Linux Energy Monitor Application

1.0

Generated by Doxygen 1.9.7

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 raymii::Command Class Reference

Static Public Member Functions

- static CommandResult exec (const std::string &command)
- static CommandResult **execFgets** (const std::string &command)

4.1.1 Member Function Documentation

4.1.1.1 exec()

```
static CommandResult raymii::Command::exec (
            const std::string & command )  [inline], [static]
```

Execute system command and get STDOUT result. Regular system() only gives back exit status, this gives back output as well.

Parameters

command	system command to execute

Returns

commandResult containing STDOUT (not stderr) output & exitstatus of command. Empty if command failed (or has no output). If you want stderr, use shell redirection (2&>1).

The documentation for this class was generated from the following file:

- include/command.h

4.2 raymii::CommandResult Struct Reference

Public Member Functions

- bool **operator==** (const CommandResult &rhs) const
- bool **operator!=** (const CommandResult &rhs) const

Public Attributes

- std::string **output**
- int **exitstatus**

Friends

- std::ostream & **operator**<< (std::ostream &os, const CommandResult &result)

The documentation for this struct was generated from the following file:

- include/command.h

4.3 DateTime Class Reference

```
#include <date_time.h>
```

Static Public Member Functions

- static int Year ()
- static int Month ()
- static int Day ()
- static int Hour ()
- static int Min ()
- static int Sec ()
- static std::string CurrentDate ()
- static std::string CurrentTime ()

4.3.1 Detailed Description

This class represents the current date and time.

4.3.2 Member Function Documentation

4.3.2.1 CurrentDate()

```
static std::string DateTime::CurrentDate ( )  [static]
```

Getter method for current date.

Returns

The formatted current date.

4.3.2.2 CurrentTime()

```
static std::string DateTime::CurrentTime ( )  [static]
```

Getter method for current time.

Returns

The formatted current time.

4.3.2.3 Day()

```
static int DateTime::Day ( )  [static]
```

Getter method for current day.

Returns

The current day.

4.3.2.4 Hour()

```
static int DateTime::Hour ( )  [static]
```

Getter method for current hour.

Returns

The current hour.

4.3.2.5 Min()

```
static int DateTime::Min ( )  [static]
```

Getter method for current minute.

Returns

The current minute.

4.3.2.6 Month()

```
static int DateTime::Month ( )  [static]
```

Getter method for current month.

Returns

The current month.

4.3.2.7 Sec()

```
static int DateTime::Sec ( )  [static]
```

Getter method for current second.

Returns

The current second.

4.3.2.8 Year()

```
static int DateTime::Year ( )  [static]
```

Getter method for current year.

Returns

The current year.

The documentation for this class was generated from the following file:

- include/date_time.h

4.4 Format Class Reference

```
#include <format.h>
```

Static Public Member Functions

- static std::string ElapsedTime (long times)
- static std::string Date (int year, int month, int day)
- static std::string Time (int hour, int min, int sec)
- static std::string Decimal (double value, int precision)
- static std::string Percentage (double percent)

4.4.1 Detailed Description

This class helps format data of various types.

4.4.2 Member Function Documentation

4.4.2.1 Date()

```
static std::string Format::Date (
             int year,
             int month,
             int day )  [static]
```

Get formatted date given the year, month and day.

Parameters

year	The given year.
month	The given month.
day	The given day.

Returns

The formatted date in YY/MM/DD.

4.4.2.2 Decimal()

```
static std::string Format::Decimal (
             double value,
             int precision )  [static]
```

Format fractions with some number of decimals.

Parameters

value	The given fraction number.
precison	The number of decimals reserved.

Returns

The formatted number with n decimals in string.

4.4.2.3 ElapsedTime()

```
static std::string Format::ElapsedTime (
             long times )  [static]
```

Get formatted time given the elapsed times.

Parameters

times	The given elapsed times.

Returns

The formatted time in hh:mm:ss.

4.4.2.4 Percentage()

```
static std::string Format::Percentage (
             double percent )  [static]
```

Format percentages with right number of decimals.

Parameters

percent	The percentage number.

Returns

The formatted percentage number in string.

