How To Use The Generic Log Parser

Introduction

Applications create a variety of log files, and sometimes the volume of data can be overwhelming and difficult to sort through in order to find issues or verify something is working correctly. It is common for parsing tools to be created to provide a method of removing or finding data from the files in order to analyze a particular issue. In most cases these log files follow a similar pattern that basically consists of some sort of header and then the actual data for the particular log message.

In order to prevent having a separate parsing tool for each type of log, the Generic Log Parser was created to provide a method of parsing almost any log file without having to have a new tool for each type of log file. In simple terms, you tell the Generic Log Parser which files it should parse, what format the header is in and what you want to search for in each item that is logged. When the parser is started, it takes the data out of the input file(s) and then outputs only those log items that match the identified inputs into the output file. The output file will hopefully then only contain information that is important for whatever issued is being investigated.

We Need Your Help

This document is pretty long and creating the initial header expressions and various parsing configurations can be difficult and confusing to create. As such, if you have a configuration that you believe is useful, please send us either your header expression or your saved .parseconfig file so that we can include the files as part of the Generic Log Parser release.

Also, if there are any improvements you would like or bugs that you find please let us know so that we can sort them out.

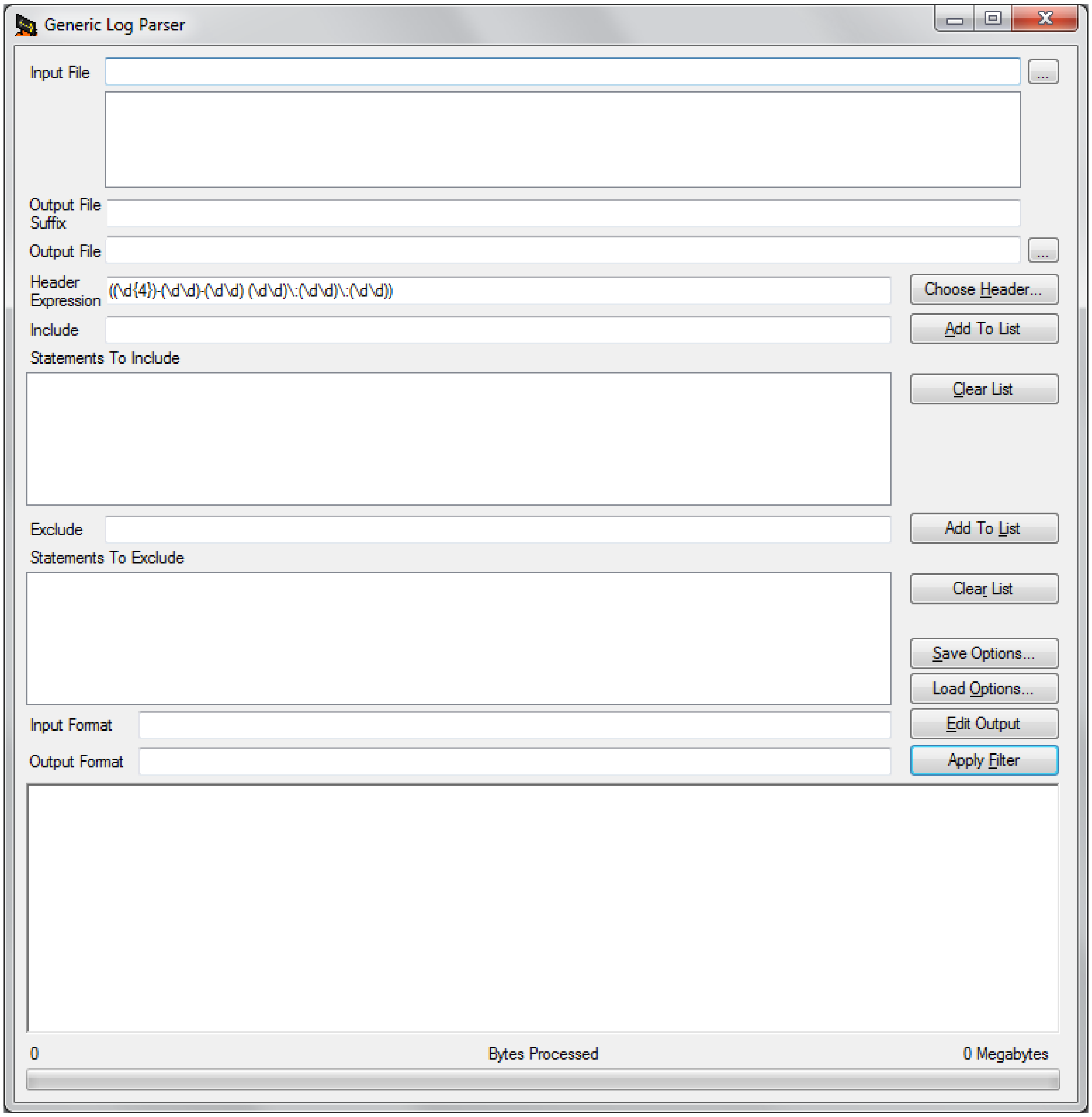
Operation

Running the executable

The Generic Log Parser is run using the GenericLogParser.exe. Configuration settings are kept in the GenricLogParser.exe.config and then .parseconfig files are used particular parsing settings.

The executable is a .Net app so the .config file is a standard .Net application config file that will most likely not be edited by a user. The .parseconfig files will be explained in more detail later, but they essentially allow the user to save a particular set of settings, as it is quite common to want to perform the same operations on multiple sets of log files and configuring some of the options is not particularly easy the first time.

When starting the GenericLogParaser.exe you will see the following screen:


Header Expression: ((\d{4})-(\d\d)-(\d\d) (\d\d)\:(\d\d)\:(\d\d))

0 Bytes Processed 0 Megabytes

Quick Tutorial

