# Log4j Materials

Log4j is a Reliable, Fast and Flexible Logging Framework (APIs) written in Java which is distributed under the Apache Software License.

Log4j has three main components:

- **loggers:** Responsible for capturing logging information.
- **appenders :** Responsible for publishing logging information to various preferred destinations.
- **layouts:** Responsible to format logging information in different styles.

## log4j Features:

- log4j is thread-safe.
- log4j is optimized for speed.
- log4j is based on a named logger hierarchy.
- log4j supports multiple output appenders per logger.
- log4j supports internationalization.
- log4j is not restricted to a predefined set of facilities.
- Logging behavior can be set at runtime using a configuration file.
- log4j is designed to handle Java Exceptions from the start.
- log4j 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.
- log4j is fail-stop. However, altough it certainly strives to ensure delivery, log4j does not guarantee that each log statement will be delivered to its destination.

The latest log4j version, including full-source code, class files and documentation can be found at http://logging.apache.org/log4j/.

There are two type of objects available with Log4j framework.

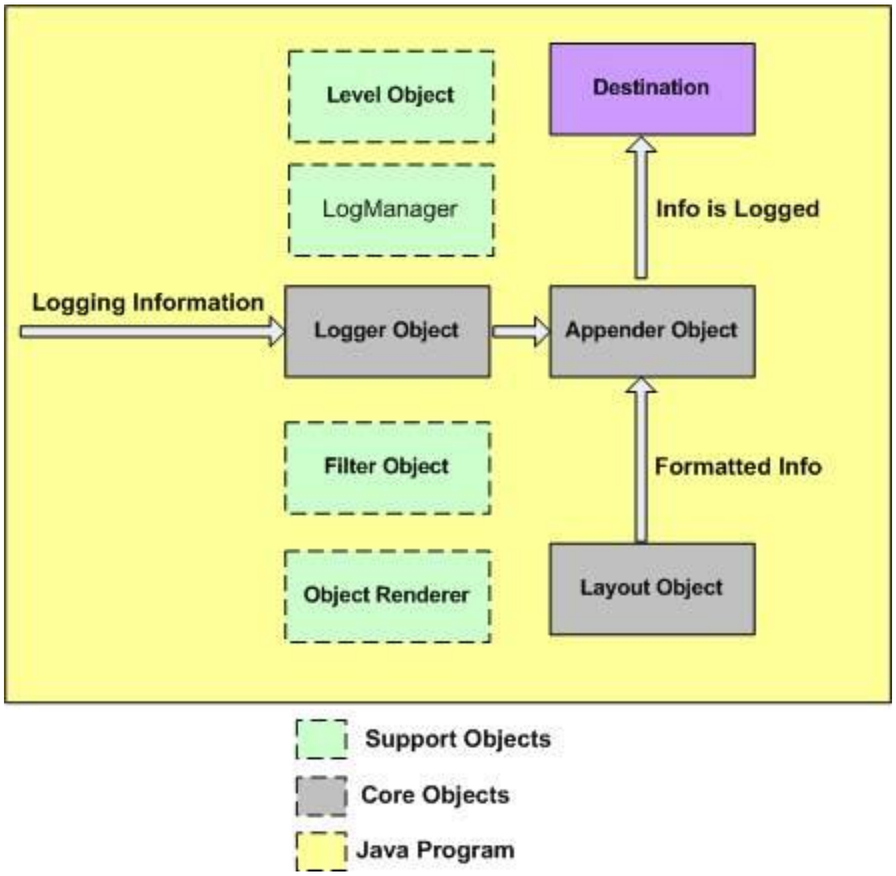
- **Core Objects:** These are mandatory objects of the framework and required to use the framework.

- **Support Objects:** These are optional objects of the framework and support core objects to perform addition but important tasks.

## Core Objects:

### Logger Object:

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



### Layout Object:

The layer provides objects which are used to format logging information in different styles. Layout layer provides support to appender objects to 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 lower level layer which provides Appender object. The Appender object is responsible for publishing logging information to various preferred destinations such as a database, file, console, UNIX Syslog etc.

## Support Objects:

There are other important objects in the log4j framework that play a vital role in the logging framework:

### Level Object:

The Level object defines the granularity and priority of any logging information. There are seven levels of logging defined within the API: OFF, DEBUG, INFO, ERROR, WARN, FATAL, and ALL.

### Filter Object:

The Filter object is used to analyze logging information and make further decisions on whether that information should be logged or not.

An Appender objects can have several Filter objects associated with them. If logging information is passed to a particular Appender object, all the Filter objects associated with that Appender need to approve the logging information before it can be published to the attached destination.

### ObjectRenderer:

The ObjectRenderer object is specialized in providing a String representation of different objects passed to the logging framework. This object is used by Layout objects to prepare the final logging information.

### LogManager:

The LogManager object manages the logging framework. It is responsible for reading the initial configuration parameters from a system-wide configuration file or a configuration class.

## log4j - Configuration

The *log4j.properties* file is a log4j configuration file which keeps properties in key-value pairs. By default, the LogManager looks for a file named *log4j.properties* in the CLASSPATH.

