



Version 0.1

GRASSFIELD.org
MARCH 2008

SOME RIGHTS RESERVED

This work is licensed under the Creative Commons Attribution 2.5 India License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/2.5/in> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

VERSION HISTORY

DATE	VERSION	DESCRIPTION	AUTHOR
03/30/08	0.1	Initial Draft	Pandian R

1 About this document

This document can be used as a nutshell guide to implement log4j in a java application.

The audiences are intended to be developers/senior developers. Knowledge of core java is mandatory for this document.

I may be poor in documenting. Please keep me informed about the grammatical or typo errors.

2 About Log4j

Log4j is a debugging mechanism used in java applications. This is used to insert logs in a java application, that will be useful for debugging purposes. There are many ways to do the logging, either they can be written in a stream, or syslog daemon etc. The distinctive feature of log4j is the hierarchical logging priorities. Logs of different severity can be inserted, and logs of which severity to be logged can also be configured. Using log4j we can come out of many System.out.println which is really unprofessional way of inserting application traces

Table of Contents

1	About this document.....	4
2	About Log4j.....	4
1	Log4j Features.....	6
1.1	Logger.....	6
1.2	Appenders.....	7
1.2.1	Common Appenders.....	7
	FileAppender.....	7
	RollingFileAppender.....	7
	DailyRollingFileAppender.....	7
	ConsoleAppender.....	7
1.2.2	Network appenders.....	8
	Socket Appender.....	8
	SocketHubAppender	8
	JMSAppender	8
	NTEventLogAppender	8
1.2.3	Special Appenders.....	8
	AsyncAppender.....	8
	ExternallyRolledFileAppender.....	8
	JDBCAppender.....	8
	LF5Appender.....	8
	SMTPAppender.....	8
	SyslogAppender.....	8
	TelnetAppender.....	8
	WriterAppender.....	8
1.3	Layout.....	9
1.3.1	simple layout.....	9
1.3.2	PatternLayout	9
	Substitute symbol.....	10
1.3.3	HTML layout.....	10
1.3.4	XMLLayout.....	10
1.4	Levels.....	10
2	Log4j installation.....	11
3	Pros & Cons of Log4j	12
3.1	Pros:.....	12
3.2	Cons:.....	12

1 Log4j Features

Basic features of any logging libraries are control over the logging facility, managing the output destinations and managing the output formats. Considering these in mind Log4j has three main features

1. Logger
2. Appender
3. Layout

1.1 Logger

Logger are named entities. A logger is said to be parent of another child logger. For example, org.parent is the parent logger for org.parent.child. Similarly, ancestor.parent is the parent for ancestor.parent.child

There is a root logger, that resides at the top of all loggers. It exists always, and it cannot be retrieved by name. Invoking `Logger.getRootLogger()` method invokes it. All other loggers are accessed by `Logger.getLogger()` method.

The following statement fetches the root logger.

```
Logger logger = Logger.getRootLogger();
```

The following statement creates a new logger by a named text.

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

Globally, One instantiates the logger with the name of the class

```
static Logger logger = Logger.getLogger(test.class);
```

There are 5 levels comes with log4j by default.

static Level DEBUG - The DEBUG Level designates fine-grained informational events that are most useful to debug an application.

static Level INFO - The INFO level designates informational messages that highlight the progress of the application at coarse-grained level.

static Level WARN - The WARN level designates potentially harmful situations.

static Level ERROR - The ERROR level designates error events that might still allow the application to continue running.

static Level FATAL - The FATAL level designates very severe error events that will presumably lead the application to abort.

Customized levels are also possible.

In addition, there are two special levels of logging available

static Level ALL -The ALL Level has the lowest possible rank and is intended to turn on all logging.

static Level OFF - The OFF Level has the highest possible rank and is intended to turn off logging.

So, a level can be associated to a logger like

```
logger.setLevel((Level)Level.DEBUG);
```

All the 7 levels can be applied in the above statement.