What follows this section is a very in depth discussion of the various parts of the screen. This section is a quick overview of using the screen.

- Use the "…" button to select your input file(s)
- Add an "Output File Suffix" to make what's in your output file more clear if you want to.
- Use the "Choose Header…" button to select what format of the header of the log files you are going to be processing.
- Enter in the Include and Exclude values you are concerned about with the logs.
- Click "Apply Filter"

Short Field Descriptions

The individual fields are explained in more detail in the "Using the Parser" section below but here is a quick description of each field:

- Input File (and the list box below it) - Used to show which log file(s) are going to be processed.
- Output File Suffix - a suffix that is put on the end of the name of the output file to help the user identify what data a particular output file contains (e.g. DeviceMessages, AppMessages, MainThread, etc.)
- Header Expression - This is a regular expression that indicates what the header for a log message looks like.
- Choose Header… button - Allows the user to choose a saved header expression and also has a tool for creating a new expression.
- Include (along with Statements To Include and the Add To List button) - these are used for specifying what data you would actually parse out of the input and put into the output.
- Exclude (along with Statements To Exclude and Add To List button) - these are similar to the Include statements except for they are records that you would like to exclude from the output.
- Input Format - a regular expression that indicates what the input data looks like. Is used with Output Format and is similar to Header Expression but doesn't always have to be specified.
- Output Format - a regular expression that indicates what the output data should look like. Used with Input Format it allows input data that is in one format to be output in a different format. Not always used.
- Save Options… button - once a user has a set of options that they like and want to save, this button can be used to save those options into a .parseconfig file.
- Load Options… button - used to load options that have been saved in .parseconfig file into the appropriate fields.
- Edit Output button - will open the output file in Notepad.
- Apply Filter button - this causes the actual process of taking the input data and putting it into the output file to occur using the parameters that were specified.
- Large text box and progress bar at the bottom of the screen - these are used when the Apply Filter button is clicked to show the user the progress of the parsing.