- The level of the root logger is defined as DEBUG and attaches appender named X to it.
- Set the appender named X to be a valid appender.
- Set the layout for the appender X

Following is the syntax of *log4j.properties* file for an appender X:

```
# Define the root logger with appender X
log4j.rootLogger = DEBUG, X

# Set the appender named X to be a File appender
log4j.appender.X=org.apache.log4j.FileAppender
```

```
# Define the layout for X appender
log4j.appender.X.layout=org.apache.log4j.PatternLayout
log4j.appender.X.layout.conversionPattern=%m%n
```

**log4j.properties Example:**

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

## Debug Level:

| Level | Description |
|---|---|
| **ALL** | All levels including custom levels. |
| **DEBUG** | Designates fine-grained informational events that are most useful to debug an application. |
| **ERROR** | Designates error events that might still allow the application to continue running. |
| **FATAL** | Designates very severe error events that will presumably lead the application to abort. |
| **INFO** | Designates informational messages that highlight the progress of the application at coarse-grained level. |
| **OFF** | The highest possible rank and is intended to turn off logging. |
| **TRACE** | Designates finer-grained informational events than the DEBUG. |
| **WARN** | Designates potentially harmful situations. |

## Appenders:

Apache log4j provides Appender objects which are primarily responsible for printing logging messages to different destinations such as consoles, files, sockets, NT event logs, etc.

Each Appender object has different properties associated with it, and these properties indicate the behavior of that object.

| Property | Description |
|---|---|
| **layout** | Appender uses the Layout objects and the conversion pattern associated with them to format the logging information. |
| **target** | The target may be a console, a file, or another item depending on the appender. |
| **level** | The level is required to control the filteration of the log messages. |
| **threshold** | Appender can have a threshold level associated with it independent of the logger level. The Appender ignores any logging messages that |

|  |  |
|---|---|
|  | have a level lower than the threshold level. |
| **filter** | The Filter objects can analyze logging information beyond level matching and decide whether logging requests should be handled by a particular Appender or ignored. |

We have used only one appender *FileAppender* in our example above. All the possible appender options are:

- AppenderSkeleton
- AsyncAppender
- ConsoleAppender
- DailyRollingFileAppender
- ExternallyRolledFileAppender
- FileAppender
- JDBCAppender
- JMSAppender
- LF5Appender
- NTEventLogAppender
- NullAppender
- RollingFileAppender
- SMTPAppender
- SocketAppender
- SocketHubAppender
- SyslogAppender
- TelnetAppender
- WriterAppender

## Logging in Files

**FileAppender Configuration:**

| 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 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 mean the logging information being appended to the end of the same file. |
| **bufferedIO** | This flag indicates whether we need buffered writing enabled. By default is set to false. |
| **bufferSize** | If bufferedI/O is enabled, this indicates the buffer size. By default 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
```

## Logging in Multiple Files:

There may be a requirement when you want to write your log message into multiple files for certain reasons like for example if file size reaches to a certain threshold etc.

To write your logging information into multiple files you would have to use *org.apache.log4j.RollingFileAppender* class which extends the *FileAppender* class and inherits all its properties.

There are following configurable parameters in addition to what have been mentioned above for FileAppender:

| Property | Description |
|---|---|
| **maxFileSize** | This is the critical size of the file above which the file will be rolled. Default value is 10MB |
| **maxBackupIndex** | This property denotes the number of backup files to be created. Default value is 1. |

Following is a sample configuration file *log4j.properties* for RollingFileAppender.

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.RollingFileAppender
# 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, should not overwrite
log4j.appender.FILE.Append=true

# Set the maximum file size before rollover
log4j.appender.FILE.MaxFileSize=5KB

# Set the the backup index
log4j.appender.FILE.MaxBackupIndex=2

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

## Daily Log File Generation:

There may be a requirement when you want to generate your log files on per day basis to keep a clean record of your logging information.

To write your logging information into files on daily basis you would have to use *org.apache.log4j.DailyRollingFileAppender* class which extends the *FileAppender* class and inherits all its properties.

There is only one important following configurable parameter in addition to what have been mentioned above for FileAppender:

| Property | Description |
|---|---|
| **DatePattern** | This indicates when to roll over the file, and the naming convention to be followed. By default roll over at midnight each day. |

DatePattern controls the rollover schedule using one of the following patterns:

| DatePattern | Description |
|---|---|
| **'.' yyyy-MM** | Roll over at the end of each month and the beginning of the next month. |
| **'.' yyyy-MM-dd** | This is the default value and roll over at midnight each day. |
| **'.' yyyy-MM-dd-a** | Roll over at midday and midnight of each day. |
| **'.' yyyy-MM-dd-HH** | Roll over at the top of every hour. |
| **'.' yyyy-MM-dd-HH-mm** | Roll over every minute. |
| **'.' yyyy-ww** | Roll over on the first day of each week depending upon the locale. |

Following is a sample configuration file *log4j.properties* to generate log files rolling over at midday and midnight of each day.

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
```

```
log4j.appender.FILE=org.apache.log4j.DailyRollingFileAppender
# 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, should not overwrite
log4j.appender.FILE.Append=true

# Set the DatePattern
log4j.appender.FILE.DatePattern='.' yyyy-MM-dd-a

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```