**What is Logging ?**

Logging is the process of writing the state of a program at various stages of its execution to some file or repository.By Logging,important statements related to your program execution can be sent to text file, console, or any other repository.  
By implementing logging in any project a reliable monitoring and debugging solution can be achieved.

**What is Log4j ?**

Log4j is a simple and flexible logging framework.Log4j is an open source logging API for Java.

Log4j handles inserting log statements in application code and managing them externally without touching application code,by using external configuration files.It will categorize log statements according to user-specified criteria and assigns different priority levels to these log statements.

Log4j lets users choose from several destinations for log statements, such as console, file, database, SMTP servers, GUI components etc.

By using log4j we can customized formats for log output and provides default formats.

## Installation of Log4j

<http://www.apache.org/dyn/closer.lua/logging/log4j/2.7/apache-log4j-2.7-bin.tar.gz>

**Components Of log4j**

Log4j is comprised of three main components.

* Logger
* Appender
* Layout

## 1) Logger :

Logger is the most essential component of the logging process. It is responsible for capturing the logging information.The component logger accepts log requests generated by log statements.It then sends their output to appropriate destination these destinations are termed as appenders.

The logger component is accessible through the Logger class of the log4j API.This class provides a static method Logger.getLogger(name).

getLogger(name) retrieves an existing logger object by the given name or , creates a new logger of given name if none exists.

The logger object is then used to set properties of logger component. It will invoke methods which generate log requests.These methods are debug(), info(), warn(), error(), fatal(), and log()

Each class in the Java application being logged can have an individual logger assigned to it or share a common logger with other classes.

Any number of loggers can be created for the application to suit specific logging needs

It is a common practice to create one logger for each class, with a name same as the fully-qualified class name.

**Getting Logger Object**

**1) Create a new logger:**

Logger logger = Logger.getLogger("MyLogger");

**2) Instantiate a static logger globally, based on the name of the class :**

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.

**Example**

public class Test {

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

public static void main(String[] args) {

// Our logic will goes here

}

}

To summarize priority level is given below.

trace < debug < info < warn < error < fatal

FATAL is largest whereas TRACE has lowest priority

# log4j Appenders

Appender job is to write the log statement into the external file or database or repository.

In log4j there are various appenders available such as..

## ConsoleAppender

appends log events to System.out or System.err using a layout specified by the user. The default target is System.out

## FileAppender

appends log events to a file.

## WriterAppender

appends log events to a Writer or an OutputStream depending on the user's choice.

## RollingFileAppender

extends FileAppender to backup the log files when they reach a certain size.

## DailyRollingFileAppender

extends FileAppender so that the underlying file is rolled over at a user chosen frequency.

## SMTPAppender

sends an e-mail when a specific logging event occurs, typically on errors or fatal errors

## TelnetAppender

specializes in writing to a read-only socket

## SyslogAppender

sends messages to a remote syslog domain.

# Layout

The Layout component defines the format in which the log statements are written to the log destination by appender

Layout is an abstract class in log4j API; it can be extended to create user-defined layouts

Some ready-made layouts are also available in log4j package; they are PatternLayout, SimpleLayout, DateLayout, HTMLLayout,and XMLLayout.

# A simple example of log4j

May 7, 2014 by [Mukesh Kumar](http://javawebtutor.com/aboutus.html) at 3:47 pm

In this article we will discuss how to generate log statement in a java program.

## Tools Used

* JDK 1.6
* Eclipse Indigo

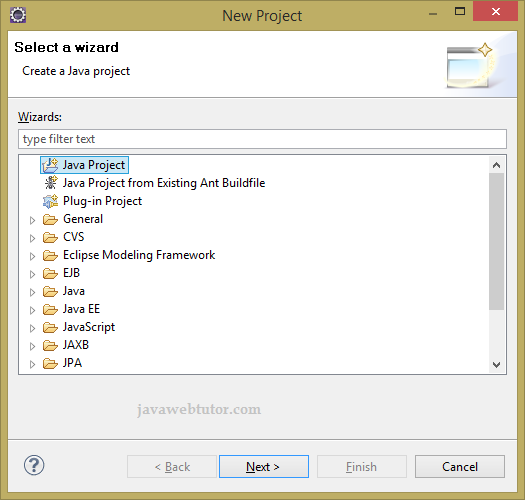
## JAR files required for this application