Using the Parser

Selecting/Creating the Header Expression

To use the parser, the first thing you need to do is select a header expression or create a new one. The header expression will generally be the same for all log files that are generated by the same product. The header expression is a regular expression that shows the format of what the header of a log message looks like. Regular expressions use tokens to indicate what type of data should show up in a particular place in a string. They are not particularly user friendly, but luckily the Generic Log Parser lets you save your configurations so that you generally will only have to figure things out once and then you can just select the value from a list. You also don't need to specify the entire header expression, generally just enough of the beginning of it to identify that a new log message has started. This is probably easier to understand with an example. Here is a small amount of output from a sample log file:

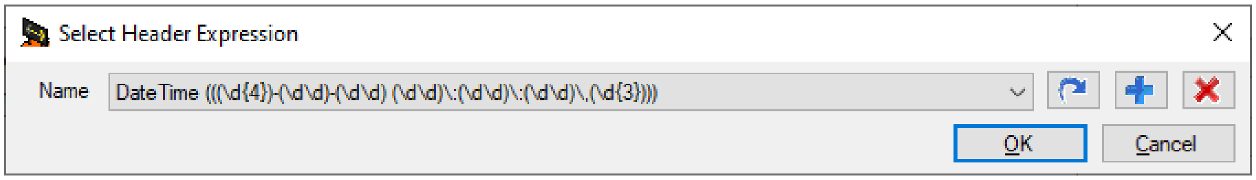
```
2016-03-29 03:36:57.906 (297) - DEBUG->RoundRobinLogging (Thread WinMain (1800)) - Line: 42 -
File: RoundRobinLogging.cpp
Will be logging using the queue and the MessageLoggingThread

2016-03-29 03:36:57.906 (297) - DEBUG->RoundRobinLogging (Thread WinMain (1800)) - Line: 43 -
File: RoundRobinLogging.cpp
Found 75 files on init and will be using file boot index 61

2016-03-29 03:36:57.906 (297) - SYSTEM->LoggerBase (Thread WinMain (1800)) - Line: 91 - File:
LoggerBase.cpp
RoundRobinLogging: Setting 0 Active Logging Groups
```

For the above data each log message has a header that starts with the date and time followed by a tick count, then what the type of log message is (DEBUG, ERROR, etc.), what "logging group" the message is from, the thread the message is being logged from and then the line and file where the message is being logged from. That is a lot to figure out the format of. For the most part, though you can just worry about what is at the beginning and that will be enough for the parser to identify the message. In this example, the time and date part of the header should be enough to tell the parser that a new message has started.

The date and time parts of these log headers will always be in the form "nnnn-nn-nn nn:nn:nn.nnn" with the n's representing a digit 0-9 so we'll want to create a regular expression to represent that.

NOTE: A guide to what tokens are valid in a regular expression can be found here: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference

It is possible to just type the header expression in but that isn't really recommended. Instead, click on the "Choose Header..." button and it will bring up a small screen:

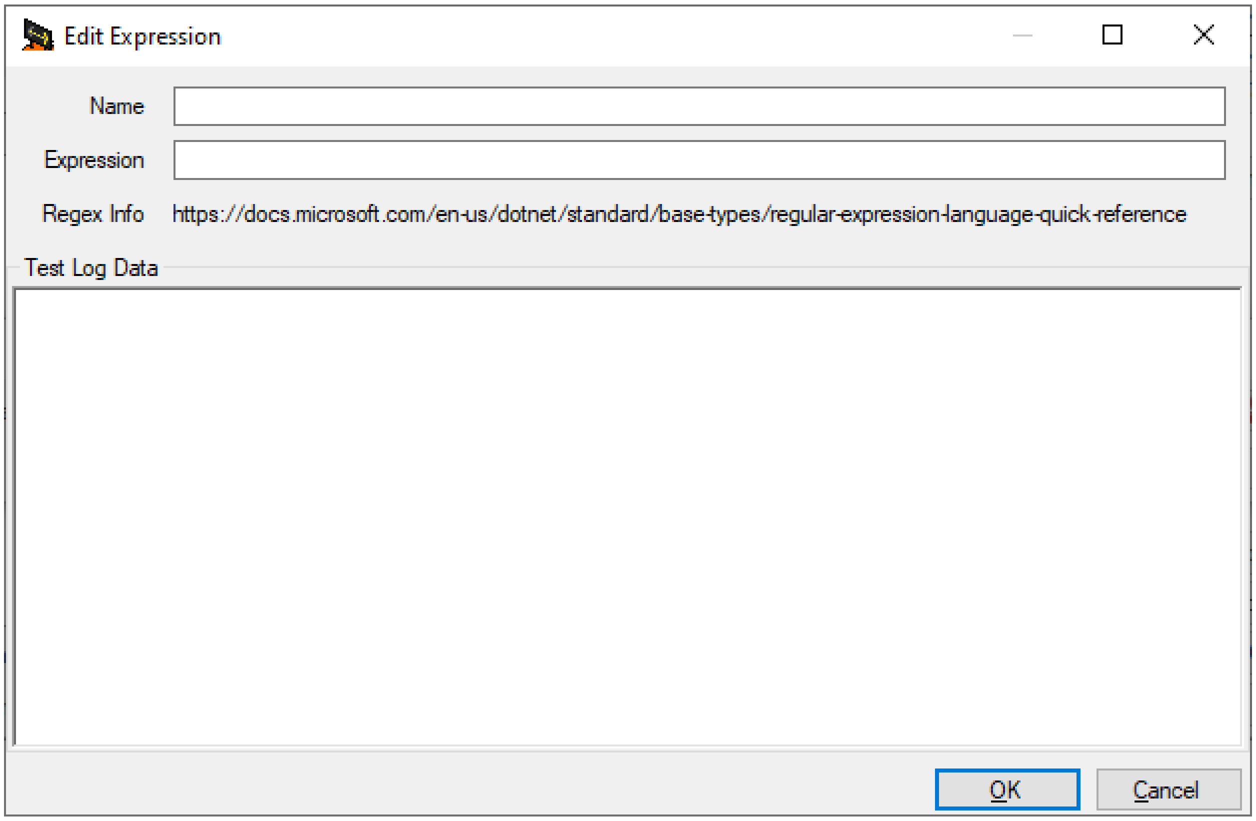

Select Header Expression

Name: DateTime (((\d{4})-(\d\d)-(\d\d) (\d\d)\:(\d\d)\:(\d\d)\.(\d{3})))

If there is already an expression for what you are trying to parse then go ahead and select that in the combo box and hit OK. Otherwise, you can edit your own expression. The icon buttons mean the following:

- Arrow - Edit the expression currently in the combo box.
- Plus - Create a new expression
- Red X - Remove the expression currently in the combo box.

To create a new expression click on the Plus icon and you will see this screen:

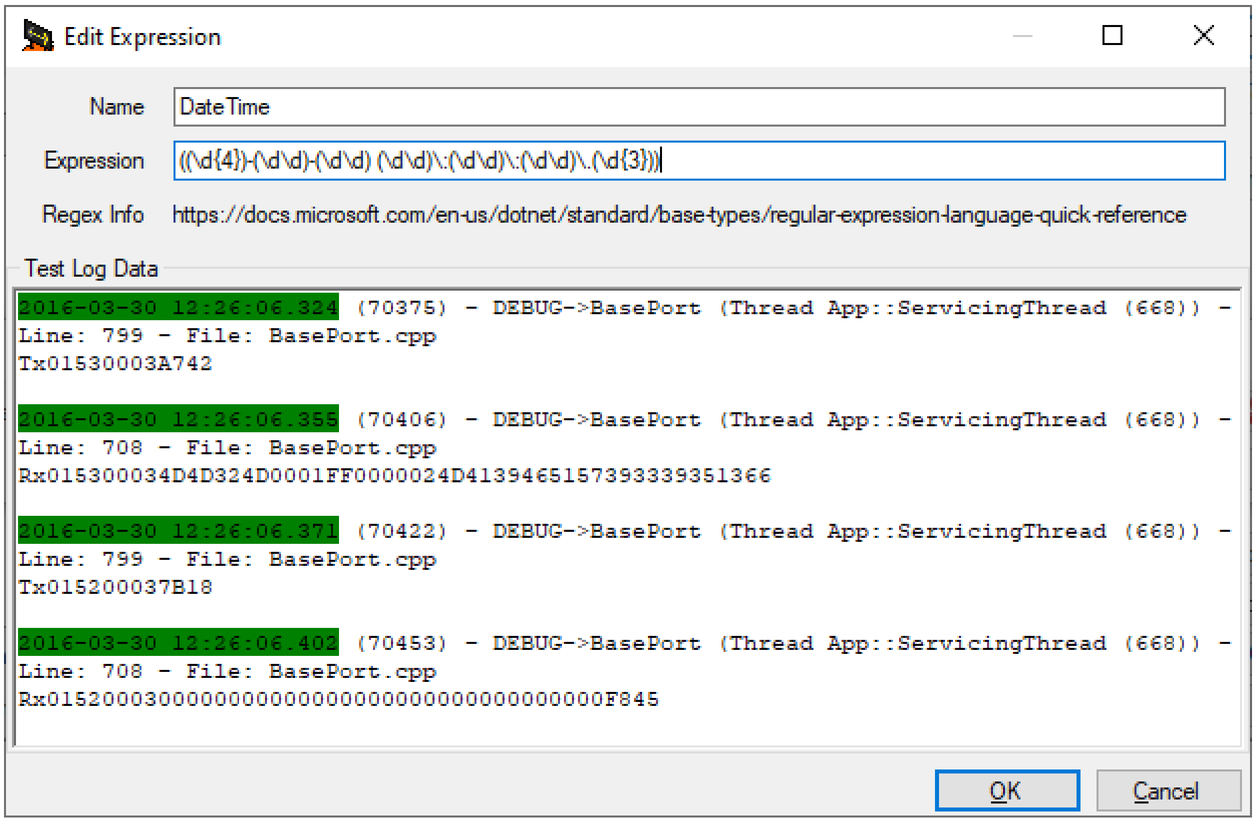


Regex Info https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference

Basically what you want to do is give your expression a name (such as Our Device Logs) and then you will actually type the expression in the expression text box. The large "Test Log Data" entry field is there to help you with making sure your expression is indicating what you want it to. It is way beyond the scope of this document to give a regular expressions tutorial but this example, here's mainly what you need to know.

- Most characters by themselves indicate that that particular character should appear in that position in the string. So if there is a dash (-) or a colon (:) that means you actually expect that character to be in the string at that position.
- \ is one of the special characters
- \d indicates you expect there to be a number (0-9) in that position in the string.
- {n} after a character placeholder like \d means you expect there to be n of that character in the string. So \d{4} means you expect there to be four numbers in a row.

Based on that, for our "nnnn-nn-nn nn:nn:nn.nnn" string we would expect "\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}.\d{3}" to work. We can use the Edit Expression screen to double check. Put the expression in the Expression field and our sample messages in the "Test Log Data" field. The parts of the test log data that match the expression should be highlighted in green as shown below.



Name: DateTime

Expression: ((\d{4})-(\d\d)-(\d\d) (\d\d)\:(\d\d)\:(\d\d)\.(\d{3}))

Regex Info: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference

Test Log Data

```
2016-03-30 12:26:06.324 (70375) - DEBUG->BasePort (Thread App::ServicingThread (668)) -
Line: 799 - File: BasePort.cpp
Tx01530003A742

2016-03-30 12:26:06.355 (70406) - DEBUG->BasePort (Thread App::ServicingThread (668)) -
Line: 708 - File: BasePort.cpp
Rx015300034D4D324D0001FF0000024D4139465157393339351366

2016-03-30 12:26:06.371 (70422) - DEBUG->BasePort (Thread App::ServicingThread (668)) -
Line: 799 - File: BasePort.cpp
Tx015200037B18

2016-03-30 12:26:06.402 (70453) - DEBUG->BasePort (Thread App::ServicingThread (668)) -
Line: 708 - File: BasePort.cpp
Rx01520003000000000000000000000000000000000F845
```

So if you use the Expression above as the Header expression, when parsing the log files everything between the start of one green section to the start of the next green section will be considered one log message. The Generic Log Parser searches for a header expression and when it finds one it searches for the next header expression. Then it takes the string in between and checks to see if it matches up with the Include and Exclude statements to decide if the message should be included in the output file.