1.2 Appenders

To enable or disable the log is part of the our target. The destination of logging is also equally important. Log4j allows the logging requests to be printed in multiple locations. In log4j terms, the output is called as an *appender*. Currently appenders exist for console, files, GUI components, remote socket servers, JMS, NT Event logs and remote Unix syslog daemons. More than one appender can be attached to a logger.

The appenders available are

1. Common Appenders
2. Network Appenders
3. Special Appenders
4. Third-party Appenders

1.2.1 Common Appenders

FileAppender

It appends log events to a file.

A file appender can be created like this.

```
FileAppender appender = null;
try {
    appender = new FileAppender(new PatternLayout(), "filename");
} catch (Exception e) {}
```

Another useful constructor is:

```
FileAppender(Layout layout, String filename, boolean append)
```

Instantiate a FileAppender and open the file designated by filename.

So that one may choose whether or not to append the file specified or not. If this is not specified, the default is to append.

RollingFileAppender

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

DailyRollingFileAppender

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

ConsoleAppender

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

A ConsoleAppender can be created like this:

```
ConsoleAppender appender = new ConsoleAppender(new PatternLayout());
```

Which creates a console appender, with a default PatternLayout. The default output of System.out is used.

1.2.2 Network appenders

Socket Appender

it sends LoggingEvent objects to a remote a log server, usually a SocketNode.

SocketHubAppender

it sends LoggingEvent objects to a set of remote log servers, usually a SocketNodes.

JMSAppender

A simple appender that publishes events to a JMS Topic. The events are serialized and transmitted as JMS message type ObjectMessage

NTEventLogAppender

Append to the NT event log system.

1.2.3 Special Appenders

AsyncAppender

It lets users log events asynchronously. It uses a bounded buffer to store logging events.

ExternallyRolledFileAppender

This appender listens on a socket on the port specified by the PORT_OPTION for a "RollOver" message. When such a message is received, the underlying log file is rolled over and an acknowledgment message is sent back to the process initiating the roll over.

JDBCAppender

It provides for sending log events to a database.

LF5Appender

It logs events to a swing based logging console. The swing console supports turning categories on and off, multiple detail level views, as well as full text searching and many other capabilities.

SMTPAppender

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

SyslogAppender

It sends messages to a remote syslog daemon.

TelnetAppender

It is a log4j appender that specializes in writing to a read-only socket.

WriterAppender

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

A `WriterAppender` can be created like this:

```
WriterAppender appender = null;

try {
    appender = new WriterAppender(new PatternLayout(), new
    FileOutputStream("filename"));
} catch(Exception e) {}
```

This `WriterAppender` uses the constructor that takes a `PatternLayout` and an `OutputStream` as arguments, in this case a `FileOutputStream` is used to output to a file, there are other constructors available.

1.3 Layout

The output of a log can be customized with layouts. The available layouts are

1. `SimpleLayout`
2. `PatternLayout`
3. `HTMLayout`
4. `XMLLayout`

1.3.1 simple layout

```
org.apache.log4j.SimpleLayout
```

`SimpleLayout` formats the output in a very simple manner, it prints the Level, then a dash '-' and then the log message. The demo class `LogWithSimpleLayout` is available in the eclipse project attached. The output of the program is as follows.

```
DEBUG - THIS IS A DEBUG MESSAGE
INFO - THIS IS A INFO MESSAGE
WARN - THIS IS A WARN MESSAGE
ERROR - THIS IS A ERROR MESSAGE
FATAL - THIS IS A FATAL MESSAGE
```

1.3.2 PatternLayout

```
org.apache.log4j.PatternLayout
```

`PatternLayout` formats the output based on a conversion pattern specified, or if none is specified, the default conversion pattern. The demo class `LogWithPatternLayout` is available in the eclipse project attached. The output of the program is as follows.

```
Milliseconds since program start: 0
Classname of caller: org.grassfield.log.LogWithPatternLayout
Date in ISO8601 format: 2008-03-30 12:50:21,078
Location of log event:
org.grassfield.log.LogWithPatternLayout.main(LogWithPatternLayout.java:33)
Message: THIS IS A DEBUG MESSAGE
```

