Now, consider a particular situation. Suppose, You write a application which receives json formatted message. If the message is valid, the message should be stored in a database. Now, suppose, according to requirement, the whole program’s log need to be printed in a file (including invalid messages related log) and the messages, if those are invalid, need to be printed in a separate log file. (Other than the log file which keeps program log)

Now, the main challenge is, how can log4j2 be configured in such a way that the second file will only contain invalid message related log?

Here, the problem is solved in a specific approach. A Separate log level is defined. Now, the standard log levels defined in log4j2 are:

|  |  |
| --- | --- |
| Standard Levels | Int levels |
| OFF | 0 |
| FATAL | 100 |
| ERROR | 200 |
| WARN | 300 |
| INFO | 400 |
| DEBUG | 500 |
| TRACE | 600 |
| ALL | INTEGER.MAX\_VALUE |

Now, the rule is, if INFO is enabled as log level, all messages with log level FATAL, ERROR and INFO will be printed.

So, if a custom log level is defined which has lower int level value than FATAL, and the second log’s file log level is set to that, it can solve the problem. However, there might be other ways to solve this problem.

Now, Again some variables are defined, within <Properties> opening and </Properties> closing tag.

**<Properties>**

**<Property name="LOG\_DIR1">ServerApplication.log**

**</Property>**

**<Property name="ARCHIVE1">logs/ServerApplication-%d{MM-dd-yyyy}-%i.log**

**</Property>**

**<Property name="PATTERN">%p %d{dd-MM-yyyy ,HH:mm:ss,SSS} [%t- %F-%L] %m%n</Property>**

**<Property name="LOG\_DIR2">ErroneousMessageRecieved.log**

**</Property>**

**<Property name="ARCHIVE2">logs/ErroneousMessageRecieved-%d{MM-dd-yyyy}-%i.log**

**</Property>**

**</Properties>**

Now, custom log levels are defined with in <CustomLevels> opening and </CustomLevels> closing tag:

**<CustomLevels>**

**<CustomLevel name="ERRORMESSASERECIEVED" intLevel="50" />**

**</CustomLevels>**

Now, two RollingFileAppenders are defined to print log in two separate files.

First,

**<RollingFile name="RollingFile1" fileName="${LOG\_DIR1}"**

**filePattern="${ARCHIVE1}" append="true">**

**<PatternLayout pattern="${PATTERN}" />**

**<Policies>**

**<SizeBasedTriggeringPolicy size="1024 MB" />**

**</Policies>**

**<DefaultRolloverStrategy max="7" />**

**</RollingFile>**

Now, first rolling file is defined like that.

Now, SizeBasedTriggeringPolicy is applied as you can see.

Second RollingFileAppender is defined as below:

**<RollingFile name="RollingFile2" fileName="${LOG\_DIR2}"**

**filePattern="${ARCHIVE2}" append="true">**

**<PatternLayout pattern="${PATTERN}" />**

**<Policies>**

**<SizeBasedTriggeringPolicy size="1024 MB" />**

**</Policies>**

**<DefaultRolloverStrategy max="7" />**

**</RollingFile>**

Use of append=true and some other issues are already discussed in previous examples.

A ConsoleAppender is also configured to write logs in console, too.

**<Console name="Console" target="SYSTEM\_OUT">**

**<PatternLayout pattern="${PATTERN}" />**

**</Console>**

Now, loggers are defined here.

**<Loggers>**

**<Root level="DEBUG">**

**<AppenderRef ref="Console" level="DEBUG"/>**

**<AppenderRef ref="RollingFile1" level="DEBUG"/>**

**<AppenderRef ref="RollingFile2" level="ERRORMESSASERECIEVED"/>**

**</Root>**

**</Loggers>**

Now, Though, the RollingFile2 has different log level, still it is mentioned within <Root> opening and </Root> closing tag.

rel

The reason behind this is, actually, a RootLogger instance can be created and using the RootLogger instance, the message can be printed to the RollingFile2.

Otherwise, we have to define another Logger and create an instance of that logger.