* log4j has three main components:

**loggers**: Responsible for capturing logging information.

**appenders**: Responsible for publishing logging information to various preferred destinations.

**layouts**: Responsible for formatting logging information in different styles.

log4j Features

* It is thread-safe.
* It is optimized for speed.
* It is based on a named logger hierarchy.
* It supports multiple output appenders per logger.
* It supports internationalization.
* It is not restricted to a predefined set of facilities.
* Logging behavior can be set at runtime using a configuration file.
* It is designed to handle Java Exceptions from the start.
* It uses multiple levels, namely ALL, TRACE, DEBUG, INFO, WARN, ERROR and FATAL.
* The format of the log output can be easily changed by extending the *Layout* class.
* The target of the log output as well as the writing strategy can be altered by implementations of the Appender interface.
* It is fail-stop. However, although it certainly strives to ensure delivery, log4j does not guarantee that each log statement will be delivered to its destination.

### **Logger Object**

The top-level layer is the Logger which provides the Logger object. The Logger object is responsible for capturing logging information and they are stored in a namespace hierarchy.

### **Layout Object**

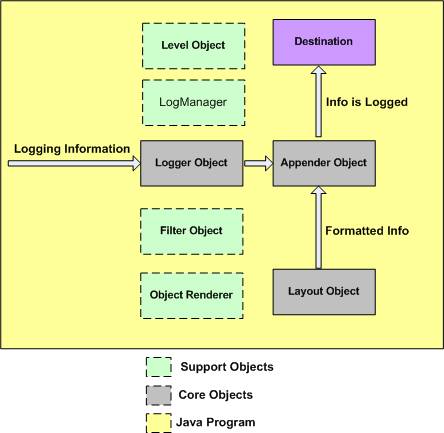
The layout layer provides objects which are used to format logging information in different styles. It provides support to appender objects before publishing logging information.

Layout objects play an important role in publishing logging information in a way that is human-readable and reusable.

### **Appender Object**

This is a lower-level layer which provides Appender objects. The Appender object is responsible for publishing logging information to various preferred destinations such as a database, file, console, UNIX Syslog, etc.

The following virtual diagram shows the components of a log4J framework:



**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 useLog4j.

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()
* fetal()

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

## Priority Order

debug < info < warn < error < fatal

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

**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

**Layout**

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

We have different type of layout classes in log4j

* SimpleLayout
* PatternLayout
* HTMLLayout
* XMLLayout

let us see one simple program in Log4j

For working with log4j, we must set log4j.jar in our class path

**Files Required**

* Client.java
* my.txt   [ We will let the appender to write into this file ]

## Client.java

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 Client {

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

  public static void main(String[] args) {

  Layout l1 = new SimpleLayout();

  Appender a;

  //Appender a = new ConsoleAppender(l1);

  try

  {

  a = new FileAppender(l1,"my.txt", false);

  // 3rd parameter is true by default

  // true = Appends the data into my.txt

  // false = delete previous data and re-write

  l.addAppender(a);

  }

  catch(Exception e) {}

  l.fatal("This is the error message..");

  System.out.println("Your logic executed successfully....");

  }

}

Once we run this client program, my.txt will contains….

## my.txt

FATAL – This is the error message..

**Explanation**

* First step is to create one Logger class object [ see line number 9 ]
* Second step is to create Layout object  [ see line number 13 ]
* Once Layout is ready, our next step is to create Appender [ see line number 18 ]
* In appender i have passed 3 parameters like.. first parameter is object of layoutbecause, appender will write the error messages based on the layout we selected, then 2nd parameter is file name with extension [ in this file only appender will writes the messages ], then 3rd parameter is by default true, means appender will appends the error messages, if we give false then appender will clears the previous data in my.txt file and write newly

By default the file name would be log4j.properties. This properties file stores data in the form of key, values pairs, in this file keys are fixed but values are our own.  We can include all the log4j related properties into this file.

## log4j.properties

log4j.rootLogger=DEBUG,CONSOLE,LOGFILE

log4j.appender.CONSOLE=  
log4j.appender.CONSOLE.layout=  
log4j.appender.CONSOLE.layout.ConversionPattern=

log4j.appender.LOGFILE=  
log4j.appender.LOGFILE.File=  
log4j.appender.LOGFILE.MaxFileSize=  
log4j.appender.LOGFILE.layout=  
log4j.appender.LOGFILE.layout.ConversionPattern=

Let us see how to use log4j.properties file

**Files Required**

* Client.java
* log4j.properties
* my.txt [ We will let the appender to write into this file ]

Client.java

|  |  |
| --- | --- |
| import org.apache.log4j.Logger;    public class Client {      static Logger l = Logger.getLogger(Client.class.getName());      public static void main(String[] args) {          l.debug("This is debug message");        l.info("This is info message");        l.warn("This is warn message");        l.fatal("This is fatal message");        l.error("This is error message");          System.out.println("Your logic executed successfully....");      }  } |  |

Once we run this client program, my.txt will contains….