4.4.2.5 Time()

```
static std::string Format::Time (
             int hour,
             int min,
             int sec )  [static]
```

Get formatted time given the hour, minute and second.

Parameters

hour	The given hour.
min	The given minute.
sec	The given second.

Returns

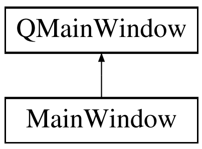
The formatted time in hh:mm:ss.

The documentation for this class was generated from the following file:

- include/format.h

4.5 MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Member Functions

- **MainWindow** (QWidget ∗parent=nullptr)
- void **setWindowStyle** ()

The documentation for this class was generated from the following file:

- include/mainwindow.h

4.6 Memory Class Reference

```
#include <memory.h>
```

Public Member Functions

- float TotalMemory ()
- float UsedMemory ()
- float Utilisation ()

4.6.1 Detailed Description

This class represents a computer memory and provides methods to retrieve information about its usage.

4.6.2 Member Function Documentation

4.6.2.1 TotalMemory()

```
float Memory::TotalMemory ( )
```

Getter method for total memory.

Returns

The amount of total memory in Kb.

4.6.2.2 UsedMemory()

```
float Memory::UsedMemory ( )
```

Getter method for used_memory.

Returns

The amount of used memory in Kb.

4.6.2.3 Utilisation()

```
float Memory::Utilisation ( )
```

Calculates and returns the memory utilisation as a percentage.

Returns

The memory utilisation as a percentage.

The documentation for this class was generated from the following file:

- include/memory.h

4.7 Power Class Reference

```
#include <power.h>
```

Public Member Functions

- double CurrHourEnergyUsage ()
- double CurrPowerUsage ()
- double TotalEnergyUsage ()
- vector< double > HoursEnergyUsages ()
- double HoursEnergyUsages (int hour)
- void ResetLogVector ()
- void SetLogVector (vector< double > datas)
- void UpdateLogVector (int hour)
- void UpdatePrevHoursEnergy ()
- void SetExtra (double value)
- void UpdatePowerAndEnergyUsage ()

4.7.1 Detailed Description

This class represents a computer power monitor and provides methods to retrieve and update information about power and energy usage.

4.7.2 Member Function Documentation

4.7.2.1 CurrHourEnergyUsage()

```
double Power::CurrHourEnergyUsage ( )
```

Getter method for current hour energy usage.

Returns

The energy usage in Wh drawn in current hour.

4.7.2.2 CurrPowerUsage()

```
double Power::CurrPowerUsage ( )
```

Getter method for current power usage.

Returns

The real-time power usage in Watts.

4.7.2.3 HoursEnergyUsages() `[1/2]`

```
vector< double > Power::HoursEnergyUsages ( )
```

Getter method for hourly energy usages in current day.

Returns

The vector containing energy usage in Wh drawn in every hour.

4.7.2.4 HoursEnergyUsages() `[2/2]`

```
double Power::HoursEnergyUsages (
             int hour )
```

Getter method for energy usage in particular hour.

Parameters

hour	The given hour.

Returns

The amount of energy usage in Wh in the given hour.

4.7.2.5 ResetLogVector()

```
void Power::ResetLogVector ( )
```

Reset vector for logging hourly energy usage.

4.7.2.6 SetExtra()

```
void Power::SetExtra (
             double value )
```

Setter method for extra.

Parameters

value	The value of extra.

4.7.2.7 SetLogVector()

```
void Power::SetLogVector (
            vector< double > datas )
```

Setter method for hourly energy usages.

Parameters

datas	The vector containing updated values of energy usages.

4.7.2.8 TotalEnergyUsage()

```
double Power::TotalEnergyUsage ( )
```

Get the total energy usage drawn in the current date.

Returns

The total energy usage in Wh.

4.7.2.9 UpdateLogVector()

```
void Power::UpdateLogVector (
            int hour )
```

Update energy usage in a particular hour.

Parameters

hour	The given (current) hour.

4.7.2.10 UpdatePowerAndEnergyUsage()

```
void Power::UpdatePowerAndEnergyUsage ( )
```

Update power and energy usage.

