org.apache.log4j

Class PatternLayout

java.lang.Object
  └ org.apache.log4j.Layout
    └ **org.apache.log4j.PatternLayout**

All Implemented Interfaces:
OptionHandler

```
public class PatternLayout
extends Layout
```

A flexible layout configurable with pattern string. This code is known to have synchronization and other issues which are not present in org.apache.log4j.EnhancedPatternLayout. EnhancedPatternLayout should be used in preference to PatternLayout. EnhancedPatternLayout is distributed in the log4j extras companion.

The goal of this class is to `format` a `LoggingEvent` and return the results as a String. The results depend on the *conversion pattern*.

The conversion pattern is closely related to the conversion pattern of the printf function in C. A conversion pattern is composed of literal text and format control expressions called *conversion specifiers*.

You are free to insert any literal text within the conversion pattern.

Each conversion specifier starts with a percent sign (%) and is followed by optional *format modifiers* and a *conversion character*. The conversion character specifies the type of data, e.g. category, priority, date, thread name. The format modifiers control such things as field width, padding, left and right justification. The following is a simple example.

Let the conversion pattern be **"%-5p [%t]: %m%n"** and assume that the log4j environment was set to use a PatternLayout. Then the statements

```
Category root = Category.getRoot();
root.debug("Message 1");
root.warn("Message 2");
```

would yield the output

```
DEBUG [main]: Message 1
WARN  [main]: Message 2
```

Note that there is no explicit separator between text and conversion specifiers. The pattern parser knows when it has reached the end of a conversion specifier when it reads a conversion character. In the example above the conversion specifier **%-5p** means the priority of the logging event should be left justified to a width of five characters. The recognized conversion characters are

Conversion Character	Effect
c	Used to output the category of the logging event. The category conversion specifier can be optionally followed by *precision specifier*, that is a decimal constant in brackets. If a precision specifier is given, then only the corresponding number of right most components of the category name will be printed. By default the category name is printed in full. 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. This conversion specifier can be optionally followed by *precision specifier*, that is a decimal constant in brackets. If a precision specifier is given, then only the corresponding number of right most components of the class name will be printed. By default the class name is output in fully qualified form. For example, for the class name "org.apache.xyz.SomeClass", the pattern **%C{1}** will output "SomeClass". **WARNING** Generating the caller class information is slow. Thus, use should be avoided unless execution speed is not an issue.
d	Used to output the date of the logging event. The date conversion specifier may be followed by a *date format specifier* enclosed between braces. For example, **%d{HH:mm:ss,SSS}** or **%d{dd MMM yyyy HH:mm:ss,SSS}**. If no date format specifier is given then ISO8601 format is assumed. The date format specifier admits the same syntax as the time pattern string of the `SimpleDateFormat`. Although part of the standard JDK, the performance of `SimpleDateFormat` is quite poor. For better results it is recommended to use the log4j date formatters. These can be specified using one of the strings "ABSOLUTE", "DATE" and "ISO8601" for specifying `AbsoluteTimeDateFormat`, `DateTimeDateFormat` and respectively `ISO8601DateFormat`. For example, **%d{ISO8601}** or **%d{ABSOLUTE}**. These dedicated date formatters perform significantly better than `SimpleDateFormat`.
F	Used to output the file name where the logging request was issued. **WARNING** Generating caller location information is extremely slow and should be avoided unless execution speed is not an issue.
l	Used to output location information of the caller which generated the logging event. The location information depends on the JVM implementation but usually consists of the fully qualified name of the calling method followed by the callers source the file name and

	line number between parentheses. The location information can be very useful. However, its generation is *extremely* slow and should be avoided unless execution speed is not an issue.
L	Used to output the line number from where the logging request was issued. **WARNING** Generating caller location information is extremely slow and should be avoided unless execution speed is not an issue.
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. **WARNING** Generating caller location information is extremely slow and should be avoided unless execution speed is not an issue.
n	Outputs the platform dependent line separator character or characters. This conversion character offers practically the same performance as using non-portable line separator strings such as "\n", or "\r\n". Thus, it is the preferred way of specifying a line separator.
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	Used to output the MDC (mapped diagnostic context) associated with the thread that generated the logging event. The **X** conversion character *must* be followed by the key for the map placed between braces, as in **%X{clientNumber}** where `clientNumber` is the key. The value in the MDC corresponding to the key will be output. See MDC class for more details.
%	The sequence %% outputs a single percent sign.

By default the relevant information is 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.

The optional format modifier is placed between the percent sign and the conversion character.

