ShowCPU - Documentation

1. Introduction and how to start

ShowCPU is an application programmed in the course of a "homework assessment" and accompanies the interview as a software engineer in the UI Software Development Team at Daimler AG, Sindelfingen, Germany. The major task of *ShowCPU* is to provide information on the CPU within a Linux system by reading */proc/cpuinfo* and subsequently displaying the data in a graphical user interface (GUI). *ShowCPU* is capable of reading the information for each CPU socket and logical core. At this stage of development, only the 1st socket's information is displayed in the GUI, which is sufficient for most systems. However, *ShowCPU* is programmed to allow further development towards displaying information for each specific CPU socket on machines with multiple CPUs such as high-performance clusters or servers. The *ShowCPU* application was developed on Ubuntu 16.04.03 LTS using the open-source version of Qt Creator 4.4.0, Qt Quick 5.9, GCC 5.3.1 and C++11.

To install and run *ShowCPU* (tested on Ubuntu 16.04.03), please follow the instructions below:

1) **Install Qt from terminal:**
   - wget http://download.qt.io/official_releases/qt/5.9/5.9.1/qt-opensource-linux-x64-5.9.1.run
   - chmod +x qt-opensource-linux-x64-5.9.1.run
   - ./qt-opensource-linux-x64-5.9.1.run

   More information on installing Qt is provided on https://wiki.qt.io/Install_Qt_5_on_Ubuntu.

2) **Download the *ShowCPU* source code from GitHub:**
   https://github.com/fsteinbach88/showcpu
3) **Build and run *ShowCPU* on your system:**
   Open the "*ShowCPU.proc*" project file in Qt Creator. Click Build → Run or hit Ctrl+r, the application will start. Click on "Load CPU Information" to read /proc/cpuinfo and display the data. Moreover, an executable is created in the source-code directory. You can paste or move it locally to run the application again.

In what follows, a more specific documentation of the source-code basics is provided for further development. An overview of all source-code files is shown in Fig.1. A basic flow-chart of the workflow of *ShowCPU* is provided in Fig. 2.

2. Information for developers

2.1 Backend

The source-code of *ShowCPU* is structured into three C++ files (excluding headers) and five important QML files (Fig. 1). The *main.cpp* file contains the main-function that calls the QML application. Furthermore, it contains basic functions to read and store the data from */proc/cpuinfo*. Two basic classes in *ShowCPU* facilitate data handling *CPUInfo* and *GUIBackend*: The information in *cpuinfo* is stored in the class *CPUInfo*, which is declared and defined in *cpuinfoclass_header.h* and *cpuinfoclass.cpp*, respectively. The file *guibackend.cpp* defines a class called *GUIBackEnd*, which allows communication between the information in *CPUInfo* and the GUI (see 2.2 Frontend). Distributing the CPU information in two classes may become useful for structuring and further development. The topics below provide information on the *CPUInfo* class and on how *cpuinfo* is read in general:

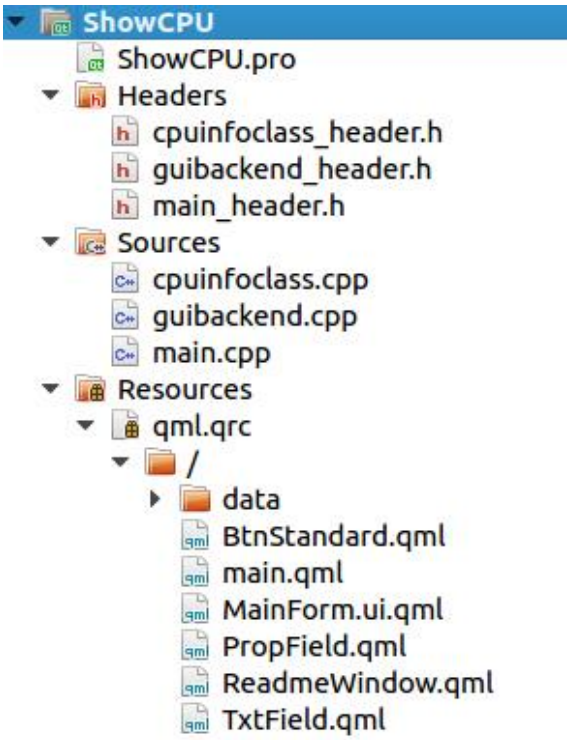

Figure 1: Overview of source-code files of ShowCPU

The CPUInfo class

The *CPUInfo* class mainly consists of public variables of type std::vector<…>. Each element in the vectors represents a specific logical core. This sorting is done as also *cpuinfo* distributes the CPU information sorted for each logical core. Each instance of the *CPUInfo* class represents one CPU socket. Additionally, some methods are programmed for *CPUInfo* that help assessing the most important information. More of these methods may be programmed during future development.

Reading /proc/cpuinfo

Reading of *cpuinfo* mainly happens in the function *ReadFile*. This function returns a vector<CPUInfo>, where each vector element represents on instance of *CPUInfo*, hence, represents on CPU socket (i.e. vector.size() will usually be 1 on most systems that are no servers or high-performance clusters). As *cpuinfo* is structured into blocks that are separated by an empty line and represent individual logical cores, *ReadFile* reads block by block. Reading a block starts with reading the actual “socket ID” using the *GetSocketID* function. Using the “socket ID”, the information is afterwards sorted in the correct instance of *CPUInfo* representing this socket.

