Polarion Log Parser / Analyzer

The Polarion Log Parser / Analyzer is a simple tool to parse Polarion log files and output an analysis of various aspects decoded from them. It can produce either text-based (machine-readable) or cleaner (human-readable) HTML output.

1 Execution

1.1 Prerequisites

To run the tool, you must install a Java runtime environment (RTE) on your computer. This tool is built using Java 11, so it requires a Java 11 RTE or newer.

We recommend adding the Java binaries to the OS PATH environment variable so you can call the Java interpreter directly.

1.2 Execution

The tool is distributed as a JAR file and can be executed via the following command:

```
java -jar plpa.jar [options] <log_file_path>
```

1.3 Program parameters

You can add the path to the Polarion log file for basic, text-based output or add the following parameters to return additional data:

`-a` Adding this parameter prints all lines of each multi-output rule. This works only for text-based output. In most cases, it does not make sense to print out hundreds or thousands of log entries of the same type with the same information, so by default, the analyzer only prints out what it considers meaningful results (10 by default). If you need all lines printed out, you can do so by adding this parameter.

`-h <html_file_path>` Adding this parameter creates an additional, human-readable, HTML file at the specified location.
Every output line has a tooltip that displays the original log file line.

`-l <log_parsing_rules_file_path>` Parses Polarion log files that are in a different format than the default.

`-r <analysis_rules_file_path>` Lets you add your own rules in addition to the defaults (Note: New rules with the same name and in the same section as existing ones overwrite them).

`-e` Shows empty ERROR, EXCEPTION, and TXLOGGER results. (They are hidden by default.)

`-t <txlogger_time_threshold>` Only processes TXLOGGER lines with a Total time greater or equal to the threshold values. Shorter transactions are not processed (they usually don't cause any issues). The default value is 3.0 seconds.

The last parameter must always be the log file path. If the Polarion log file has older rotations in the same directory as the main/last log file, they are also analyzed in the correct order.

2 Rules

The Polarion log file analysis is done via Regular expression-based rules written in JSON.

There are two sets of rules:

1. Polarion log line string parsing rules
2. Polarion analysis rules

2.1 Log line parsing rules

This set of rules is defined for understanding the Polarion log format. By default, Polarion logs are formatted as follows:

```
<timestamp> [<thread>] <level> <class> - <message>
```

The rules parse the lines one after another and fill internal data structures based on the above format.
You must adjust the rules if your Polarion log files use a different format.
Here's the default set of regular expression rules for parsing standard log lines:

```
{
    "timestamp": "^(.*) \\[[a-zA-Z].* - ",
    "timestampFormat": "yyyy-MM-dd HH:mm:ss,SSS",
    "thread": "\\[(.*)\\] [A-Z]",
    "level": "\\] ([A-Z ]{5}).* -",
    "class": "\\] [A-Z ]{5} (.*) -",
    "exceptionClass": "^(\\b[\\w\\.]+\\b): ",
    "message": "- (.*)",
    "exceptionMessage": "^\\b[\\w\\.]+\\b: (.*)"
}
```

(Note: The backslash character must always be doubled.)

You must define all rules and all regular expression patterns. Most lines comply with the format above, but exceptions do not. There is only the exception class and message. (So there are the two `exceptionClass` and `exceptionMessage` rules.)

2.2 Analysis rules

This set of rules is the heart of the tool. It defines how the log file lines are understood, extracts relevant information, and puts it into an output feed. There are several types of rules, most of which are based on the message component of the log line.

Defined rule types:

- **MESSAGE** - A simple rule type to read simple single-line information like the Polarion version, build, or underlying operating system.
- **WHEN** - This is similar to a message-based rule to log when a specific action happens. (It prints the timestamp.)
- **DURATION** - A double-rule type that searches for the beginning and end of something and calculates the duration between them. (For example, the length of the indexing phase, the DB History creator, etc.)
- **COUNT** - A simple rule that counts the occurrences of lines matching a simple rule.
- **TXLOGGER** - A special rule to parse TXLOGGER entries. It understands the total length of an activity, and its results are sorted by length from the longest to the shortest.
- **WARN** - A rule that processes WARN level entries.
- **ERROR** - A rule that processes FATAL and ERROR level entries.
- **EXCEPTION** - A rule that processes special lines that only hold an exception with its message.