4.7.2.11 UpdatePrevHoursEnergy()

```
void Power::UpdatePrevHoursEnergy ( )
```

Update the total energy usage drawn in previous hours.

The documentation for this class was generated from the following file:

- include/power.h

4.8 PowerDAO Class Reference

```
#include <power_dao.h>
```

Public Member Functions

- void InitHoursLogFile (std::string curr_date, std::vector< double > usages)
- void InitDaysLogFile ()
- std::string LastLoggedDate ()
- void UpdateDaysLogFile (std::string last_logged_date, double total_usage)
- void UpdateHoursLogFile (std::string curr_date, std::vector< double > usages)
- std::vector< double > HoursEnergyUsages ()
- std::map< std::string, double > LastNDaysEnergyUsage (int n)

4.8.1 Detailed Description

This class is responsible for accessing data in power logging files.

4.8.2 Member Function Documentation

4.8.2.1 HoursEnergyUsages()

```
std::vector< double > PowerDAO::HoursEnergyUsages ( )
```

Read datas from the hours log file.

Returns

The hourly energy usages read from the logging file.

4.8.2.2 InitDaysLogFile()

```
void PowerDAO::InitDaysLogFile ( )
```

Create and initiate the days log file when the application is used the first time.

4.8.2.3 InitHoursLogFile()

```
void PowerDAO::InitHoursLogFile (
            std::string curr_date,
            std::vector< double > usages )
```

Create and initiate the hours log file when the application is used the first time.

Parameters

curr_date	The current date.
usages	The hourly energy usages.

4.8.2.4 LastLoggedDate()

```
std::string PowerDAO::LastLoggedDate ( )
```

Get the last date that has been logged in the file.

Returns

The last logged date.

4.8.2.5 LastNDaysEnergyUsage()

```
std::map< std::string, double > PowerDAO::LastNDaysEnergyUsage (
            int n )
```

Read datas from the days log file.

Parameters

n	The last n days.

Returns

The date and corresponding energy usage.

4.8.2.6 UpdateDaysLogFile()

```
void PowerDAO::UpdateDaysLogFile (
            std::string last_logged_date,
            double total_usage )
```

Append the total energy usage drawn in last day to the logging file.

Parameters

last_logged_date	The last logged date.
total_usage	The total energy usage in Wh drawn in last day.

4.8.2.7 UpdateHoursLogFile()

```
void PowerDAO::UpdateHoursLogFile (
            std::string curr_date,
            std::vector< double > usages )
```

Update the hours logging file.

Parameters

curr_date	The current date.
usages	The current status of energy usages in the given day.

The documentation for this class was generated from the following file:

- include/power_dao.h

4.9 Process Class Reference

```
#include <process.h>
```

Public Member Functions

- bool operator< (Process const &a) const
- bool operator> (Process const &a) const
- int Pid ()
- std::string User ()
- std::string Command ()
- float CpuUtilisation ()
- std::string Ram ()
- long int UpTime ()
- void SetPid (int pid)
- void SetUser (int pid)
- void SetCommand (int pid)
- void SetCpuUtilisation (int pid, long curr_total_jiffies)
- void SetRam (int pid)
- void SetUpTime (int pid)

4.9.1 Detailed Description

This class represents a system process with attrbutes of ID, user name, command, cpu utilisation, memory usage and up time.

4.9.2 Member Function Documentation

4.9.2.1 Command()

```
std::string Process::Command ( )
```

Getter method for command.

Returns

The command that generated the process.

4.9.2.2 CpuUtilisation()

```
float Process::CpuUtilisation ( )
```

Getter method for cpu utilisation.

Returns

The cpu utilisation of the process as a percentage.

4.9.2.3 operator<()

```
bool Process::operator< (
            Process const & a ) const
```

Overloads the less operator according to cpu utilisation.

Parameters

a	Another Process object.

Returns

True if this process is less than a; otherwise false.

4.9.2.4 operator>()

```
bool Process::operator> (
            Process const & a ) const
```

Overloads the greater operator according to cpu utilisation.

Parameters

a	Another Process object.

