uniqFEED®

Hi,

We are happy that you are taking part in our technical assessment which is the first part of the interview stage for our open engineering positions.

As previously explained, you have a time frame of up to 24 hours to complete this test. The test comprises two parts. Part one consists of some tasks that you can solve in a mock git repository project. Part two consists of answering questions we have for you in written form.

How to complete this test

Time cap

- We estimate the assessment to take you 3-5 hours
- The maximum time you have to complete the test is 24 hours beginning with the moment we send you the instructions. Please make sure you have handed in your answers to both parts of the test before the time expires or else we will assume you choose not to hand in any results.

Read before you write

- Each task or question has a different complexity so take your time and study the instructions and questions carefully before you begin. Make sure you have understood all details properly to prevent moving in the wrong direction.

Instructions Part One

- Clone the repository we have sent you a separate invite to
- Create a branch to implement your changes on
- Choose a minimum of 2 tasks to work on, optionally you can choose to solve as many more as you want
- Once you are done, commit your code changes to the source repository
    - Tip: commit the changes to individual tasks separately to make it easier to review for us J , add clear commit messages
- Push your changes to remote and create a pull request from your branch to master for us to review your changes. We will assess your results based on the pull request so don't forget to create one!
    - See also: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/creating-a-pull-request

Instructions Part Two

- Please answer the questions in written form within a text file.
- Either commit that file to the sources in git, name it "Answers Part Two.txt" or so
- OR append that text file to an email with the title: "HR-186780 Assessment, Answers Part Two" and send it to *stefan.dunca@uniqfeed.com*

What if something goes wrong?

- In case of emergency you can contact Stefan Dunca via email: *stefan.dunca@uniqfeed.com*
- We will reply as soon as possible to your notification, we are on standby the day of your assessment

Now, let's get going. We wish you the best of success for your assessment!

Sincerely,
uniqFEED Team

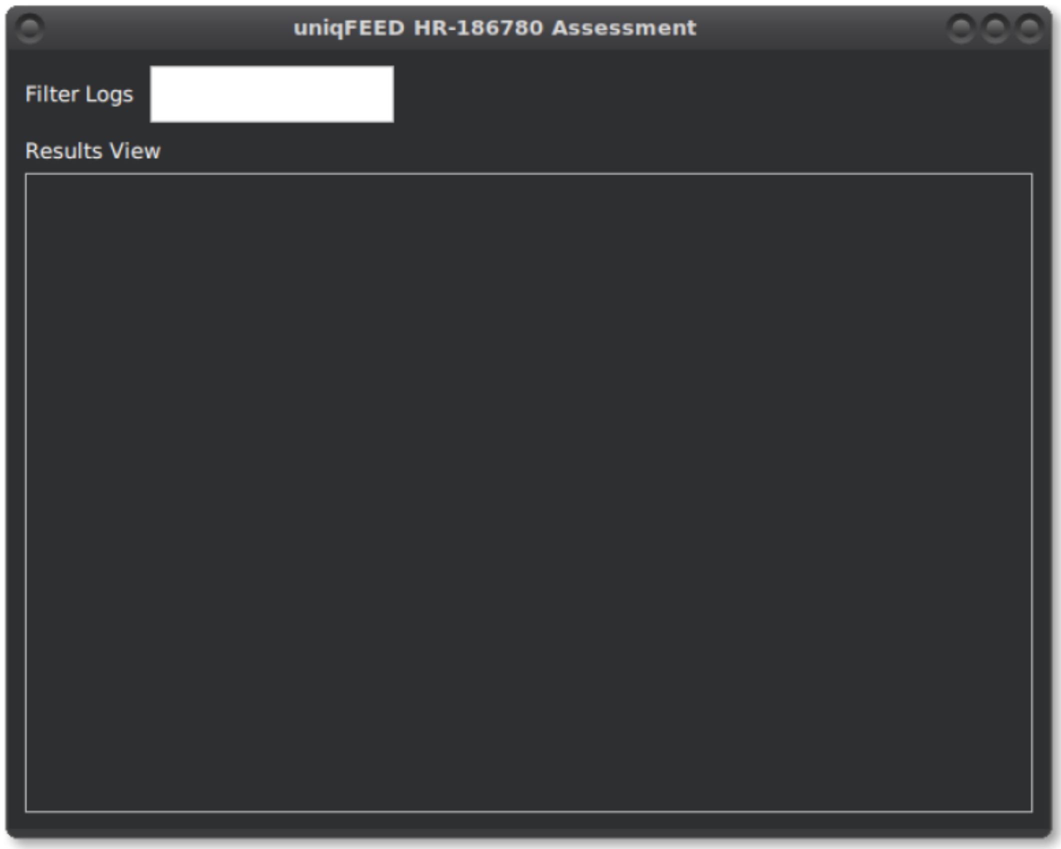
uniqFEED®

Part One –Log Message Interface

Your task is to write a system that assists users in parsing the log files of a complex pretend system. The system should contain both a backend providing the business logic and a frontend consisting of a user interface for operators to use.