The tool contains a comprehensive set of initial rules that give you the most common and relevant information in the parsed log file but also allows you to build additional rules for extended functionality.

All rules are defined by their type (one of the above), their rule name (a single, free text description), one or more regular expressions for log lines, and potentially a hint. (That can help solve specific issues.)
The first four rule types are so-called single result rule types. The other types are multi-result rule types. The difference lies in some internal logic. It's expected that multi-result types may be found multiple times, and in the case of the `TXLOGGER` rule, results are sorted from longest to shortest, so you always see the longest operations. The multi-result rules can also define another `countLimit` field that defines how many occurrences you want to see in the output. The defined value overrides the internal default value of 10. (This might be useful for `TXLOGGER` rules.)

2.2.1 Extra rules

You can build your own JSON file with additional rules and put it as a parameter for analyzing your log files. The following example adds two simple rules that parse the Java runtime environment version and a list of extensions. They are part of an additional section (a new tab in the HTML output) titled "Extras".

```
{
    "Extras": [
        {
            "type": "MESSAGE",
            "name": "Java version",
            "messageRegEx": "^java.version = (.*)"
        }, {
            "type": "MESSAGE",
            "name": "Polarion extensions",
            "messageRegEx": "^Extensions: \\[(.*)\\]"
        }
    ]
}
```

You can add additional rules in the same section as other objects within the "Extras" JSON array or add sections as arrays into the global object / outer parentheses. You can also use the existing (default) section names like "Basics" and "Errors" and add additional, custom rules to them. If you add a rule with the same name as an existing rule in a section, the existing rule is overridden.

The following sections describe additional details and examples for each rule type.

2.2.2 MESSAGE

This is the simplest rule type. It uses the `messageRegEx` expression to match it with the message component of the log line and stores the first group found via the expression. It must contain a single group definition in the parentheses ( ) to extract the value for output. Suppose you have the following rule:

```
"type": "MESSAGE",
"name": "OS",
"messageRegEx": "^os\\.[a-z]* = (.*)"
```

And the following excerpt of the log file:

```
2022-11-21 00:04:48,700 [main] INFO
com.polarion.psvn.launcher.PolarionSVNApplication -
org.xsocket.connection.server.readbuffer.usedirect = true
```

```
2022-11-21 00:04:48,700 [main] INFO
com.polarion.psvn.launcher.PolarionSVNApplication - os.arch = amd64
2022-11-21 00:04:48,700 [main] INFO
com.polarion.psvn.launcher.PolarionSVNApplication - os.name = Linux
2022-11-21 00:04:48,700 [main] INFO
com.polarion.psvn.launcher.PolarionSVNApplication - os.version =
5.10.0-14-amd64
2022-11-21 00:04:48,700 [main] INFO
com.polarion.psvn.launcher.PolarionSVNApplication - osgi.arch = x86_64
```

The message part is always the part behind the class name and dash. In this example, the second, third, and fourth lines will match the abovementioned rule and take the first matched group. It's any string behind the equal sign so that the following results will be added for the above rules: `amd64`, `Linux` and `x86_64`. (This is what's written in the tool's text or HTML-based output.)

2.2.3 WHEN

This rule type shows the time stamp of a log entry defined by one of the following regular expression fields: `messageRegEx`, `classRegEx`, and `threadRegEx`. If there's a match for one of the expressions, the timestamp is printed into the output. Example:

```
"type": "WHEN",
"name": "Reindex phase 1 start",
"messageRegEx": "Platform startup \\(Phase: 1/9\\)"
```

2.2.4 DURATION

The duration type measures the time between two events identified in the log file. It calculates the time between the two lines' timestamps. This type needs to match the beginning of an event and the end of the same event. You can also define whether the first or last occurrence of a regular expression match should be used for the time calculation.