* log4j-1.2.15.jar

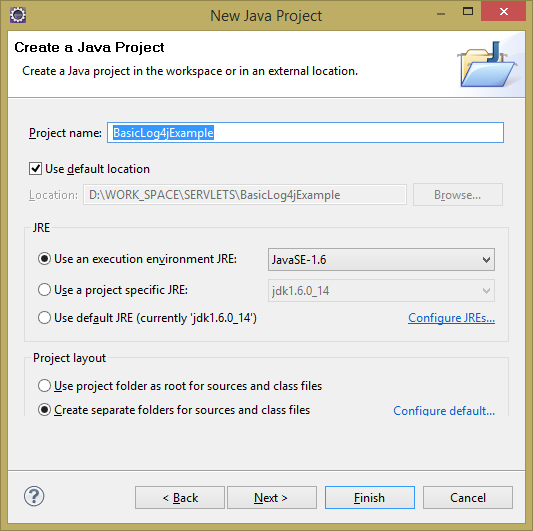
Let us start with our first log4j application.

## Step 1: Create Java project

Open Eclipse and goto File -> New -> Project and select Java Project in the New Project wizard screen and click Next.

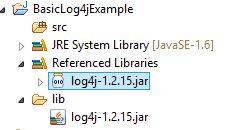


Provide the name of the project as BasicLog4jExample and click on Finish.



## Step 2: Add jar files to build path of the project

For simplicity we are going to create lib folder to accommodate necessary jar files required for this project. Creating new folder in eclipse is too easy,just right click on Project and select -> New -> folder and provide the name of the folder as lib. After creation of the folder copy the log4j-1.2.15.jar into this folder and add this jar file to the classpath of the project.



## Step 3: Create Java class

Create a package com.jwt.log4j and create a java class HelloWorld inside this package. Add following lines of code in this class.

**HelloWorld.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package com.jwt.log4j;    import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.Logger;    public class HelloWorld {        static final Logger logger = Logger.getLogger(HelloWorld.class);        public static void main(String[] args) {          BasicConfigurator.configure();          logger.debug("Sample debug message");          logger.info("Sample info message");          logger.warn("Sample warn message");          logger.error("Sample error message");          logger.fatal("Sample fatal message");      }  } |

## Description of the code:

* **Logger.getLogger():** Logger class is used for handling the majority of log operations and getLogger method is used for return a logger according to the value of the parameter. If the logger already exists, then the existing instance will be returned. If the logger is not already exist, then create a new instance.

* **Logger.debug():** This method is used to check that the specified category is DEBUG enabled or not, if yes then it converts the massage passed as a string argument to a string by using appropriate object renderer of class ObjectRenderer.

* **Logger.info():** This method is used to check that the specified category is INFO enabled or not, if yes then it converts the massage passed as a string argument to a string by using appropriate object renderer of class ObjectRenderer.

* **Logger.warn():** This method is used to check that the specified category is WARN enabled or not, if yes then it converts the massage passed as a string argument to a string by using appropriate object renderer of class ObjectRenderer.

* **Logger.error():** This method is used to check that the specified category is ERROR enabled or not, if yes then it converts the massage passed as a string argument to a string by using appropriate object renderer of class ObjectRenderer.

* **Logger.fatal():** This method is used to check that the specified category is FATAL enabled or not, if yes then it converts the massage passed as a string argument to a string by using appropriate object renderer of class ObjectRenderer.

* **BasicConfigurator. configure()** method will log all the messages on the console.

## Output

0 [main] DEBUG com.jwt.log4j.HelloWorld - Sample debug message

1 [main] INFO com.jwt.log4j.HelloWorld - Sample info message

2 [main] WARN com.jwt.log4j.HelloWorld - Sample warn message

2 [main] ERROR com.jwt.log4j.HelloWorld - Sample error message

2 [main] FATAL com.jwt.log4j.HelloWorld - Sample fatal message

# Log4j Configuration Using Properties File

## Log4j Configuration Using Properties File

In this tutorial you will learn how to configure Log4j using properties file.There are mainly two ways to configure log4j externally.We can easily configure the log4j by using a properties file or xml file. The main benefit of this way is once the log statements are in place you can easily control them using the external configuration file without modifying the source code. Let's see how you can configure the log4j configuration by using the properties file.