Returns

True if this process is greater than a; otherwise false.

4.9.2.5 Pid()

```
int Process::Pid ( )
```

Getter method for pid.

Returns

The process id.

4.9.2.6 Ram()

```
std::string Process::Ram ( )
```

Getter method for memory.

Returns

The memory used by the process in Mb.

4.9.2.7 SetCommand()

```
void Process::SetCommand (
            int pid )
```

Setter method for process command.

Parameters

pid	The process id.

4.9.2.8 SetCpuUtilisation()

```
void Process::SetCpuUtilisation (
            int pid,
            long curr_total_jiffies )
```

Setter method for cpu utilisation.

Parameters

pid	The process id.
curr_total_jiffies	The current total cpu jiffies.

4.9.2.9 SetPid()

```
void Process::SetPid (
            int pid )
```

Setter method for process id.

Parameters

pid	The process id.

4.9.2.10 SetRam()

```
void Process::SetRam (
            int pid )
```

Setter method for memory usage.

Parameters

pid	The process id.

4.9.2.11 SetUpTime()

```
void Process::SetUpTime (
             int pid )
```

Setter method for up time of the process.

Parameters

pid	The process id.

4.9.2.12 SetUser()

```
void Process::SetUser (
             int pid )
```

Setter method for process user.

Parameters

pid	The process id.

4.9.2.13 UpTime()

```
long int Process::UpTime ( )
```

Getter method for up time.

Returns

The age of the process in seconds.

4.9.2.14 User()

```
std::string Process::User ( )
```

Getter method for user.

Returns

The user (name) who runs the process.

The documentation for this class was generated from the following file:

- include/process.h

4.10 Processor Class Reference

```
#include <processor.h>
```

Public Member Functions

- int PhysicalCores ()
- int LogicalCores ()
- int HyperThreadedCores ()
- int ECores ()
- int PCores ()
- std::vector< float > Utilisations ()
- int Temperature ()
- void UpdateUtilisations ()

4.10.1 Detailed Description

This class represents a computer processor and provides methods to retrieve information about its cores and utilisation.

4.10.2 Member Function Documentation

4.10.2.1 ECores()

```
int Processor::ECores ( )
```

Getter method for e cores.

Returns

The number of efficiency cores of the processor.

4.10.2.2 HyperThreadedCores()

```
int Processor::HyperThreadedCores ( )
```

Getter method for hyperthreaded cores.

Returns

The number of hyperthreaded cores of the processor.

4.10.2.3 LogicalCores()

```
int Processor::LogicalCores ( )
```

Getter method for logical cores.

Returns

The number of logical cores of the processor.

4.10.2.4 PCores()

```
int Processor::PCores ( )
```

Getter method for p cores.

Returns

The number of performance cores of the processor.

4.10.2.5 PhysicalCores()

```
int Processor::PhysicalCores ( )
```

Getter method for physical cores.

Returns

The number of physical cores of the processor.

4.10.2.6 Temperature()

```
int Processor::Temperature ( )
```

Getter method for temperature.

Returns

The cpu temperature in degree Celsius.

4.10.2.7 UpdateUtilisations()

```
void Processor::UpdateUtilisations ( )
```

Update the utilisation of each cpu core and overall usage.

4.10.2.8 Utilisations()

```
std::vector< float > Processor::Utilisations ( )
```

Getter method for cpu utilisations.

Returns

The vector containing the utilisation in percentage of each cpu core.

The documentation for this class was generated from the following file:

- include/processor.h

4.11 System Class Reference

```
#include <system.h>
```

Public Member Functions

- string OperatingSystem ()
- string Kernel ()
- long UpTime ()
- int TotalProcesses ()
- int RunningProcesses ()
- float TotalMemory ()
- float UsedMemory ()
- float MemoryUtilisation ()
- int CpuTemperature ()
- vector< float > CpuUtilisations ()
- vector< Process > SortedProcesses ()
- double PowerUsage ()
- vector< double > HoursEnergyUsages ()
- double TotalEnergyUsage ()
- map< string, double > LastWeekEnergyUsage ()
- double TotalEnergyUsageLastWeek ()
- void BindToPCores ()
- void BindToAllCores ()
- void BindToECores ()
- void BindToPAndECores ()
- void SetEnergyCap (double cap)
- double EnergyCap ()

Static Public Member Functions

- static System ∗ Instance ()

4.11.1 Detailed Description

This class represents a computer system, integrated with essential components such as processor, memory and power. It also provides methods for retrieving essential information about the system.

