**Simple Logging Facade for Java (SLF4J)**

The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks (e.g. java.util.logging, logback, log4j) allowing the end user to plug in the desired logging framework at *deployment* time

If we develop any framework we can not force to any one to use this logging framework so that we will use slf4j which is java specification and what ever logging framework developer will use automatically it will work with this logging framework.

If developer will not use any logging framework then he/she got warning that logging configuration.

Q= logger lavel ?

|  |  |
| --- | --- |
| 104down vote | This table might be helpful for you:  [Log Level](https://i.stack.imgur.com/7o9Kk.png) |

## **Five standard log4j levels**

### DEBUG Level

This log4j level helps developer to debug application. Level of message logged will be focused on providing support to a application developer.

### INFO Level

This log4j level gives the progress and chosen state information. This level will be generally useful for end user. This level is one level higher than DEBUG.

### WARN Level

This log4j level gives a warning about an unexpected event to the user. The messages coming out of this level may not halt the progress of the system.

### ERROR Level

This log4j level gives information about a serious error which needs to be addressed and may result in unstable state. This level is one level higher than WARN.

### FATAL Level

This log4j level is straightforward and you don’t get it quite often. Once you get this level and it indicates application death.

## **Two special log4j levels**

### ALL Level

This log4j level is used to turn on all levels of logging. Once this is configured and the levels are not considered.

### OFF Level

This log4j level is opposite to ALL level. It turns off all the logging.

## **New Trace Level added in Log4j**

### TRACE Level

This log4j level gives more detailed information than the DEBUG level and sits top of the hierarchy. This level is introduced from version 1.2.12 in log4j.

## **Custom log4j levels**

Log4j’s levels are mostly sufficient for all common applications. Rarely you may need a new Level apart from the levels provided by log4j. In that case you can extend [org.apache.log4j.Level](http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/Level.html) class and have your own custom level implementation.

## **Log4j Logger Output**

When a logger is created, generally you assign a level. The logger outputs all those messages equal to that level and **also all greater levels**than it. So the order for the standard log4j levels are:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Log4J Levels | TRACE Level | DEBUG Level | INFO Level | WARN Level | ERROR Level | FATAL Level |
| **TRACE Level** | Y | Y | Y | Y | Y | Y |
| **DEBUG Level** | N | Y | Y | Y | Y | Y |
| **INFO Level** | N | N | Y | Y | Y | Y |
| **WARN Level** | N | N | N | Y | Y | Y |
| **ERROR Level** | N | N | N | N | Y | Y |
| **FATAL Level** | N | N | N | N | N | Y |
| **ALL Level** | Y | Y | Y | Y | Y | Y |
| **OFF Level** | N | N | N | N | N | N |

### Log4j Level Inheritance Rule

There can be instances when you don’t assign a level to the logger. In such cases log4j handles it seamlessly based on the following level inheritance rule:  
The inherited level for a given logger C, is equal to the first non-null level in the logger hierarchy, starting at C and proceeding upwards in the hierarchy towards the root logger.

That means, if you have a package as com.foo.bar and you didn’t allocate a level, then it will inherit the level from com.foo package. Still in that package also if the level is not available, then it will inherit from the log4j root level. The log4j’s root logger is instantiated and available always. Log4j’s root logger by default has DEBUG level.