## Steps to create this example

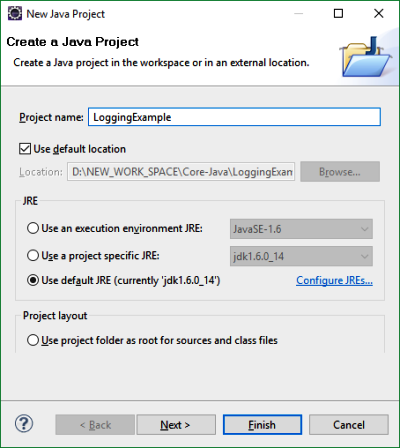
We will develop a Java project using eclipse IDE for this example.Follow the steps mentioned below to implement this example.

## Step 1. Download log4j jar

First download the latest version of log4j from [*here*](https://logging.apache.org/log4j/1.2/download.html) and unzip it in your local drive.

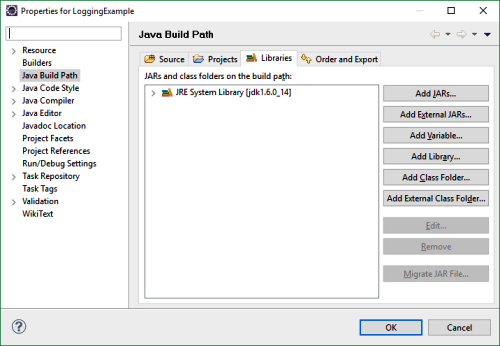
## Step 2 . Create Java Project

Open Eclipse IDE and create a Java Project and name it. In this example and name it as *"LoggingExample"*.

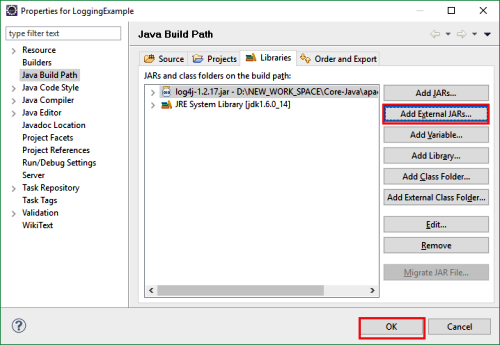


## Step 3 . Add log4j jar file to classpath of the project

Next step is to add the log4j jar which you have downloaded to the *"LoggingExample"* application by right click on the project in Package Explorer -> Build Path -> Configure Build Path



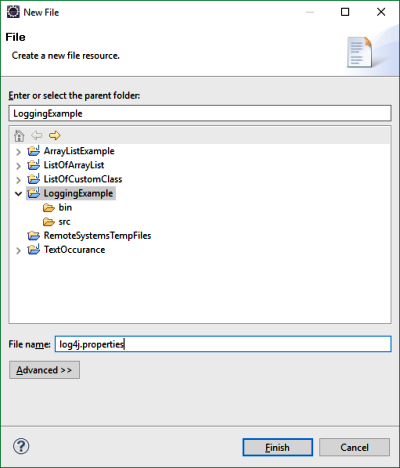
In the Java Build Path dialogue, go to Libraries tab and Click on Add External JARs button and add the log4j-xxx.jar file by browsing your local drive where log4j jar file is extracted and Click OK.



Now the log4j jar file is available to be used in your application.

## Step 4 . Create log4j.properties file

Now create a file named as "log4j.properties" in src package. This file will hold all the configuration settings for log4j for this application. You can create this file by right click on the 'src' -> New -> File ->and then provide the name of the file as log4j.properties and click Finish button.



Now Copy the below line of code to the log4j.properties file you just created.

log4j.rootLogger=DEBUG, CA

log4j.appender.CA=org.apache.log4j.ConsoleAppender

log4j.appender.CA.layout=org.apache.log4j.PatternLayout

log4j.appender.CA.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n

In any application the first step is to define the log level.The rootLogger is the one that resides on the top of the logger hierarchy. Here we set its level to DEBUG and added the console appender (CA) to it. The console appender can have arbitrary name, here its name is CA.

