LOG4J - PATTERNLAYOUT

If you want to generate your logging information in a particular format based on a pattern, then you can use **org.apache.log4j.PatternLayout** to format your logging information.

The PatternLayout class extends the abstract **org.apache.log4j.Layout** class and overrides the **format()** method to structure the logging information according to a supplied pattern.

PatternLayout is also a simple Layout object that provides the following-*Bean Property* which can be set using the configuration file:

Sr.No.	Property & Description
1	**conversionPattern** Sets the conversion pattern. Default is %r [%t] %p %c %x - %m%n

Pattern Conversion Characters

The following table explains the characters used in the above pattern and all other characters that you can use in your custom pattern:

Conversion Character	Meaning
c	Used to output the category of the logging event. For example, for the category name "a.b.c" the pattern %c{2} will output "b.c".
C	Used to output the fully qualified class name of the caller issuing the logging request. For example, for the class name "org.apache.xyz.SomeClass", the pattern %C{1} will output "SomeClass".
d	Used to output the date of the logging event. For example, %d{HH:mm:ss,SSS} or %d{dd MMM yyyy HH:mm:ss,SSS}.
F	Used to output the file name where the logging request was issued.
l	Used to output location information of the caller which generated the logging event.
L	Used to output the line number from where the logging request was issued.
m	Used to output the application supplied message associated with the logging event.
M	Used to output the method name where the logging request was issued.
n	Outputs the platform dependent line separator character or characters.
p	Used to output the priority of the logging event.
r	Used to output the number of milliseconds elapsed from the construction of the layout until the creation of the logging event.

t	Used to output the name of the thread that generated the logging event.
x	Used to output the NDC (nested diagnostic context) associated with the thread that generated the logging event.
X	The X conversion character is followed by the key for the MDC. For example, X{clientIP} will print the information stored in the MDC against the key clientIP.
%	The literal percent sign. %% will print a % sign.

Format Modifiers

By default, the relevant information is displayed as output as is. However, with the aid of format modifiers, it is possible to change the minimum field width, the maximum field width, and justification.

Following table covers various modifiers scenarios:

Format modifier	left justify	minimum width	maximum width	comment
%20c	false	20	none	Left pad with spaces if the category name is less than 20 characters long.
%-20c	true	20	none	Right pad with spaces if the category name is less than 20 characters long.
%.30c	NA	none	30	Truncate from the beginning if the category name is longer than 30 characters.
%20.30c	false	20	30	Left pad with spaces if the category name is shorter than 20 characters. However, if the category name is longer than 30 characters, then truncate from the beginning.
%-20.30c	true	20	30	Right pad with spaces if the category name is shorter than 20 characters. However, if category name is longer than 30 characters, then truncate from the beginning.

PatternLayout Example

Following is a simple configuration file for PatternLayout:

```
# Define the root logger with appender file
log = /usr/home/log4j
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=%d{yyyy-MM-dd}-%t-%x-%-5p-%-10c:%m%n
```

Now consider the following Java Example which would generate logging information:

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

```
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class log4jExample{
   /* Get actual class name to be printed on */
   static Logger log = Logger.getLogger(log4jExample.class.getName());

   public static void main(String[] args)throws IOException,SQLException{
      log.debug("Hello this is an debug message");
      log.info("Hello this is an info message");
   }
}
```

Compile and run the above program. It would create a log.out file in /usr/home/log4j directory which would have the following log information:

```
2010-03-23-main--DEBUG-log4jExample:Hello this is an debug message
2010-03-23-main--INFO -log4jExample:Hello this is an info message
```