## log4j.properties

log4j.rootLogger = DEBUG,abc  
log4j.appender.abc = org.apache.log4j.FileAppender  
log4j.appender.abc.file = my.txt  
log4j.appender.abc.layout = org.apache.log4j.SimpleLayout

## my.txt

DEBUG – This is debug message  
INFO – This is info message  
WARN – This is warn message  
FATAL – This is fatal message  
ERROR – This is error message

## Execution Flow

* Run Client.java
* Log4j environment created, at line number 5
* As our default properties file name is **log4j**.properties, we no need to import properties file explicitly into Client.java, by default our java class will verify for the properties file named log4j.properties.  If we give the name other than log4j to the properties we have to import manually into our java class [ will see this later, like how to manually ]
* So once Logger object created at line number 5, our class will be able to know about the content in log4j.properties
* In log4j.properties the content always will be in key,value pairs only

## Explanation

* If we use .properties file, we no need to import any related classes into our java class
* log4j.rootLogger = DEBUG,abc  — > Here DEBUG means we are specifying the level from where log4j methods execution must start,  see my.txt file it showing all messages right.  But if we wrote log4j.rootLogger = WARN,abc then it will prints the messages in l.warn(), l.error(), l.fatal() and ignores l.debug(), l.info()
* I have used FileAppender as my Appender, so if we want to change my appender toConsoleAppender, i will open log4j.properties file and do the modifications,  so no need to touch our java classes, this is the main advantage of .properties file

So just change layout into HTMLLayout and check the output

Let us see how to use FileAppender and ConsoleAppender at a time.

## Files Required

* Client.java
* log4j.properties
* my.txt [ We will let the appender to write into this file ]

import org.apache.log4j.Logger;

public class Client {

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

  public static void main(String[] args) {

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

      l.info("This is info message");

      l.warn("This is warn message");

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

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

      System.out.println("Your logic executed successfully....");

  }

}

## log4j.properties

log4j.rootLogger=DEBUG,CONSOLE,LOGFILE

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender  
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout  
log4j.appender.CONSOLE.layout.ConversionPattern=%-4r [%t] %-5p %c %x – %m%n

log4j.appender.LOGFILE=org.apache.log4j.RollingFileAppender  
log4j.appender.LOGFILE.File=my.txt  
log4j.appender.LOGFILE.MaxFileSize=1kb  
log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout  
log4j.appender.LOGFILE.layout.ConversionPattern=[%t] %-5p %c %d{dd/MM/yyyy HH:mm:ss} – %m%n

**my.txt**

[main] DEBUG Client 27/02/2012 21:39:15 – This is debug message  
[main] INFO  Client 27/02/2012 21:39:15 – This is info message  
[main] WARN  Client 27/02/2012 21:39:15 – This is warn message  
[main] FATAL Client 27/02/2012 21:39:15 – This is fatal message  
[main] ERROR Client 27/02/2012 21:39:15 – This is error message

**Note**

1. For standalone Java app, make sure the log4j.properties file is under theproject/classes directory
2. For Java web applications, make sure the log4j.properties file is under the WEB-INF/classes directory

* to configure log4j. The simple way to do that is using *BasicConfigurator. configure()* method. This will log all the messages on the console.

package com.vaannila.helloworld;

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("Hello World!");

}

}

Log4j will be usually configured using a properties file or xml file externally. So once the log statements are in place you can easily control them using the external configuration file without modifying the source code.

Three main components you need to configure to obtain the same result are the logger, appender and layout. The logger object is the one that is used to log messages, appender is the one that specifies the output destination like console or a file and layout is the one that specify the format in which the log messages should be logged.

When you configure using the BasicConfigurator.configure() method by default it uses the ConsoleAppender and PatternLayoutfor all the loggers.

The following configuration creates the same result as the *BasicConfigurator.configure()* method.

[view source](http://www.dzone.com/tutorials/java/log4j/sample-log4j-properties-file-configuration-1.html#viewSource)

[print](http://www.dzone.com/tutorials/java/log4j/sample-log4j-properties-file-configuration-1.html#printSource)[?](http://www.dzone.com/tutorials/java/log4j/sample-log4j-properties-file-configuration-1.html#about)

log4j.rootLogger=DEBUG, CA

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

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

log4j.appender.CA.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

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.