The first optional format modifier is the *left justification flag* which is just the minus (-) character. Then comes the optional *minimum field width* modifier. This is a decimal constant that represents the minimum number of characters to output. If the data item requires fewer characters, it is padded on either the left or the right until the minimum width is reached. The default is to pad on the left (right justify) but you can specify right padding with the left justification flag. The padding character is space. If the data item is larger than the minimum field width, the field is expanded to accommodate the data. The value is never truncated.

This behavior can be changed using the *maximum field width* modifier which is designated by a period followed by a decimal constant. If the data item is longer than the maximum field, then the extra characters are removed from the *beginning* of the data item and not from the end. For example, it the maximum field width is eight and the data item is ten characters long, then the first two characters of the data item are dropped. This behavior deviates from the printf function in C where truncation is done from the end.

Below are various format modifier examples for the category conversion specifier.

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 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.

Below are some examples of conversion patterns.

%r [%t] %-5p %c %x - %m%n

This is essentially the TTCC layout.

%-6r [%15.15t] %-5p %30.30c %x - %m%n

Similar to the TTCC layout except that the relative time is right padded if less than 6 digits, thread name is right padded if less than 15 characters and truncated if longer and the category name is left padded if shorter than 30 characters and truncated if longer.

The above text is largely inspired from Peter A. Darnell and Philip E. Margolis' highly recommended book "C -- a Software Engineering Approach", ISBN 0-387-97389-3.

Since:
0.8.2
Author:

James P. Cakalic, Ceki Gülcü

Field Summary

protected int	**BUF_SIZE**
static String	**DEFAULT_CONVERSION_PATTERN** Default pattern string for log output.
protected int	**MAX_CAPACITY**
static String	**TTCC_CONVERSION_PATTERN** A conversion pattern equivalent to the TTCCCLayout.

Fields inherited from class org.apache.log4j.Layout
LINE_SEP, LINE_SEP_LEN

Constructor Summary

PatternLayout() Constructs a PatternLayout using the DEFAULT_LAYOUT_PATTERN.
PatternLayout(String pattern) Constructs a PatternLayout using the supplied conversion pattern.

Method Summary

void	**activateOptions**() Does not do anything as options become effective
protected PatternParser	**createPatternParser**(String pattern) Returns PatternParser used to parse the conversion string.
String	**format**(LoggingEvent event) Produces a formatted string as specified by the conversion pattern.
String	**getConversionPattern**() Returns the value of the **ConversionPattern** option.
boolean	**ignoresThrowable**() The PatternLayout does not handle the throwable contained within LoggingEvents.
void	**setConversionPattern**(String conversionPattern) Set the **ConversionPattern** option.

Methods inherited from class org.apache.log4j.Layout
getContentType, getFooter, getHeader

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

DEFAULT_CONVERSION_PATTERN

```
public static final String DEFAULT_CONVERSION_PATTERN
```

Default pattern string for log output. Currently set to the string **"%m%n"** which just prints the application supplied message.

See Also:
Constant Field Values

TTCC_CONVERSION_PATTERN

```
public static final String TTCC_CONVERSION_PATTERN
```

A conversion pattern equivalent to the TTCCCLayout. Current value is **%r [%t] %p %c %x - %m%n**.

See Also:
Constant Field Values

BUF_SIZE

```
protected final int BUF_SIZE
```

See Also:
Constant Field Values

MAX_CAPACITY

```
protected final int MAX_CAPACITY
```

See Also:
Constant Field Values

Constructor Detail

PatternLayout

```
public PatternLayout()
```

Constructs a PatternLayout using the DEFAULT_LAYOUT_PATTERN. The default pattern just produces the application supplied message.

PatternLayout

```
public PatternLayout(String pattern)
```

Constructs a PatternLayout using the supplied conversion pattern.

Method Detail

setConversionPattern

`public void **setConversionPattern**(String conversionPattern)`

Set the **ConversionPattern** option. This is the string which controls formatting and consists of a mix of literal content and conversion specifiers.

---

getConversionPattern

`public String **getConversionPattern**()`

Returns the value of the **ConversionPattern** option.

---

activateOptions

`public void **activateOptions**()`

Does not do anything as options become effective

---

ignoresThrowable

`public boolean **ignoresThrowable**()`

The PatternLayout does not handle the throwable contained within `LoggingEvents`. Thus, it returns `true`.

Specified by:
`ignoresThrowable` in class `Layout`

Since:
0.8.4

---

createPatternParser

`protected PatternParser **createPatternParser**(String pattern)`

Returns PatternParser used to parse the conversion string. Subclasses may override this to return a subclass of PatternParser which recognize custom conversion characters.

Since:
0.9.0

---

format

`public String **format**(LoggingEvent event)`

Produces a formatted string as specified by the conversion pattern.

Specified by:

`format` in class `Layout`