4.11.2 Member Function Documentation

4.11.2.1 BindToAllCores()

```
void System::BindToAllCores ( )
```

Bind the most cpu-consuming processes to all cores.

4.11.2.2 BindToECores()

```
void System::BindToECores ( )
```

Bind the most cpu-consuming processes to efficiency cores.

4.11.2.3 BindToPAndECores()

```
void System::BindToPAndECores ( )
```

Bind the most cpu-consuming processes to 1/2 p-cores and 1/2 e-cores.

4.11.2.4 BindToPCores()

```
void System::BindToPCores ( )
```

Bind the most cpu-consuming processes to performance cores.

4.11.2.5 CpuTemperature()

```
int System::CpuTemperature ( )
```

Getter method for cpu temperature.

Returns

The cpu temperature in degree Celsius.

4.11.2.6 CpuUtilisations()

```
vector< float > System::CpuUtilisations ( )
```

Getter method for cpu utilisations.

Returns

The vector containing the utilisation of each cpu core.

4.11.2.7 EnergyCap()

```
double System::EnergyCap ( )
```

Getter method for energy cap.

Returns

The value of energy cap in Wh.

4.11.2.8 HoursEnergyUsages()

```
vector< double > System::HoursEnergyUsages ( )
```

Getter method for hourly energy usages in a day.

Returns

The vector containing energy usage in Wh in every hour.

4.11.2.9 Instance()

```
static System * System::Instance ( ) [static]
```

Static method that thread-safely provides a single instance of the class.

Returns

The single instance of System class.

4.11.2.10 Kernel()

```
string System::Kernel ( )
```

Getter method for kernel.

Returns

The kernel of the operating system.

4.11.2.11 LastWeekEnergyUsage()

```
map< string, double > System::LastWeekEnergyUsage ( )
```

Getter method for energy usages drawn in last week.

Returns

The date and corresponding energy usage in Wh during last week.

4.11.2.12 MemoryUtilisation()

```
float System::MemoryUtilisation ( )
```

Getter method for memory utilisation.

Returns

The live memory utilisation as a percentage.

4.11.2.13 OperatingSystem()

```
string System::OperatingSystem ( )
```

Getter method for operating system.

Returns

The type of the operating system.

4.11.2.14 PowerUsage()

```
double System::PowerUsage ( )
```

Getter method for live power usage.

Returns

The real-time power usage in watts.

4.11.2.15 RunningProcesses()

```
int System::RunningProcesses ( )
```

Getter method for running processes.

Returns

The number of running processes.

4.11.2.16 SetEnergyCap()

```
void System::SetEnergyCap (
             double cap )
```

Setter method for energy cap.

Parameters

cap	The value of energy cap.

4.11.2.17 SortedProcesses()

```
vector< Process > System::SortedProcesses ( )
```

Getter method for processes in descending order on cpu usage.

Returns

The sorted vector containing all processes.

4.11.2.18 TotalEnergyUsage()

```
double System::TotalEnergyUsage ( )
```

Getter method for today's total energy usage.

Returns

The total energy usage in Wh drawn today.

4.11.2.19 TotalEnergyUsageLastWeek()

```
double System::TotalEnergyUsageLastWeek ( )
```

Getter method for total energy usage drawn in last week.

Returns

The total energy usage in Wh in last week.

4.11.2.20 TotalMemory()

```
float System::TotalMemory ( )
```

Getter method for total memory.

Returns

The amount of system total memory in Kb.

4.11.2.21 TotalProcesses()

```
int System::TotalProcesses ( )
```

Getter method for total processes.

Returns

The number of total processes.

4.11.2.22 UpTime()

```
long System::UpTime ( )
```

Getter method for up time.

Returns

The up time of the system.