Substitute symbol

SYMBOL	DESCRIPTION
%c	Logger, %c{2 } last 2 partial names
%C	Class name (full agony), %C{2 } last 2 partial names
%d{dd MMM yyyy HH:MM:ss }	Date, format see java.text.SimpleDateFormat
%F	File name
%l	Location (caution: compiler-option-dependently)
%L	Line number
%m	user-defined message
%M	Method name
%p	Level
%r	Milliseconds since program start
%t	Threadname
%x, %X	see Doku
%%	individual percentage sign
Caution: %C, %F, %l, %L, %M slow down program run!	

1.3.3 HTML layout

`org.apache.log4j.HTMLLayout`

HTMLLayout formats the output as a HTML table.

1.3.4 XMLLayout

`org.apache.log4j.xml.XMLLayout`

XMLLayout formats the output as a XML.

1.4 Levels

Levels denote priority or severity. They are explained in Logger. Loggers can be assigned levels. The set of possible levels, that is DEBUG, INFO, WARN, ERROR and FATAL are defined in the `org.apache.log4j.Level` class.

If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level.

The root logger resides at the top of the logger hierarchy. It always exists and always has an assigned level.

The logger is the core component of the logging process. In log4j, there are 5 normal levels Levels of logger available, the following is borrowed from the log4j API

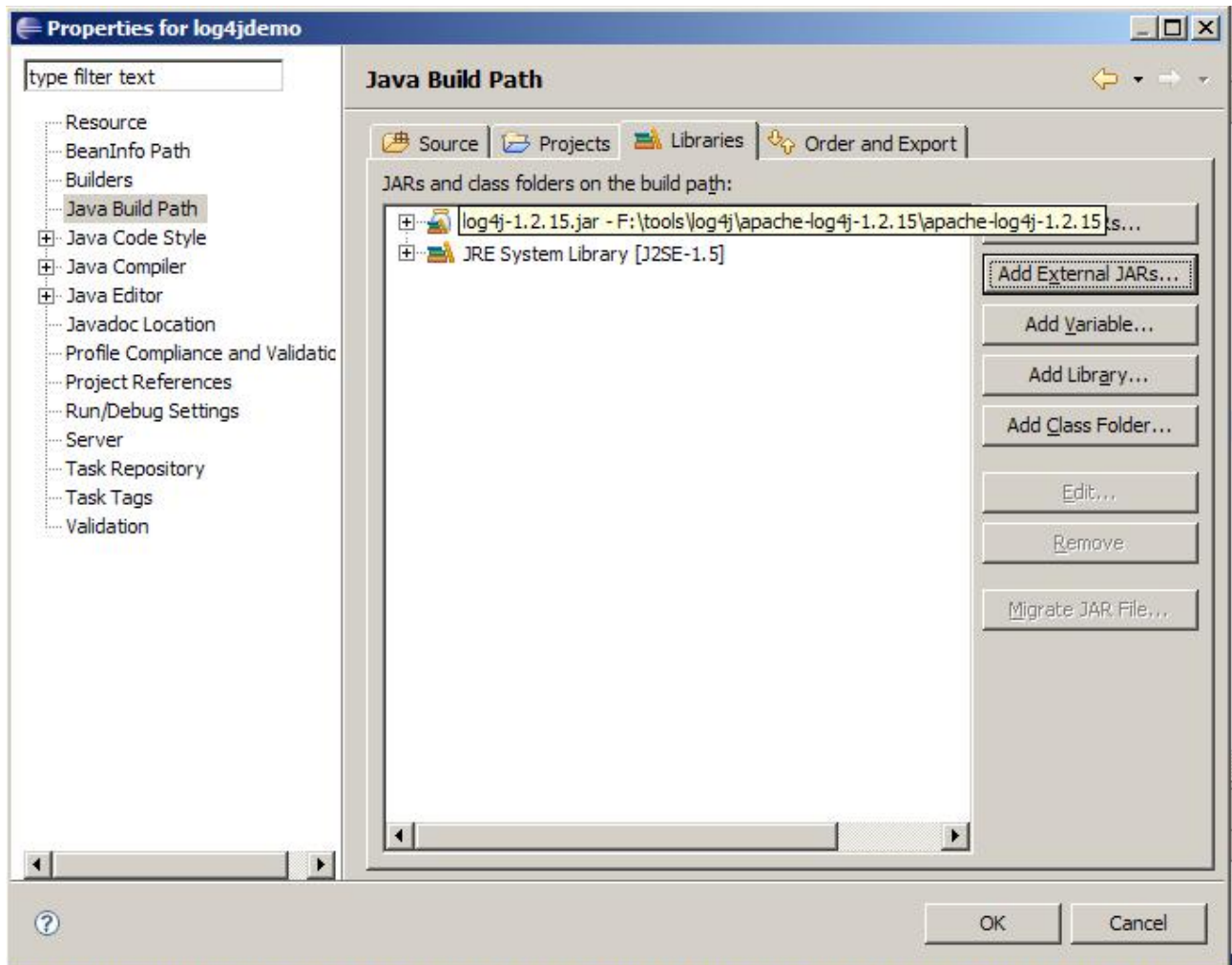
LEVEL	DEFINITION
static Level DEBUG	The DEBUG Level designates fine-grained informational events that are most useful to debug an application.
static Level INFO	The INFO level designates informational messages that highlight the progress of the application at coarse-grained level.
static Level WARN	The WARN level designates potentially harmful situations.
static Level ERROR	The ERROR level designates error events that might still allow the application to continue running.