* set its layout to PatternLayout. The PatternLayout uses the ConversionPattern to format the message.

Once the appender is created and its layout is set you need to specify which loggers can use this appender. If you set this appender to the rootLogger then all the loggers will log messge to this appender. Since the rootLogger is on top of the hierarchy all the loggers will inherit its logger level and its appenders.

package com.vaannila.helloworld;

import org.apache.log4j.Logger;

import org.apache.log4j.PropertyConfigurator;

public class HelloWorld {

static final Logger logger = Logger.getLogger(HelloWorld.class);

public static void main(String[] args) {

PropertyConfigurator.configure("log4j.properties");

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");

}

}

You need to use the PropertyConfigurator.configure() method to configure log4j using a properties file. Log4j should be configured only once for the entire application.

The log4j levels follow the following order.

* *DEBUG*
* *INFO*
* *WARN*
* *ERROR*
* *FATAL*

If you set the rootLogger level to *WARN* then only the *WARN*, *ERROR* and *FATAL* level log messages will be displayed and the rest will be dropped.

# **Log4j Multiple Appender Example**

log4j.rootLogger=DEBUG, CA, FA

#Console Appender

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

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

log4j.appender.CA.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

#File Appender

log4j.appender.FA=org.apache.log4j.FileAppender

og4j.appender.FA.File=sample.log

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

log4j.appender.FA.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

# Set the logger level of File Appender to WARN

log4j.appender.FA.Threshold = WARN

Here we create two appenders *ConsoleAppender* and *FileAppender*. You need to set the *File* attribute of the *FileAppender* to the log file name, here it is *sample.log*. Add the *FileAppender* (*FA*) and the *ConsoleAppender* (*CA*) to the *rootLogger*.

package com.vaannila.helloworld;

import org.apache.log4j.Logger;

import org.apache.log4j.PropertyConfigurator;

public class HelloWorld {

static final Logger logger = Logger.getLogger(HelloWorld.class);

public static void main(String[] args) {

PropertyConfigurator.configure("log4j.properties");

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");

}

}

FileAppender Configuration

FileAppender has the following configurable parameters:

|  |  |
| --- | --- |
| **Property** | **Description** |
| immediateFlush | This flag is by default set to true, which means the output stream to the file being flushed with each append operation. |
| encoding | It is possible to use any character-encoding. By default, it is the platform-specific encoding scheme. |
| threshold | The threshold level for this appender. |
| Filename | The name of the log file. |
| fileAppend | This is by default set to true, which means the logging information being appended to the end of the same file. |
| bufferedIO | This flag indicates whether we need buffered writing enabled. By default, it is set to false. |
| bufferSize | If buffered I/O is enabled, it indicates the buffer size. By default, it is set to 8kb. |

Following is a sample configuration file **log4j.properties** for FileAppender −

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

# Set the name of the file

log4j.appender.FILE.File=${log}/log.out

# Set the immediate flush to true (default)

log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode

log4j.appender.FILE.Threshold=debug

# Set the append to false, overwrite

log4j.appender.FILE.Append=false

# Define the layout for file appender

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

log4j.appender.FILE.layout.conversionPattern=%m%n

**Log4j XML Configuration**

We can also use XML file to configure log4j.

The following code shows the *log4j.properties* file we used in the previous example.

log4j.rootLogger=DEBUG, CA

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

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

log4j.appender.CA.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

The following code shows the corresponding *log4j.xml* file.

[print](http://www.dzone.com/tutorials/java/log4j/log4j-xml-configuration.html#printSource)[?](http://www.dzone.com/tutorials/java/log4j/log4j-xml-configuration.html#about)

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

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j='<http://jakarta.apache.org/log4j/>'>

<appender name="CA" class="org.apache.log4j.ConsoleAppender">

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%-4r [%t] %-5p %c %x - %m%n" />

</layout>

</appender>

<root>

<level value="debug" />

<appender-ref ref="CA" />

</root>

</log4j:configuration>

To configure log4j using xml file we use *DOMConfigurator.configure()* method.

package com.vaannila.helloworld;

import org.apache.log4j.Logger;

import org.apache.log4j.xml.DOMConfigurator;

public class HelloWorld {

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

public static void main(String[] args) {

DOMConfigurator.configure("log4j.xml");