4.11.2.23 UsedMemory()

```
float System::UsedMemory ( )
```

Getter method for used memory.

Returns

The amount of current used memory in Kb.

The documentation for this class was generated from the following file:

- include/system.h

4.12 SystemParser Class Reference

Static Public Member Functions

- static std::string **OperatingSystem** ()
- static std::string **Kernel** ()
- static std::vector< int > **Pids** ()
- static std::vector< float > **MemoryInfo** ()
- static float **TotalMemory** ()
- static float **AvalMemory** ()
- static int **TotalProcesses** ()
- static int **RunningProcesses** ()
- static long **UpTime** ()
- static std::vector< long > **CpuTimes** (int cid)
- static long **TotalJiffies** (int cid)
- static long **IdleJiffiesC** (int cid)
- static long **ActiveJiffiesC** (int cid)
- static long **ActiveJiffiesP** (int pid)
- static std::string **Command** (int pid)
- static std::string **Ram** (int pid)
- static std::string **Uid** (int pid)
- static std::string **User** (int pid)
- static long int **UpTime** (int pid)

The documentation for this class was generated from the following file:

- include/system_parser.h

Chapter 5

File Documentation

5.1 command.h

```
#ifndef COMMAND_H
#define COMMAND_H
// Copyright (C) 2021 Remy van Elst
//
//     This program is free software: you can redistribute it and/or modify
//     it under the terms of the GNU General Public License as published by
//     the Free Software Foundation, either version 3 of the License, or
//     (at your option) any later version.
//
//     This program is distributed in the hope that it will be useful,
//     but WITHOUT ANY WARRANTY; without even the implied warranty of
//     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//     GNU General Public License for more details.
//
//     You should have received a copy of the GNU General Public License
//     along with this program.  If not, see <http://www.gnu.org/licenses/>.
#include <array>
#include <ostream>
#include <string>
#include <cstdio>

namespace raymii {

    struct CommandResult {
        std::string output;
        int exitstatus;
        friend std::ostream &operator«(std::ostream &os, const CommandResult &result) {
            os « "command exitstatus: " « result.exitstatus « " output: " « result.output;
            return os;
        }
        bool operator==(const CommandResult &rhs) const {
            return output == rhs.output &&
                   exitstatus == rhs.exitstatus;
        }
        bool operator!=(const CommandResult &rhs) const {
            return !(rhs == *this);
        }
    };

    class Command {
    public:
        static CommandResult exec(const std::string &command) {
            int exitcode = 0;
            std::array<char, 8192> buffer{};
            std::string result;
#ifdef _WIN32
#define popen _popen
#define pclose _pclose
#define WEXITSTATUS
#endif
            FILE *pipe = popen(command.c_str(), "r");
            if (pipe == nullptr) {
                throw std::runtime_error("popen() failed!");
            }
            try {
                std::size_t bytesread;
                while ((bytesread = std::fread(buffer.data(), sizeof(buffer.at(0)), sizeof(buffer),
pipe)) != 0) {
```

```
                    result += std::string(buffer.data(), bytesread);
                }
            } catch (...) {
                pclose(pipe);
                throw;
            }
            // Workaround "error: cannot take the address of an rvalue of type 'int'" on MacOS
            // see e.g. https://github.com/BestImageViewer/geeqie/commit/75c7df8b96592e10f7936dc1a28983be4089578c
            int res = pclose(pipe);
            exitcode = WEXITSTATUS(res);
            return CommandResult{result, exitcode};
        }

        // Only for reference in the article. Use regular ::exec.
        static CommandResult execFgets(const std::string &command) {
            int exitcode = 0;
            std::array<char, 8192> buffer{};
            std::string result;
#ifdef _WIN32
#define popen _popen
#define pclose _pclose
#define WEXITSTATUS
#endif
            FILE *pipe = popen(command.c_str(), "r");
            if (pipe == nullptr) {
                throw std::runtime_error("popen() failed!");
            }
            try {
                while (std::fgets(buffer.data(), buffer.size(), pipe) != nullptr) {
                    result += buffer.data();
                }
            } catch (...) {
                pclose(pipe);
                throw;
            }
            // Workaround "error: cannot take the address of an rvalue of type 'int'" on MacOS
            // see e.g. https://github.com/BestImageViewer/geeqie/commit/75c7df8b96592e10f7936dc1a28983be4089578c
            int res = pclose(pipe);
            exitcode = WEXITSTATUS(res);
            return CommandResult{result, exitcode};
        }
    };

}// namespace raymii
#endif//COMMAND_H
```