Hit OK when you are happy with your expression. Then make sure it is selected in the combo box on the previous screen and hit OK again. The expression should now appear in the Header Expression text box.

Note: The header expressions you create are saved in the GenericLogParser.exe.config for the current user.

Selecting the Input Files

Now that we have a header expression we need to select the files that we want to parse the data out of. This is done easily enough by using the "..." button to the right of the Input File text box and selecting the file or files that you want to include as part of your parsing. Some notes:

- Currently every time you select some files from a directory they will replace what is already in the input files list box so if you want to work with more than one file at the same time they need to be in the same directory and you need to select them all at the same time.
- If you select a file you didn't mean to you can just highlight it in the input files list box and hit the Delete button.

- The selected input files will be processed in alphabetical order so it is recommended that if you want to process multiple files that their time order matches their alphabetical order.

Output File Suffix and Output File

When you select input files the Output File will automatically be changed to be the name and directory of the first file with Parsed appended to the name, so E:\Temp\SomeLogFile.txt becomes E:\Temp\SomeLogFileParsed.txt.

If you are only going to need to parse things once that should be fine and you can leave the "Output File Suffix" blank. However, if you are going to do multiple parsings (e.g. one for incoming messages and one for outgoing messages) you can add an additional suffix in the "Output File Suffix" text box and that will be added to the end of the output file name as well. So if you enter "IncomingMessages" as the "Output File Suffix", E:\Temp\SomeLogFileParsed.txt will become E:\Temp\SomeLogFileParsedIncomingMessages.txt.

You can also manually edit the output file name but as a general rule that shouldn't be necessary.

Include and Exclude Statements

This is where you get to indicate what you actually want in your output. Once the log parser has found a log message it will look to see if any of the expressions that are part of the values you have put in the Exclude list match the log message. If so, then that message will not be put in the output. If the log message has not been rejected then the log parser looks at the expressions in the Include list and sees if the log message matches with any of them. If so then the log message will be included in the output and the parser moves on to the next log message.

A couple tips about the Include and Exclude fields:

- If you just want to search for one thing you can just type it in the Include or Exclude text box as appropriate and then apply the filtering. If you want to Include or Exclude multiple things then after you type the value in the text box you need to click the "Add To List" button for that text box so it is included in the list box. Then you can type your next value. Make sure you click "Add To List" for the last item you type as well.
- The values in the Include and Exclude list are treated as regular expressions but most of the time you can just put some text you want to search for and that will be fine.
- Because the expressions you are including and excluding are treated as regular expressions they are case sensitive so be careful with that.
- If you have multiple expressions to include or exclude the parser only needs one to match to include or exclude the message.

Input Format and Output Format

This is an advanced topic that you will only need if you want to reformat the output of the log messages to something different. If you don't need to reformat the output, you can skip this section. An example of when you might want to reformat the output is if you have a lot that has some sort of messages in it and you want to make it so that the messages line up better.

For example, let's say your original log file input looks like:

```
2016-03-30 12:26:06.324 (70375) - DEBUG->BasePort (Thread App::ServicingThread (668)) - Line: 799
- File: BasePort.cpp
Tx01530003A742

2016-03-30 12:26:06.355 (70406) - DEBUG->BasePort (Thread App::ServicingThread (668)) - Line: 708
- File: BasePort.cpp
Rx015300034D4D324D0001FF0000024D4139465157393339351366

2016-03-30 12:26:06.371 (70422) - DEBUG->BasePort (Thread App::ServicingThread (668)) - Line: 799
- File: BasePort.cpp
Tx015200037B18

2016-03-30 12:26:06.402 (70453) - DEBUG->BasePort (Thread App::ServicingThread (668)) - Line: 708
- File: BasePort.cpp
Rx01520003000000000000000000000000000000000F845
```

You can use the input and output format to not only parse the output but reformat it so that maybe it looks something like the following:

```
70375: 2016-03-30 12:26:06.324 - Tx01530003A742
70406: 2016-03-30 12:26:06.355 - Rx015300034D4D324D0001FF0000024D4139465157393339351366
70422: 2016-03-30 12:26:06.371 - Tx015200037B18
70453: 2016-03-30 12:26:06.402 - Rx01520003000000000000000000000000000000000F845
```

As before this requires regular expressions and the easiest way to explain things is by using an example. Unfortunately the screen doesn't have a specific way to test your expression but you can cheat and use the Header Expression editor. Note that the Header Expression editor has some issues with \r and \n. The log file might have \r\n but when you copy into the "Test Log Data" text box it gets converted to \n.