Setting Up

- Ensure you have checked out the source code and created your development branch
- Compile and run the source code, it should compile and when you run the program a UI should appear that looks somewhat like this one



The UI

- The UI is very simple: it consists of a single text field to enter filter search terms into. Try entering a random term and hitting enter. You should see a log message being thrown to the console showing that your input has been accepted

- Beneath the search section there is a group box reserving some room to display results in. This is where we want you to add your UI innovations J
- All code for the UI is written in QML, located in main.qml and is part of the main Qt resource archive qml.qrc

The Backend

- The code in main.cpp is a skeleton structure to bring up the UI. We have already added the class LogSystemController for you to use as a starting point for implementing the business logic. You may of course replace that class if you want.

Coding Guidelines

- All your code can be added to the existing source files or you may decide to add new source files as needed, this is entirely up to you
- Please make sure you you write clean and legible code, use comments wherever you deem them useful

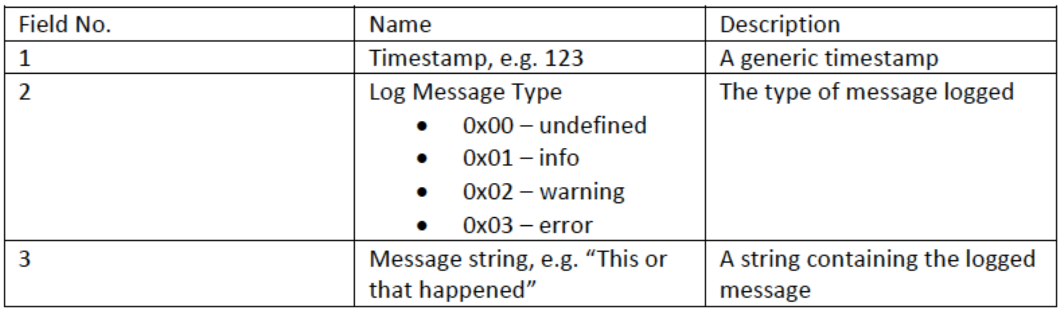
The Buildsystem

- The buildsystem used is qmake, if you add source files ensure that you make the corresponding changes in the .pro project file

The Log Format

- The log file needed to be parsed is located in the root of the source repository and named “logs.txt”.
- Each line in the log consists of up to three comma-separated parts and could like this:

```
154,0x01,"Established handshake with master server"
```



Field No.	Name	Description
1	Timestamp, e.g. 123	A generic timestamp
2	Log Message Type • 0x00 – undefined • 0x01 – info • 0x02 – warning • 0x03 – error	The type of message logged
3	Message string, e.g. “This or that happened”	A string containing the logged message

If a line in the file contains only a “.” It translates to that the system shutdown. You can use this log message or ignore it.

Task Backlog

Following table shows the tasks we want you to work on. We want you to work on at least 2 tasks. If you want you can decide to choose more than two tasks, though this is not expected of you. Task 1 is mandatory and expected to be done first.

Task	Mandatory	Description
1 – Parsing	Yes	Parse the log file logs.txt into a data model. Use the log format description above as a foundation on how to structure your model data. Output status information on the console or optionally in a status bar within the main application window, as it is important that the user gets some feedback on if the parsing worked or not.
2 – Log Message Display	No	Display all log content in a structured way, e.g. using a table format or a list. Ensure the user can access all log output by enabling scrolling. Optionally make it possible for the user to sort the output by timestamp or message type.
3 – Filtering	No	Use the Filter Logs input field to enable a user to filter the log messages. For example a user might want to filter according to a certain timestamp, error type, or log message substring. Present the results in a structured way, e.g. using a table format or a list.
4 – Statistics	No	This is a creative UI task, create a UI element that can display on of the following statistics: • Timeline: Over time, when were messages logged, i.e. in a two dimensional timeline graph • Amounts: Display the amount of logged messages by message type, i.e. in a bar graph or pie chart

Notes

- Tasks 2-4 each get slightly more complex, so consider this when you choose what task you tackle
- You may use any help you need, i.e. the internet, calling a friend to discuss ideas (as long as it isn't you friend coding these implementations ;-))
- The rules are there are no rules, please get creative with these tasks, we do not expect a certain solution, we rather want to see how you approach solving these problems
- Feel free to change the existing code, you can change anything as long as it satisfies the requirements above
- Please make sure your code compiles without errors! This will be part of the grading.

uniqFEED®

Part Two –Thread Synchronization

Our code has to accommodate several threads running at the same time. Please answer the following questions in written form and hand in your answers as described above.

You have N execution threads that call the same functions in sequence in the same order: func1(); func2(). You want all the threads to complete the execution of func1() before they call func2().

- Implement a sync mechanism for this
- What primitives would you use?
- Write an implementation in a language of your choice or pseudo-code
- Can you make it re-entrant?
- Discuss corner cases