LEVEL	DEFINITION
static Level FATAL	The FATAL level designates very severe error events that will presumably lead the application to abort.
static Level ALL	The ALL Level has the lowest possible rank and is intended to turn on all logging.
static Level OFF	The OFF Level has the highest possible rank and is intended to turn off logging.

2 Log4j installation

Apart from mastering the APIs we need to know how to set the work environment for log4j. Here I am explaining with eclipse based project.

1. Download the log4j distribution from <http://jakarta.apache.org/log4j/docs/download.html>.
2. Extract the archived files to some suitable directory.
3. Add the file log4j-1.2.15.jar to your CLASSPATH environment variable.



3 Pros & Cons of Log4j

3.1 Pros:

- log4j is optimized for speed. The burden of logging is minimized.
- log4j is based on a named logger hierarchy.
- log4j is thread-safe.
- The format of the logged message can be specified similar to C language's printf() function. For e.g. Formatting can be specified as "%r [%t] %-5p %c{2} %x - %m\n"
- log4j is not restricted to a predefined set of facilities.
- Logging behavior can be set at runtime using a configuration file. Configuration files can be property files or in XML format.
- log4j is designed to handle Java Exceptions from the start.
- log4j can direct its output to a file, the console, a java.io.OutputStream, java.io.Writer, a remote server using TCP, a remote Unix Syslog daemon, to a remote listener using JMS, to the NT EventLog or even send e-mail.
- 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.
- log4j supports multiple output appenders per logger.
- It supports hierarchical logging. It is possible to select (at runtime) which log statements are output at arbitrary granularity. Users can choose to implement their own log formats and output strategies.

3.2 Cons:

- Internationalization (Though specified in features, it is not mentioned anywhere else in the documentation) is not provided
- Asynchronous logging facility is not provided
- Threads are used in Handler objects. As threads should be managed only by the EJB container and should not be created in the code written by a developer, we have to find a workaround for this issue. The solutions like writing Java Messaging Service handler or calling the initialization code of the framework from start-up classes of application server need more investigation.