A standard log message from an application might look like the following:

```
2016-03-30 12:26:06.371 (70422) - DEBUG->BasePort (Thread App::ServicingThread (668)) - Line: 799
- File: BasePort.cpp
Tx015200037B18
```

We can separate it into the following parts. The separates such as spaces, dashes and arrows have been left out but they need to be part of the expression when it is built.

- Beginning of line
- Date
- Time including milliseconds
- Tick Count since application started (surrounded by parenthesis)
- Log Message Type (DEBUG, ERROR, etc.)
- Logging group
- Thread Name
- Thread Number (in parenthesis) (and Thread Name and Number in additional parenthesis)
- Line Number
- File Name
- The actual log message

Using the "?<NameOfItem> syntax you can create an expression that essentially labels the different parts of the message:

```
^(?<Date>(\d{4})-(\d{2})-(\d{2})) (?<Time>(\d\d):(\d\d):(\d\d).(\d{3})) \((?<TickCount>(\d+))\) - (?<LogType>\w+)->(?<DebugGroup>\w+) \(Thread (?<ThreadName>.+) \((?<ThreadNumber>\d+)\)\) - Line: (?<LineNumber>\d+) - File: (?<FileName>[^\r\n]+)([\r\n]+)(?<RestOfIt>.*)\r\n
```

Here's some tips on what that regular expression actually means

- The carat (^) indicates the start of a line
- Items in parenthesis indicate a grouping so if you actually are trying to say that there is a parenthesis in the string you have to put a backslash in front of it, i.e. \(and \)
- \w means a "word character" which is essentially A-Z and the numbers
- + means that there can be 1 or more of the preceding thing being matched (like \w)
- . means any character
- * means 0 or more of the preceding thing being matched.

So now that we have defined what our input is going to look like (which should be put in the Input Format box, now we need to define what our Output Format should be.

To indicate you want to put one of the items that was labeled in the input you use $(Name) and then the rest can just be characters. So for our example the Output Format would look like the following:

```
${TickCount}: ${Date} ${Time} - ${RestOfIt}
```

which should be reasonably straightforward.

Apply Filter

Now that you've done all that work, click the "Apply Filter" button and the input files will be processed one by one and matching records will be put into the output file. You will be able to see the progress in the text box at the bottom of the screen as well as on the progress bar at the very bottom of the screen.

One thing to note is that if it seems like nothing is happening this is a good indication that your header expression has something wrong with it so you should double check that.

Save Options and Load Options

Since a user is quite likely to have some complicated sets of options they use for various logs, along with some difficult to create and remember regular expressions, the screen provides a method for saving the current settings to a file and then allowing them to be loaded later. Not only does this let you save your hard work, if someone else has already done the work to figure out how to parse a particular file they can just send you their .parseconfig file and you can load that file.

To save your current options, click on the "Save Options…" button choose an output file and then select the individual parts of your current options that you want to save. Unless you actually change the extension the options are saved in a .parseconfig file.

Similarly, if you want to load some options that you have previously saved, click on the "Load Options…" button and select your file and the options you have saved will be filled in on the screen.

TODO List

These are items that we are already in the process of fixing or updating.

- Expression builder for Input and Output Formats
- Multiple output files, based both on a file to file matchup and size limitations
- Ability to specify a different editor when the "Edit Output" button is clicked.
- Input files from multiple directories.
- Case insensitive searches
- There might be a bug that prevents the last record in a log from being processed.
- Remember the last input file directory separately from the .parseconfig directory.