Log4jXmlFormat

Log4j XML Configuration Primer

Basic example

Below is a basic xml configuration file for log4j that will get you started:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <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 %c{1} - %m%n"/>
    </layout>
  </appender>

  <root>
    <priority value ="debug" />
    <appender-ref ref="console" />
  </root>

</log4j:configuration>
```

This will print all debug or higher messages to the console/screen. Items of note:

- The appender is defined first, with a name (in this case "console"). A layout is defined for the appender (in this case

  ```
  PatternLayout
  ```

  ), and a pattern is defined for the layout. What is required for the layout is layout specific, so check the javadoc description for the layout class you choose to use (

  ```
  PatternLayout
  ```

  is used most commonly).
- No loggers are defined in this example, but the configuration for the "root" logger is defined. It is configured to level debug, and the appender named "console" is attached to it. All loggers inherit from root, so in this example, all debug or higher messages from all loggers will be printed to the console appender.

XML Configuration Format

In order to better understand the more detailed examples, it is useful to understand the expected format for the xml configuration files. This is defined by the log4j.dtd which is located in the log4j distribution jar in the package org.apache.log4j.xml. The contents of this file will not be listed in its entirety, so please feel free to open/print the file yourself. If you are not familiar with xml dtd file formats, then you should go find a good book on that subject first.

Near the beginning of the file is the following declaration:

```
<!ELEMENT log4j:configuration (renderer*, appender*,(category|logger)*,root?, categoryFactory?)>
```

This element defines the expected structure of the xml configuration file: 0 or more renderer elements, followed by jocuri barbie0 or more appender elements, followed by 0 or more logger elements, followed by 0 or 1 root element, followed by 0 or 1 categoryFactory element. If this order is not followed, then errors will be printed by the xml parser at the time the xml file is read in. Also, as a note, the "category" element is the same as the logger element. Prior to log4j version 1.2, loggers were known as category. Much of the documentation still refers to category. Just understand that they are the same thing.

Further along in the log4j.dtd is the following declaration which defines the allowed attributes:

```
<!ATTLIST log4j:configuration
  xmlns:log4j              CDATA #FIXED "http://jakarta.apache.org/log4j/"
  threshold                (all|debug|info|warn|error|fatal|off|null) "null"
  debug                    (true|false|null)  "null"
>
```

- **debug** - Probably the most important attribute for log4:configuration, setting it to "true" will print out information as the configuration file is read and used to configure the log4j environment. Very useful when trying to fiure out why your configuration file is not doing what you expect.
- **threshold** - <yet to be described>

Understanding the expected structure of the xml configuration file makes it easier to concentrate on the specific elements one needs to configure.

Appender Configuration

One can instrument all the code one writes to output useful debug trace messages, but if log4j is not configured to have at least one appender, all will be for naught. None of the useful messages will be displayed anywhere.

Looking again to the log4j.dtd, appender elements are declared to be:

```
<!ELEMENT appender (errorHandler?, param*, layout?, filter*, appender-ref*)>
<!ATTLIST appender
  name                 ID         #REQUIRED
  class         CDATA        #REQUIRED
>
```

An appender element must have name and class attributes. The name is the value used to reference the appender in the rest of the configuration file. The class attribute should be the fully qualified class name of the appender class to use (ie

```
org.apache.log4j.ConsoleAppender
```

).

An appender element can also contain child elements:

- 0 or 1 **errorHandler** element - <yet to be described>
- 0 or more **param** elements - Each appender can be configured with setting specific to the functioning of the appender. This is implemented by getter and setter methods in the appender class. The param element is used to access the setter methods. The format for param elements is simple; they are atomic elements with a name attribute and a value attribute. The name attribute should be the name of the setter method with the "set" part of the method name omitted (ie method name "setTarget" would be "Target"). The value attribute is the value the setter method should be set with.
- 0 or 1 **layout** element - Not all appenders use or require a layout. For appenders that do, the layout element defines what layout class to use. The layout element has one attribute, class, which is the fully qualified class name of the layout class to use. Similar to the appender element, the layout element is allowed to have 0 or more param child elements. Again, the param elements are used to set specific values for the layout class, which varies based on what layout class is used.
- 0 or more **filter** elements - See the **Filter Configuration** section below for more details.
- 0 or more **appender-ref** elements - <yet to be described>

So, from the above, the simple example of the appender named "console" from the basic example starts to make more sense:

```
<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 %c{1} - %m%n"/>
  </layout>
</appender>
```

The name of of the appender is "console" and this is the name that is used to refer to the appender in the rest of the configuration file. The class to use for the appender is

```
org.apache.log4j.ConsoleAppender
```

.

The console appender has one param element defined. Looking at the javadoc for

```
ConsoleAppender
```

, the

```
setTarget
```

method is used to choose which console stream to print messages to, System.out or System.err. The example configures the appender to use System.out.

The console appender also has a layout element defined which uses

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

. Looking at the javadoc for

```
PatternLayout
```

, the `setConversionPattern` method takes a string describing the layout for messages. The details of this format can also be found in the javadoc.

The details of the configuration for a specific appender class vary from class to class. Your best bet is to review the javadoc for the appender class you want to use. Pay particular attention to the setter property methods and the values they expect. Each setter method can be accessed using the **param** element in the xml configuration.

Currently, the following appender classes exist:

- `org.apache.log4j.ConsoleAppender` ConsoleAppender
- `org.apache.log4j.FileAppender` FileAppender
- `org.apache.log4j.jdbc.JDBCAppender` JDBCAppender
- `org.apache.log4j.AsyncAppender` AsyncAppender
- `org.apache.log4j.net.JMSAppender` JMSAppender
- `org.apache.log4j.lf5.LF5Appender` LF5Appender
- `org.apache.log4j.nt.NTEventLogAppender` NTEventLogAppender
- `org.apache.log4j.varia.NullAppender` NullAppender
- `org.apache.log4j.net.SMTPAppender` SMTPAppender
- `org.apache.log4j.net.SocketAppender` SocketAppender
- `org.apache.log4j.net.SocketHubAppender` SocketHubAppender
- `org.apache.log4j.net.SyslogAppender` SyslogAppender
- `org.apache.log4j.net.TelnetAppender` TelnetAppender
- `org.apache.log4j.WriterAppender` WriterAppender

Logger Configuration

Now the appenders are configured. But how to configure loggers to output messages at a certain level? How to configure loggers to output to specific appender? Welcome to logger configuration.

The most important logger you need to configure is the root logger. From the simple example, this was done with the following configuration:

```
<root>
  <priority value ="debug" />
  <appender-ref ref="console" />
</root>
```

The root logger is configured to output log message at level "debug" or higher to the appender named "console". All loggers inherit their settings from the root logger, so with no other configuration settings, all loggers will output all of their messages to the "console" appender automatically. This may be fine for simple debugging, but eventually more specific logger configuration is going to be required.

Looking again to the log4j.dtd, logger elements are declared to be:

```
<!ELEMENT logger (level?,appender-ref*)>
<!ATTLIST logger
  name                ID        #REQUIRED
  additivity        (true|false) "true"
>
```

A logger element must have a name attribute. This is the name of the logger used when creating the Logger instance(usually the fully qualified class name). It can also have an optional additivity attribute. More on this later.

A logger element can also contain child elements:

- 0 or 1 **level** element - This defines the level of log messages that will be allowed to be logged for this logger. Normal usage has a value of "debug", "info", "warn", "error", or "fatal". Only that level or above will be reported to the log.
- 0 or more **appender-ref** elements - This references a defined appender that log messages from this logger should be directed to. Appender-ref elements are simple elements that have a ref attribute. The value for this attribute should be the name of the appender.

A typical logger configuration element would look similar to this:

```
<logger name="com.mycompany.apackage.MyClass">
  <level value="info"/>
</logger>
```

Logger Inheritance

<yet to be described>

Additivity

The output of a log statement of logger C will go to all the appenders in C and its ancestors. This is the meaning of the term "appender additivity".

However, if an ancestor of logger C, say P, has the additivity flag set to false, then C's output will be directed to all the appenders in C and it's ancestors upto and including P but not the appenders in any of the ancestors of P.

Loggers have their additivity flag set to true by default.

Example config;

```
<logger name="com.eatmutton.muttonsite.torque" additivity="false">
   <level value="info" />
   <appender-ref ref="local-torque" />
</logger>
```

Additivitiy section taken from http://logging.apache.org/log4j/docs/manual.html.

Converting Configuration Files To XML format

I have converted the configuration examples from the log4j manual to xml format. Hopefully people can use this to convert their own configuration files.

Example 1

```
# Set root logger level to DEBUG and its only appender to A1.
log4j.rootLogger=DEBUG, A1

# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <!-- A1 is set to be a ConsoleAppender -->
  <appender name="A1" class="org.apache.log4j.ConsoleAppender">
    <!-- A1 uses PatternLayout -->
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%-4r [%t] %-5p %c %x - %m%n"/>
    </layout>
  </appender>
  <root>
    <!-- Set root logger level to DEBUG and its only appender to A1 -->
    <priority value ="debug" />
    <appender-ref ref="A1" />
  </root>
</log4j:configuration>
```

Example 2

```
log4j.rootLogger=DEBUG, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout

# Print the date in ISO 8601 format
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c - %m%n

# Print only messages of level WARN or above in the package com.foo.
log4j.logger.com.foo=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="A1" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
      <!-- Print the date in ISO 8601 format -->
      <param name="ConversionPattern" value="%d [%t] %-5p %c - %m%n"/>
    </layout>
  </appender>
  <logger name="com.foo">
    <!-- Print only messages of level warn or above in the package com.foo -->
    <level value="warn"/>
  </logger>
  <root>
    <priority value ="debug" />
    <appender-ref ref="A1" />
  </root>
</log4j:configuration>
```

Example 3

```
log4j.rootLogger=debug, stdout, R

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

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=example.log

log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <appender name="stdout" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
      <!-- Pattern to output the caller's file name and line number -->
      <param name="ConversionPattern" value="%5p [%t] (%F:%L) - %m%n"/>
    </layout>
  </appender>
  <appender name="R" class="org.apache.log4j.RollingFileAppender">
    <param name="file" value="example.log"/>
    <param name="MaxFileSize" value="100KB"/>
    <!-- Keep one backup file -->
    <param name="MaxBackupIndex" value="1"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p %t %c - %m%n"/>
    </layout>
  </appender>
  <root>
    <priority value ="debug" />
    <appender-ref ref="stdout" />
    <appender-ref ref="R" />
  </root>
</log4j:configuration>
```

Filter Configuration

Filters can be defined at appender level. For example, to filter only certain levels, the LevelRangeFilter can be used like this:

```
        <appender name="TRACE" class="org.apache.log4j.ConsoleAppender">
                <layout class="org.apache.log4j.PatternLayout">
                        <param name="ConversionPattern" value="[%t] %-5p %c - %m%n" />
                </layout>
                <filter class="org.apache.log4j.varia.LevelRangeFilter">
                        <param name="levelMin" value="DEBUG" />
                        <param name="levelMax" value="DEBUG" />
                </filter>
        </appender>
```

Advanced Topics

<yet to be described>

More examples

(Please feel free to add your own configuration examples here)

Note that TimeBasedRollingPolicy can only be configured with xml, not log4j.properties

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<!-- Note that this file is refreshed by the server every 60seconds, as specified in web.xml -->

<log4j:configuration debug="true">

        <appender name="ROLL" class="org.apache.log4j.rolling.RollingFileAppender">
                <!-- The active file to log to -->
                <param name="file" value="/applogs/myportal/portal.log" />
                <param name="append" value="true" />
                <param name="encoding" value="UTF-8" />

                <rollingPolicy class="org.apache.log4j.rolling.TimeBasedRollingPolicy">
                        <!-- The file to roll to, this is a fairly intelligent parameter, if the file
                        ends in .gz, it gzips it, based on the date stamp it rolls at that time,
                        default is yyyy-MM-dd, (rolls at midnight)
                        See: http://logging.apache.org/log4j/companions/extras/apidocs/org/apache/log4j/rolling
/TimeBasedRollingPolicy.html -->
                        <param name="FileNamePattern" value="/applogs/myportal/portal.%d.log.gz" />
                </rollingPolicy>

                <layout class="org.apache.log4j.PatternLayout">
                        <!-- The log message pattern -->
                        <param name="ConversionPattern" value="%5p %d{ISO8601} [%t][%x] %c - %m%n" />
                </layout>
        </appender>

        <!-- Loggers to filter out various class paths -->

        <logger name="org.hibernate.engine.loading.LoadContexts" additivity="false">
                <level value="error"/>
                <appender-ref ref="ROLL" />
        </logger>

        <!-- Debugging loggers -->

        <!-- Uncomment to enable debug on calpoly code only -->
        <!--
        <logger name="edu.calpoly">
                <level value="debug"/>
                <appender-ref ref="ROLL" />
        </logger>
        -->

        <root>
                <priority value="info" />
                <appender-ref ref="ROLL" />
        </root>

</log4j:configuration>
```