# What Is Log4j, Why Log4j Came into Picture

For logging log4j is the best even today, let us see little history behind this

While developing Java/J2EE applications, for debugging an application that is to know the status of a java application at its execution time, in general we use system.out.println statements in the application right…

But we have some disadvantages while using SOPL (system.out.println) statements.

* Generally, SOPL statements are printed on console, so there are temporary messages and whenever the console is closed then automatically the messages are removed from the console
* It is not possible to store the SOPL messages in a permanent place and these are single threaded model, means these will prints only one by one message on the console screen
* In order to overcome the problems of SOPL statements Log4j came into picture, with Log4j we can store the flow details of our Java/J2EE in a file or databases
* This is an Open-Source tool given by Apache, for only java projects, to record or write the status of an application at various places
* Working with log4j is nothing but working with classes & interfaces given in org.apache.log4j.\*
* Log4j is a common tool, used for small to large scale Java/J2EE projects
* In Log4j we use log statements rather SOPL statements in the code to know the status of a project while it is executing
* In real time, after a project is released and it is installed in a client location then we call the location as on-site right, when executing the program at on-site location, if we got any problems occurred then these problems must report to the off showered engineers, in this time we used to mail these Log files only so that they can check the problems easily

# What Are The Main Components Of Log4J

We have mainly 3 components to work with Log4j …

Logger

Appender

Layout

## Logger

## Logger is a class, in org.apache.log4j.\*

* We need to create Logger object one per java class
* This component enables Log4j in our java class
* Logger methods are used to generate log statements in a java class instead of sopls
* So in order to get an object of Logger class, we need to call a static factory method [factory method will gives an object as return type]
* We must create Logger object right after our class name, I will show you

## Getting Logger Object

static Logger log = Logger.getLogger(YourClassName.class.getName())

Note:  while creating a Logger object we need to pass either fully qualified class name or class object as a parameter, class means current class for which we are going to use Log4j.

public class Client {

   static Logger l = Logger.getLogger(Client.class.getName());

   public static void main(String[] args) {

     // Our logic will goes here

   }

}

Logger object has some methods, actually we used to print the status of our application by using these methods only

We have totally 5 methods in Logger class

* debug()
* info()
* warn()
* error()
* fatal()

As a programmer its our responsibility to know where we need to use what method, did you observe there ? method names are different right, but all are same

I mean, fatal is the highest error like some database down/closed

Remember:  Friends don’t confuse by seeing all these **5** methods all are same, for example if our application is about 100 lines and we have JDBC related code in some 45th line or some where there we used to write fatal() method.  All it could be is just for human identification purpose names are different, else these 5 methods will print one text message only ;)

You will get more clarity once you saw the first program on log4j.

## Appender

Appender job is to write the messages into the external file or database or smtp

* Logger classes generates some statements under different levels right, this Appender takes these log statements and stores in some files or database
* Appender is an interface

In log4j we have different Appender implementation classes

* FileAppender [ writing into a file]
* ConsoleAppender [ Writing into console]
* JDBCAppender [ For Databases]
* SMTPAppender [ Mails]
* SocketAppender [ For remote storage]
* SocketHubAppender
* SyslogAppendersends
* TelnetAppender

Again, in FileAppender we have 2 more

* RollingFileAppender
* DailyRollingFileAppender

For now, just remember, I will explain while executing the program

## Layout

This component specifies the format in which the log statements are written into the destination repository by the appender

We have different type of layout classes in log4j

* SimpleLayout
* PatternLayout
* HTMLLayout
* XMLLayout

Example

**import** org.apache.log4j.Appender;

**import** org.apache.log4j.FileAppender;

**import** org.apache.log4j.Layout;

**import** org.apache.log4j.Logger;

**import** org.apache.log4j.SimpleLayout;

**public** **class** ReadData {

**static** Logger *logger* = Logger.*getLogger*(ReadData.**class**);

**public** **static** **void** main(String[] args) **throws** IOException {

Layout layout = **new** SimpleLayout();

Appender appender = **new** FileAppender(layout, "abc.txt", **false**);

*logger*.addAppender(appender);

*logger*.info("I am INFO"); /\* and SO on \*/

Example for Production

**Pom.xml**

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.16.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.16.0</version>

</dependency>

**resources/ log4j2.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE xml>

<Configuration status=*"WARN"*>

<Properties>

<property name=*"basePath"*>C:/DocApmt</property>

<property name=*"logPattern"*>[%d{yyyy-MM-dd HH:mm:ss:SSS}] %c{0}.%M() @%L -%m%n</property>

</Properties>

<Appenders>

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

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

</Console>

<RollingFile name=*"fileApp"*

fileName=*"${basePath}/DocApmt.log"* immediateFlush=*"true"*

filePattern=*"${basePath}/$${date:yyyy-MM}/DocApmt-%d{MM-dd-yyyy}-%i.log.gz"*>

<PatternLayout>

<Pattern>${logPattern}</Pattern>

</PatternLayout>

<Policies>

<TimeBasedTriggeringPolicy />

<SizeBasedTriggeringPolicy size=*"10 MB"* />

</Policies>

<DefaultRolloverStrategy max=*"200"* />

</RollingFile>

</Appenders>

<Loggers>

<Logger name=*"com.docapmt"* level=*"info"* additivity=*"false"*>

<AppenderRef ref=*"fileApp"* />

<AppenderRef ref=*"consoleApp"* />

</Logger>

<Root level=*"info"*>

<AppenderRef ref=*"fileApp"* />

<AppenderRef ref=*"consoleApp"* />

</Root>

</Loggers>

</Configuration>