Example:

```
"type": "DURATION",
"name": "Reindex phase 1 took (calculated)",
"messageRegEx": "Platform startup \\(Phase: 1/9\\)",
"messageRegExEnd": "Summary for 'Platform startup'",
"durationStartTake": "FIRST",
"durationEndTake": "FIRST"
```

The standard `messageRegEx` field defines a regular expression for the message part of the beginning of the event, and `messageRegExEnd` defines the same for the end of the event. The additional fields define what occurrence of the found line should be taken for the beginning or end of the event. Possible values are `FIRST` or `LAST`.

2.2.5 COUNT

The count rule type tallies and displays the number of occurrences of a specific pattern. The pattern can be checked on the message, the class name, the thread name, and the log level string. The default rule set uses this type to count all FATAL, ERROR, WARN level messages and exceptions. For example, the errors are counted via the following rule:

```
"type": "COUNT",
"name": "Number of Errors",
"levelRegEx": "ERROR"
```

2.2.6 WARN and ERROR

This is the first of the so-called multi-result rules since there may be dozens, even thousands, of the same type of errors. Warnings are tagged as WARN in the log, and errors are either ERROR or FATAL. All lines with these levels are added to a respective group of warnings or errors. Rules also use a simple `messageRegEx` field for matching. For example:

```
"type": "ERROR",
"name": "Error in object",
"messageRegEx": "uri: subterra:"
```

Warning rules use type "WARN".

You can define rules for specific errors like the one above or define a generic rule that will include all remaining errors not belonging to any particular group. For example, the default rule set contains the following generic rule:

```
"type": "ERROR",
"name": "Other Errors"
```

As you can see there is no reg. ex. defined.

2.2.7 EXCEPTION

Exceptions are similar to errors but are logged differently. An Exception log line only consists of the exception class name and the exception message. (No timestamp, thread name, or level). You can define rules that check both fields via the `messageRegEx` or `classRegEx` fields. Example:

```
"type": "EXCEPTION",
"name": "SVN access file",
"messageRegEx": "Unable to read access file"
```

Any exception that doesn't match an explicit rule also falls into the generic group defined for an error without a regular expression field.

2.2.8 TXLOGGER

TXLOGGER records provide timing and memory consumption data for time-consuming operations. You can also use them to define individual rules for specific operations via `messageRegEx` and `threadRegEx` fields. Currently, there is only one predefined rule for the DB History Creator job and a generic one that collects all other lines.

The length of the operation sorts the result of each rule (if found in the line). The total time searching is made via the following regular expression (Hardcoded at the moment):

```
.*Total: (\\d+\\.?\\d*) s
```

3 HTML output

The tool primarily generates machine-readable output for additional automated analysis of Polarion's log files, but it can also generate a human-readable HTML report. The HTML report provides a quick and easy analysis of the logs. The tool generates a simple, single HTML file with the same results as the machine-readable text output. All results are split into separate tabs based on the categories defined in the rules. There are essentially two types of tabs:

1. Simple rule tabs like Basics and Reindex,
2. Multi-output rule tabs like Warnings, Errors, Exceptions, and TXLOGGER.

The simple rule tabs display a basic table of all rules and their values in the log files. Some of the values may also contain multiple occurrences. The tool displays all found occurrences.

The multi-output tabs display results grouped by: Warning, Error, Exception and TXLOGGER.

The Warnings, Errors and Exceptions tabs display a time distribution of item occurrences grouped by minutes. They display the time distribution from the first to the last minute logged by the log file and the number of item occurrences of a specific type for that minute.

The TXLOGGER chart is different. It displays each TXLOGGER item longer or equal to the threshold value (see the '-t' program argument above). This helps with performance analysis.