For each block or logical core, the *ReadFile* function separates the name of a property of your CPU from the actual "value" this property has (e.g. a name is "cpu MHz", which can have the actual property value of "2900"). Line by line, each name is detected and assigned to the corresponding variable in the correct instance of *CPUInfo* by the *AssignProp* function.

2.2 Frontend

The GUI is programmed as Qt Quick application in Qt Creator 4.4.0, Qt version is 5.9.1. The majority is programmed in *main.qml*. In *main.qml* an instance of the *GUIBackEnd* class is created to communicate with the backend. Below, a short summary on this class is provided.

GUIBackEnd is declared and defined in *guibackend_header.h* and *guibackend.cpp*, respectively. It basically contains public methods to be accessed from *main.qml*. Two private variables exist that store the CPU information as returned from the backend-function *ReadFile* (*std::vector<CPUInfo> m_ALLCPUINFO*) and tell if the data was loaded successfully (*bool m_bDataLoaded*). The method *readCPUInformation* calls *ReadFile* and assigns its returned *std::vector<CPUInfo>* to *m_ALLCPUInfo*. The remaining public methods are "get" functions to read *m_ALLCPUInfo* and return a *QString* to be displayed in the GUI.

QML Items exist for (1) buttons, (2) the grey main-field behind the CPU information (item created to allow more of these fields in the future) and (3) the fields containing the CPU information. The items are programmed in (1) *BtnStandard.qml*, (2) *TxtField.qml* and (3) *PropField.qml*. Upon clicking the button labelled "Load CPU Information" (*id: startbtn*), the animations start. Once the animations are finished, the code executes *GUIBackEnd::readCPUInformation* and checks if the data was loaded successfully using *GUIBackEnd::dataLoaded*. If *dataLoaded* returns *true*, the fields of the *PropField* item are filled with the corresponding information using the abovementioned methods in *GUIBackEnd*. The workflow of *CPUInfo* is summarized in Fig.2 (following page).

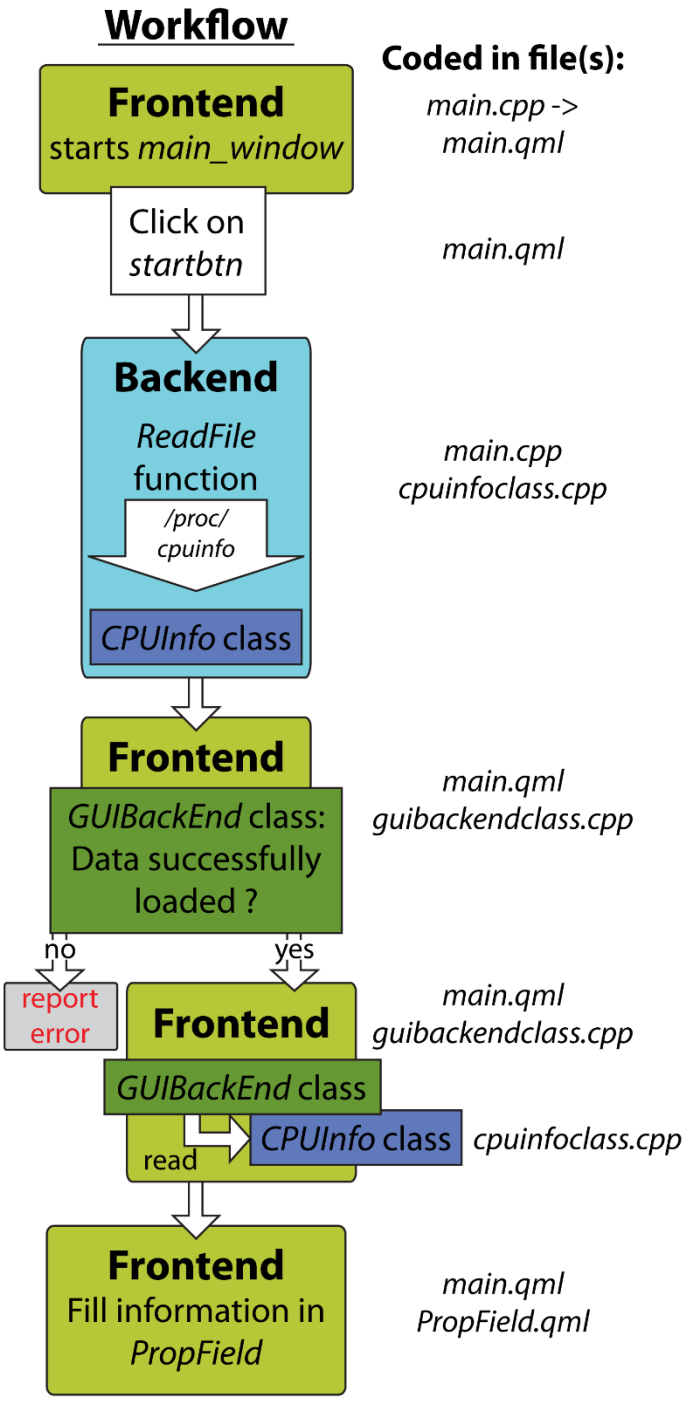

Figure 2: The workflow of ShowCPU. The right hand side indicates in which source-code files the corresponding action is programmed.