Using Log4j

Prerequisites

To log string from within a Java class, the following prerequisites are needed:

- The Log4j jar file must be on the Project's classpath.
- A Log4j configuration file must be in the root of the Project's source tree.
- A Logger object must be acquired within the Java code doing the logging.
- String must be written to the Logger object.

Core Concepts

Using Log4j is separated into the coding and configuration components. The URL for the Log4j 1.2.x JavaDocs is: http://logging.apache.org/log4j/1.2/apidocs/index.html.

Coding Log4j

The coding component of Log4j, involves the following steps (these examples are from the Artist class within the **Java Immersion Training)**.

Step 1) Import the Log4j Logger class:

```
import org.apache.log4j.Logger;
```

Step 2) Create a Logger object for use within this class, note the use of the class name, this allows very fine control over which log messages are to be written:

```
private static final Logger LOG = Logger.getLogger(Artist.class);
```

Step 3) Write Log messages at appropriate locations in the code:

```
public String getFName()
{
        if (LOG.isDebugEnabled()) LOG.debug("returning fName: " + fName);
        return fName;
}
```

or:

```
if (profile.isEmpty())
{
        LOG.error("Profile empty");
        throw new GalleryException("Profile must not be empty");
}
```

Logging Levels

Log4j has a number of Levels:

- **TRACE** – the finest detail level, use this level to log internal steps such as each record being read from a file, or the results of a computation. It is recommended you check if TRACE is enabled before you write a log message.
- **DEBUG** – the next level up, showing a coarser look at the code, use this level for the bulk of the "routine" log messages such as recording a method invocation, or recording which branch of an if statement is running. It is recommended you check if DEBUG is enabled before you write a log message (see example above).
- **INFO** – a medium level of detail, use this level to record log messages that provide meaningful details, such as which text file you are processing, or if you creating a new file because one does not yet exist.
- **WARN** – a higher level of logging, used to record events such as ignoring an exception, or using a default configuration. Most production system will be set to display WARN or higher messages.
- **ERROR** – the highest level of logging, used to record exceptions being thrown, or security violations.

Configuring Log4j

Log4j is configured via a log4j.properties or log4j.xml file. This file must be in the root of the Project's source tree. It is recommended this configuration file NOT be included in the deployed code. This allows the deployment server to provide a specific configuration customized for their environment.

Log4j has three core configuration concepts:

- **Appenders** – Where the log messages will be written
- **Layouts** – How the log messages will be written
- **Loggers** – Which log messages will be written

Appenders

Log4j has a number of Appenders, including (see the JavaDocs for org.apache.log4j.Appender):

- ConsoleAppender – the default destination
- FileAppender – a file destination
- RollingFileAppender – when the log file reaches a predefined size, and new file is created and the old file renamed.
- DailyRollingFileAppender – the file rolls over at midnight
- Many more including Database appenders, Messaging appenders, and remote connections appenders

Example (log4j.xml style):

```
<appender name="console" class="org.apache.log4j.ConsoleAppender">
   <param name="Target" value="System.out" />
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
         value="[%5p] %d{dd MMM yyyy HH:mm:ss} %c{1}:%L - %m%n" />
   </layout>
 </appender>

 <appender name="fileLog" class="org.apache.log4j.RollingFileAppender">
   <param name="file" value="../Log/gallery.log"/>
   <param name="Threshold" value="trace"/>
   <param name="MaxFileSize" value="1000KB"/>
   <param name="MaxBackupIndex" value="3"/>
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p %d %c{1}:%L - %m%n"/>
   </layout>
 </appender>
```

Layouts

Log4j has a number of Layouts, these Layouts determine how the log message is formated (see the JavaDocs for org.apache.log4j.PatternLayout). The example above shows this Layout:

```
<layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern"
        value="[%5p] %d{dd MMM yyyy HH:mm:ss} %c{1}:%L - %m%n" />
</layout>
```

Outputs (matches the ERROR logging above):
[ERROR] 06 Mar 2014 21:16:04 Artist:238 - Profile empty

Explanation of the conversion pattern:
[%5p] → output the Log Level within a set of [ ], left justified in 5 characters.
A space
%d{dd MMM yyyy HH:mm:ss} → output the date and time as:
- 2 digit day
- 3 character month abbreviation
- 4 digit year
- 2 digit hour in 24 hour format
- a colon
- 2 digit minutes
- a colon
- 2 digit seconds

A space
%c{1} → outputs one level of the fully qualified class name (i.e. package.class)
the example output Artist, where the fully qualified is: com.gallery.bizlogic.Artist
A colon
%L → output the line number within the Java class,
warning outputting the line number will effect performance
A space
A hyphen
A space
%m → outputs the log message
%n → outputs a newline

Loggers

A Logger determines which log messages will be written, and on which Appender those message will appear. The following example has four Loggers, different packages (or classes) maybe logged differently. Here the package com.gallery.bizlogic (and any sub-packages) will write WARN or higher messages to the Console. ERROR messages from the com.gallery,geo package will also be written to the console. TRACE or higher messages from the class com.gallery.security.GallerySecurity will be written to a File Appender (note this appender-ref refers to an Appender defined above). Finally all log message regardless of the package/class are written by the root Appender at the ERROR level to both the console and the File Appender. Please note because there are multiple Loggers writing log messages, those messages will appear multiple times.

```
<!-- applies to this package (and sub-packages) -->
<logger name="com.gallery.bizlogic">
        <level value="warn" />
        <appender-ref ref="console" />
 </logger>

<logger name="com.gallery.geo">
        <level value="error" />
        <appender-ref ref="console" />
 </logger>

 <logger name="com.gallery.security.GallerySecurity">
        <level value="trace" />
        <appender-ref ref="fileLog" />
 </logger>

 <!--  root logger configuration must be last -->
 <root>
        <priority value="error" />          <!--  trace, debug, info, warn, error, off -->
        <appender-ref ref="console" />
        <appender-ref ref="fileLog" />
 </root>
```