Apache Log4j (Ver 2)

Background

The Log4j Analysis App presents a predefined set of dashboards and gadgets visualizing log4j logs. The Log4j analysis pack addresses the need to manage and debug Java applications and infrastructure during development, testing, and production. The App helps measure, troubleshoot, and optimize Java based applications with visualization and investigation dashboards.

Steps

1. Add Log Data In XpoLog, When adding a log to XpoLog you can now select the Log Type (logtype) for Apache log4j the are the following logtypes:
    a. *log4j*
2. Once all required information is set click next and edit the log pattern, this step is crucial to the accuracy and deployment of the App. Use the following conversion table to build the XpoLog pattern out of the log4j log format.

Example

In the Apache Log4J configuration file, can be either properties files, XML file, or in some case the log format was created programmatically for which you can manually create the pattern for the data.

log4j.appender.xpolog.layout.ConversionPattern=[%d] [%t] [%p] [%c] [%l] %m%n

The following sequence is the log structure definition for the log4j log [%d] [%t] [%p] [%c] [%l] %m%n

In XpoLog such pattern will be translated into:

for more information see below:

[{date:Date,locale=en,yyyy-MM-dd HH:mm:ss,SSS}] [{text:Thread,ftype=thread}] [{priority:Priority,ftype=severity;,DEBUG;INFO;WARNING;ERROR;FATAL}] [{string:Class,ftype=class}] [{string:Method,ftype=method}({text:Source,ftype=sourcecode}:{number:LineNumber,ftype=linenumber})] {string:Message,ftype=message}

Apache Log4j Conversion Table

logtype should be set to: *log4j*

Name and Appears with	Description	XpoLog Pattern
%c{precision} %logger{precision}	Outputs the name of the logger that published the logging event. The logger conversion specifier can be optionally followed by*precision specifier*, which consists of a decimal integer, or a pattern starting with a decimal integer. If a precision specifier is given and it is an integer value, then only the corresponding number of right most components of the logger name will be printed. If the precision contains other non-integer characters then the name will be abbreviated based on the pattern. If the precision integer is less than one the right-most token will still be printed in full. By default the logger name is printed in full.	{text:Logger,ftype=logger}
%C{precision} %class{precision}	Outputs the fully qualified class name of the caller issuing the logging request. This conversion specifier can be optionally followed by*precision specifier*, that follows the same rules as the logger name converter. Generating the class name of the caller (location information) is an expensive operation and may impact performance. Use with caution.	{text:Class,ftype=class}

<table>
<tr>
<td>%d{pattern}
%date{pattern}</td>
<td>Outputs the date of the logging event. The date conversion specifier may be followed by a set of braces containing a date and time pattern string per SimpleDateFormat .

The predefined formats are DEFAULT, ABSOLUTE, COMPACT, DATE, ISO8601, and ISO8601_BASIC.

You can also use a set of braces containing a time zone id per java.util.TimeZone.getTimeZone. If no date format specifier is given then ISO8601 format is assumed
%d{UNIX} outputs the UNIX time in seconds.
%d{UNIX_MILLIS} outputs the UNIX time in milliseconds. The UNIX time is the difference, in seconds for UNIX and in milliseconds for UNIX_MILLIS, between the current time and midnight, January 1, 1970 UTC. While the time unit is milliseconds, the granularity depends on the operating system (Windows). This is an efficient way to output the event time because only a conversion from long to String takes place, there is no Date formatting involved.</td>
<td>{date:Date,<DATE_PATTERN>}

{date:Date,locale=<?>,yyyy-MM-dd HH:mm:ss,SSS}

Note: (use default locale)

{date:Date,locale=<?>,<same pattern>}

{date:Date,locale=<?>,yyyy-MM-dd HH:mm:ss,SSS}

Note: (use default locale)

Create the relevant date pattern for each option or us

Outputs the date of the logging event. The date conv
set of braces containing a date and time pattern strin

The predefined formats are DEFAULT, ABSOLUTE,
and ISO8601_BASIC.

You can also use a set of braces containing a time zo
ne. If no date format specifier is given then ISO8601

<table>
<tr><th>Pattern</th><th>Example</th></tr>
<tr><td>%d{DEFAULT}</td><td>2012-11-02 14</td></tr>
<tr><td>%d{ISO8601}</td><td>2012-11-02T1</td></tr>
<tr><td>%d{ISO8601_BASIC}</td><td>20121102T14</td></tr>
<tr><td>%d{ABSOLUTE}</td><td>14:34:02,781</td></tr>
<tr><td>%d{DATE}</td><td>02 Nov 2012 1</td></tr>
<tr><td>%d{COMPACT}</td><td>2012110214</td></tr>
<tr><td>%d{HH:mm:ss,SSS}</td><td>14:34:02,781</td></tr>
<tr><td>%d{dd MMM yyyy HH:mm:ss,SSS}</td><td>02 Nov 2012 1</td></tr>
<tr><td>%d{HH:mm:ss}{GMT+0}</td><td>18:34:02</td></tr>
<tr><td>%d{UNIX}</td><td>1351866842</td></tr>
<tr><td>%d{UNIX_MILLIS}</td><td>1351866842</td></tr>
</table>

%d{UNIX} outputs the UNIX time in seconds. %d{UN
milliseconds. The UNIX time is the difference, in secc
UNIX_MILLIS, between the current time and midnigh
unit is milliseconds, the granularity depends on the o
efficient way to output the event time because only a
place, there is no Date formatting involved.</td>
</tr>
<tr>
<td>%enc{pattern}
%encode{pattern}</td>
<td>Escape newlines and HTML special characters in the specified pattern.</td>
<td>Create pattern according to the relevant encoding:

%enc{%m} = {string:Message,ftype=message}

%enc{%mdc{key}} = (string:Key}</td>
</tr>
</table>

%ex|exception|throwable {["none" |"full" |depth |"short" |"short.className" |"short.fileName" |"short.lineNumber" |"short.methodName" |"short.message" |"short.localizedMessage"]}	Outputs the Throwable trace bound to the LoggingEvent, by default this will output the full trace as one would normally find with a call to Throwable.printStackTrace(). You can follow the throwable conversion word with an option in the form **%throwable{option}**. **%throwable{short}** outputs the first line of the Throwable. **%throwable{short.className}** outputs the name of the class where the exception occurred. **%throwable{short.methodName}** outputs the method name where the exception occurred. **%throwable{short.fileName}** outputs the name of the class where the exception occurred. **%throwable{short.lineNumber}** outputs the line number where the exception occurred. **%throwable{short.message}** outputs the message. **%throwable{short.localizedMessage}** outputs the localized message. **%throwable{n}** outputs the first n lines of the stack trace. Specifying **%throwable{none}** or **%throwable{0}** suppresses output of the exception.	{string:Throwable,ftype=throwable} Remark: XpoLog need to consider adding a dedicate
%F %**file**	Outputs the file name where the logging request was issued	{text:Class,ftype=class}
%highlight{pattern}{style}	Adds ANSI colors to the result of the enclosed pattern based on the current event's logging level.	N/A
%K{key} %**map**{key} %**MAP**{key}	Outputs the entries in a MapMessage, if one is present in the event. The **K** conversion character can be followed by the key for the map placed between braces, as in **%K{clientNumber}** where clientNumberis the key. The value in the Map corresponding to the key will be output. If no additional sub-option is specified, then the entire contents of the Map key value pair set is output using a format {{key1,val1},{key2,val2}}	{text:Key}
%l **location**	Outputs location information of the caller which generated the logging event.	{text:Method,ftype=method}({text:Class,ftype=class}:{
%L %**line**	Outputs the line number from where the logging request was issued.	{number:LineNumber,ftype=linenumber}
%m %**msg** %**message**	Outputs the application supplied message associated with the logging event.	{string:Message,ftype=message}
%M %**method**	Outputs the method name where the logging request was issued.	{text:Method,ftype=method}
%marker	The name of the marker, if one is present.	{text:Marker,ftype=marker}
%n	Outputs the platform dependent line separator character or characters.	{eol}

%p|level{*level=label*, *level=label*, ...} %**p|level**{length=*n*} **%p|level**{lowerCase=*true*|*false*}	Outputs the level of the logging event. You provide a level name map in the form "level=value, level=value" where level is the name of the Level and value is the value that should be displayed instead of the name of the Level.	{priority:Priority,ALL;TRACE;DEBUG;INFO;WARN;EI For the example below: {priority:Priority, All; Trace; Debug; Info; Warning; Err Outputs the level of the logging event. You provide a "level=value, level=value" where level is the name of should be displayed instead of the name of the Level. For example: %level{WARN=Warning, DEBUG=Debug, ERROR=E Alternatively, for the compact-minded: %level{WARN=W, DEBUG=D, ERROR=E, TRACE=T More succinctly, for the same result as above, you ca %level{length=1} If the length is greater than a level name length, the la You can combine the two kinds of options: %level{ERROR=Error, length=2} This give you the Error level name and all other level Finally, you can output lower-case level names (the d %level{lowerCase=true}
%r %**relative**	Outputs the number of milliseconds elapsed since the JVM was started until the creation of the logging event.	{number:LogSpeed,ftype=logprocesstimemilli}
%replace{pattern}{regex}{substitution}	Replaces occurrences of 'regex', a regular expression, with its replacement 'substitution' in the string resulting from evaluation of the pattern. For example, "%replace(%msg}{\s}{}" will remove all spaces contained in the event message.	{string:<PATTERN_NAME>}
%rEx["none"|"short"|"full"|depth],[filters(packages)} %**rException**["none"|"short"|"full"|depth],[filters(packages)} %**rThrowable**["none"|"short"|"full"|depth],[filters(packages)}	The same as the %throwable conversion word but the stack trace is printed starting with the first exception that was thrown followed by each subsequent wrapping exception.	{string:NestedException,ftype= nestedexception}
%sn %**sequenceNumber**	Includes a sequence number that will be incremented in every event. The counter is a static variable so will only be unique within applications that share the same converter Class object.	{number:SequenceNumber,ftype=sequencenumber}
%style{pattern}{ANSI style}	Uses ANSI escape sequences to style the result of the enclosed pattern. The style can consist of a comma separated list of style names from the following table.	{string:<PATTERN_NAME>}
%t %**thread**	Outputs the name of the thread that generated the logging event.	{text:Thread,ftype=thread}
%x %**NDC**	Outputs the Thread Context Stack (also known as the Nested Diagnostic Context or NDC) associated with the thread that generated the logging event.	{string:NDC}
%X{key} %**mdc**{key} %**MDC**{key}	Outputs the Thread Context Map (also known as the Mapped Diagnostic Context or MDC) associated with the thread that generated the logging event.	{string:NDC_<KEY>}
%u{"RANDOM" | "TIME"} %**uuid**	Includes either a random or a time-based UUID.	{text:UUID,ftype=uniqueid}
%xEx{"none"|"short"|"full"|depth],[filters(packages)} %**xException**["none"|"short"|"full"|depth],[filters(packages)} %**xThrowable**["none"|"short"|"full"|depth],[filters(packages)}	The same as the %throwable conversion word but also includes class packaging information.	{string:NestedException,ftype=nestedexception} Include package information.