL**

|  |  |
| --- | --- |
| **Log Level** | **When It Should Be Used** |
| **OFF** | When no events will be logged |
| **FATAL** | When a severe error will prevent the application from continuing |
| **ERROR** | When an error in the application, possibly recoverable |
| **WARN** | When an event that might possible lead to an error |
| **INFO** | When an event for informational purposes |
| **DEBUG** | When a general debugging event required |
| **TRACE** | When a fine grained debug message, typically capturing the flow through the application |
| **ALL** | When all events should be logged |