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"); }}

|  |
| --- |
| log4j.rootLogger=DEBUG, CA, FA  #Console Appender  log4j.appender.CA=org.apache.log4j.ConsoleAppender  log4j.appender.CA.layout=org.apache.log4j.PatternLayout  log4j.appender.CA.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n  #File Appender  log4j.appender.FA=org.apache.log4j.FileAppender  log4j.appender.FA.File=sample.log  log4j.appender.FA.layout=org.apache.log4j.PatternLayout  log4j.appender.FA.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n  # Set the logger level of File Appender to WARN  log4j.appender.FA.Threshold = WARN |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">  <log4j:configuration xmlns:log4j='<http://jakarta.apache.org/log4j/>'>  <appender name="CA" class="org.apache.log4j.ConsoleAppender">  <layout class="org.apache.log4j.PatternLayout">  <param name="ConversionPattern" value="%-4r [%t] %-5p %c %x - %m%n" />  </layout>  </appender>  <appender name="FA" class="org.apache.log4j.FileAppender<param name="File" value="sample.log"/>  <param name="Threshold" value="WARN"/>  <layout class="org.apache.log4j.PatternLayout">  <param name="ConversionPattern" value="%-4r [%t] %-5p %c %x - %m%n" />  </layout>  </appender>  <root>  <level value="DEBUG" />  <appender-ref ref="CA" />  <appender-ref ref="FA" />  </root>  </log4j:configuration> |
|  |
|  |

**Log4j with Servlet**

**web.xml**

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

**<web-app** id="log4j-webapp-demo" version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"**>**

**<servlet>**

**<servlet-name>**Log4JTestServlet**</servlet-name>**

**<servlet-class>**test.Log4JTestServlet**</servlet-class>**

**</servlet>**

**<servlet>**

**<servlet-name>**Log4JInitServlet**</servlet-name>**

**<servlet-class>**test.Log4JInitServlet**</servlet-class>**

**<init-param>**

**<param-name>**log4j-properties-location**</param-name>**

**<param-value>**WEB-INF/log4j.properties**</param-value>**

**</init-param>**

**<load-on-startup>**1**</load-on-startup>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**Log4JTestServlet**</servlet-name>**

**<url-pattern>**/test**</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**Log4JInitServlet.java**

**package** test;

**import** java.io.File;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.ServletContext;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** org.apache.log4j.BasicConfigurator;

**import** org.apache.log4j.PropertyConfigurator;

**public** **class** Log4JInitServlet **extends** HttpServlet {

**private** **static** **final** **long** serialVersionUID = 1L;

**public** **void** init(ServletConfig config) **throws** ServletException {

System.out.println(**"Log4JInitServlet is initializing log4j"**);

String log4jLocation = config.getInitParameter(**"log4j-properties-location"**);

ServletContext sc = config.getServletContext();

**if** (log4jLocation == null) {

System.err.println(**"\*\*\* No log4j-properties-location init param, so initializing log4j with BasicConfigurator"**);

BasicConfigurator.configure();

} **else** {

String webAppPath = sc.getRealPath(**"/"**);

String log4jProp = webAppPath + log4jLocation;

File yoMamaYesThisSaysYoMama = **new** File(log4jProp);

**if** (yoMamaYesThisSaysYoMama.exists()) {

System.out.println(**"Initializing log4j with: "** + log4jProp);

PropertyConfigurator.configure(log4jProp);

} **else** {

System.err.println(**"\*\*\* "** + log4jProp + **" file not found, so initializing log4j with BasicConfigurator"**);

BasicConfigurator.configure();

}

}

**super**.init(config);

}

}

### [log4j.properties](http://www.avajava.com/tutorials/logging/how-do-i-initialize-log4j-in-a-web-application/log4j.properties.txt)

# This sets the global logging level and specifies the appenders

log4j.rootLogger=INFO, myConsoleAppender

# settings for the console appender

log4j.appender.myConsoleAppender=org.apache.log4j.ConsoleAppender

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

log4j.appender.myConsoleAppender.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

**Log4JTestServlet.java**

**package** test;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

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

**public** **class** Log4JTestServlet **extends** javax.servlet.http.HttpServlet **implements** javax.servlet.Servlet {

**private** **static** **final** **long** serialVersionUID = 1L;

**static** Logger log = Logger.getLogger(Log4JTestServlet.**class**);

**public** Log4JTestServlet() {

**super**();

}

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType(**"text/html"**);

PrintWriter out = response.getWriter();

out.println(**"Howdy<br/>"**);

log.debug(**"debug message"**);

log.info(**"info message"**);

log.warn(**"warn message"**);

log.error(**"error message"**);

log.fatal(**"fatal message"**);

}

}