Log4j logs messages at six different levels. For example, if you have a program that generates lot of warning messages then you can ignore them by setting the log level to ERROR to avoid the log file getting more friendly. The log levels and the priority is as follows,

TRACE < DEBUG < INFO < WARN < ERROR < FATAL

If you specify the log level as WARN, then the INFO, DEBUG and TRACE log level messages will be omitted while the WARN, ERROR and FATAL log level messages will be logged. In our example we have set the log level as DEBUG which means TRACE level logs will not be logged.

log4j.rootLogger=DEBUG, CA

Next comes the appender settings. I have used Console Appender that means application log information will be displayed on the console.You can also use other appenders, like FileAppender, JDBCAppender,SocketAppender,SysLogAppender etc. according to your requirement to route the logging information to appropriate destinations.

log4j.appender.CA=org.apache.log4j.ConsoleAppender

Each appender is associated with layout settings that specifies the format of information that is being logged. We have used PatternLayout for appenders.

log4j.appender.CA.layout=org.apache.log4j.PatternLayout

log4j.appender.CA.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n

In the above code

%5p - Priority of the logging event

%t - Name of the thread that initiated the logging event

%F- File name where the logging issue was requested

%L - line number that caused the logging message to be generated

You can use other layouts such as HTMLLayout, DateLayout, XMLLayout etc.

## ****Log4j Components****

Log4j has three main components, which are the following:

1. Logger
2. Appender
3. Layout

### ****Logger****

Logger is a class in the org.apache.log4j.\* package. We have to initialize one Logger object for each Java class. We use Logger’s methods to generate log statements. Log4j provides the factory method to get Logger objects.

Syntax to get Logger objects:

static Logger logger = Logger.getLogger(CurrentClass.class.getName()).

**Note**: CurrentClass is a Java class name for which we are getting logger object.

### ****Example****

public class Student{

private static final Logger LOGGER = Logger.getLogger(Student.class);

public void getStudentRecord() {

}

}

The Logger class has some methods that are used to print application status.

We have five methods in the Logger class

1. info()
2. debug()
3. warn()
4. fatal()
5. error()

How and when to use these methods depends on us. Here, the method names are different, but the process is the same for all of them: all will print a message only.

### ****Levels****

Level is a class in the org.apache.log4j.\* package. We can also make a custom level by extending the Level class. Each level has a different priority order, like this:

debug < info < warn < error < fatal

It means fatal is the highest priority error, like if/when the database is closed.

### ****Appender****

Appender is used to write messages into a file or DB or SMTP.

Log4j has different types of appenders:

1. SyslogAppendersends
2. SMTPAppender
3. JDBCAppender
4. FileAppender
5. SocketHubAppender
6. SocketAppender
7. TelnetAppender
8. ConsoleAppender

### ****Layout****

This is used to define the formatting in which logs will print in a repository.

We have different types of layouts:

1. PatternLayout
2. SimpleLayout
3. XMLLayout
4. HTMLLayout

## ****Log4j: Configuration****

### ****log4j.properties****

# Root logger option

log4j.rootLogger=INFO, file, stdout

# configuration to print into file

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=D:\\log\\logging.log

log4j.appender.file.MaxFileSize=12MB

log4j.appender.file.MaxBackupIndex=10

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

# configuration to print on console

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

**Description of log4j.properties file :**

* log4j.appender.**file**=org.apache.log4j.**RollingFileAppender**
* log4j.appender.**stdout**=org.apache.log4j.**ConsoleAppender**

These will define appender types: That means they will specify where we want to store application logs. RollingFileAppender will print all logs in a file, and ConsoleAppender will print all logs in the console.

* log4j.appender.file.File=**D:\\log\\logging.log**

That specifies the log file location.

* log4j.appender.**file.layout**=org.apache.log4j.PatternLayout
* log4j.appender**.file.layout**.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

These specify the pattern in which logs will print to the log file.

Example:

import org.apache.log4j.Logger;

public class Student {

static Logger logger = Logger.getLogger(Student.class);

public static void main(String[] args) {

logger.debug("This is debug message");

logger.info("This is info message");

logger.warn("This is warn message");

logger.fatal("This is fatal message");

logger.error("This is error message");

System.out.println("Logic executed successfully....");

}

}

logging.log(log file):

2018-05-02 16:01:45 INFO Student:12 - This is info message

2018-05-02 16:01:45 WARN Student:13 - This is warn message

2018-05-02 16:01:45 FATAL Student:14 - This is fatal message

2018-05-02 16:01:45 ERROR Student:15 - This is error message

It will not print debug level error logs because we defined our root logger as INFO-level in our log4j.properties file. Error messages with a priority greater than INFO will print.

Console logs:

16:01:45,511 &nbsp;INFO Student:12 - This is info message

16:01:45,517 &nbsp;WARN Student:13 - This is warn message

16:01:45,517 FATAL Student:14 - This is fatal message

16:01:45,518 ERROR Student:15 - This is error message

program executed successfully....

# log4j.properties example – Log4j properties file example

By Lokesh Gupta | Filed Under: [Log4j](https://howtodoinjava.com/log4j/)

[**Log4j**](https://howtodoinjava.com/log4j/) is a simple and flexible logging framework. Logging equips the developer with detailed context for application failures. With log4j it is possible to enable logging at runtime without modifying the application binary. The log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost.

This this **log4j properties file tutorial**, I am showing the example code for configuring log4j using **log4j.properties** file.

## 1. Log4j maven dependencies

[Create a maven java project](https://howtodoinjava.com/maven/create-java-project-maven/) and update log4j maven dependencies.

|  |
| --- |
| pom.xml |
| <dependency>      <groupId>log4j</groupId>      <artifactId>log4j</artifactId>      <version>1.2.17</version>  </dependency> |

## 2. log4j.properties file

This is the main properties file having all runtime configuration used by log4j. This file will have log4j appenders information, log level information and output file names for file appenders.

|  |
| --- |
| log4j.properties |
| log4j.rootLogger=DEBUG, consoleAppender, fileAppender    log4j.appender.consoleAppender=org.apache.log4j.ConsoleAppender  log4j.appender.consoleAppender.layout=org.apache.log4j.PatternLayout  log4j.appender.consoleAppender.layout.ConversionPattern=[%t] %-5p %c %x - %m%n    log4j.appender.fileAppender=org.apache.log4j.RollingFileAppender  log4j.appender.fileAppender.layout=org.apache.log4j.PatternLayout  log4j.appender.fileAppender.layout.ConversionPattern=[%t] %-5p %c %x - %m%n  log4j.appender.fileAppender.File=demoApplication.log |

## 3. log4j.properties example

|  |
| --- |
| Log4jPropertiesConfigurationExample.java |
| package com.howtodoinjava;    import org.apache.log4j.Logger;  import org.apache.log4j.PropertyConfigurator;    public class Log4jPropertiesConfigurationExample  {      static Logger logger = Logger.getLogger(Log4jPropertiesConfigurationExample.class);      public static void main(String[] args)      {          //PropertiesConfigurator is used to configure logger from properties file          PropertyConfigurator.configure("log4j.properties");            //Log in console in and log file          logger.debug("Log4j appender configuration is successful !!");      }  } |

Output in console and **demoApplication.log** in project root folder:

|  |
| --- |
| Console |
| [main] DEBUG com.howtodoinjava.Log4jPropertiesConfigurationExample&nbsp; - Log4j appender configuration is successful !! |

Now let’s see some **log4j.properties examples** to output log messages to specific location.

## 4. Log4j ConsoleAppender – Logging to console

Java program to output logs to console.

|  |
| --- |
| # Root logger  log4j.rootLogger=INFO, console    log4j.appender.console=org.apache.log4j.ConsoleAppender  log4j.appender.console.Target=System.out  log4j.appender.console.layout=org.apache.log4j.PatternLayout  log4j.appender.consoleAppender.layout.ConversionPattern=[%t] %-5p %c %x - %m%n |

## Log4j RollingFileAppender – Logging to file

Java program to output logs to file.

|  |
| --- |
| # Root logger  log4j.rootLogger=INFO, file    # Direct log messages to a log file  log4j.appender.file=org.apache.log4j.RollingFileAppender    log4j.appender.file.File=C:\\temp\info.log  log4j.appender.file.MaxFileSize=10MB  log4j.appender.file.MaxBackupIndex=10  log4j.appender.file.layout=org.apache.log4j.PatternLayout  log4j.appender.file.layout.ConversionPattern=[%t] %-5p %c %x - %m%n |