5.2 date_time.h

```
#ifndef DATE_TIME_H
#define DATE_TIME_H

#include <string>

class DateTime {

    public:
        static int Year();

        static int Month();

        static int Day();

        static int Hour();

        static int Min();

        static int Sec();

        static std::string CurrentDate();

        static std::string CurrentTime();

    private:
        static time_t now;
};

#endif
```

5.3 format.h

```
#ifndef FORMAT_H
#define FORMAT_H

#include <string>

class Format {
    public:
        static std::string ElapsedTime(long times);

        static std::string Date(int year, int month, int day);

        static std::string Time(int hour, int min, int sec);

        static std::string Decimal(double value, int precision);

        static std::string Percentage(double percent);

    private:
        static std::string AABBCC(int aa, int bb, int cc, std::string delimiter);
};

#endif
```

5.4 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTreeWidgetItem>
#include <QInputDialog>
#include <QWidget>
#include <QLabel>
#include <QtCharts>
#include <QTimer>

#include "include/system.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    void setWindowStyle();

private slots:
    void on_treeWidget_currentItemChanged(QTreeWidgetItem *current);

    void displayDate();

    void displayTime();

    void displayMemoryChart();

    void displayCpuChart();

    void displayTemperatureChart();

    void displayPowerUsage();

    void displayWeekEnergyUsage();

    void displayEnergyUsage();

    void maxTurboMode();

    void highPerformMode();

    void powerSaverMode();

    void balancedMode();

    void displayProcesses();

    void clickEnergyReportButtons();

    void displayDayReportGraph();

    void displayAccumDayReportGraph();

    void displayWeekReportGraph();

    void budgetInputReturn();

private:

    void createDonutChart(float value, std::string title, QChartView *chartView, std::string unit, int precision);

    Ui::MainWindow *ui;
    QTimer* timer = new QTimer(this);
    System* system_ = System::Instance();
};

#endif // MAINWINDOW_H
```

5.5 memory.h

```
#ifndef MEMORY_H
#define MEMORY_H

class Memory {

    public:
        Memory();

        float TotalMemory();

        float UsedMemory();

        float Utilisation();

    private:
        float total_memory;

};

#endif // MEMORY_H
```

5.6 power.h

```
#ifndef POWER_H
#define POWER_H

#include <string>
#include <vector>
#include <map>

using std::string;
using std::vector;
using std::map;

class Power {
    public:
        Power();

        double CurrHourEnergyUsage();

        double CurrPowerUsage();

        double TotalEnergyUsage();

        vector<double> HoursEnergyUsages();

        double HoursEnergyUsages(int hour);

        void ResetLogVector();

        void SetLogVector(vector<double> datas);

        void UpdateLogVector(int hour);

        void UpdatePrevHoursEnergy();

        void SetExtra(double value);

        void UpdatePowerAndEnergyUsage();

    private:
        long long EnergyUsageInUj();

        // The range of the energy counter
        static const long long MAX_ENERGY;

        vector<double> hours_energy_usages;
        double curr_hour_energy_usage = 0;
        double total_energy_usage = 0; // in a day
        double curr_power_usage = 0;

        // total energy usage (in uj) in the previous hours
        // since the system is booted
        long long prev_hours_energy = 0;

        // total energy usage since the pc is booted
        long long accum_energy_usage = 0;

        // current energy counter extracted from the system file
        long long energy = 0;

        // energy counter extracted in the last logging time
        long long prev_energy = 0;

        // number of times when the capped energy amount
        // is reached
        long long capped_times = 0;

        // The extra energy usage in the current hour
        // (in case the pc is rebooted)
        double extra = 0;
};

#endif
```

5.7 power_dao.h

```
#ifndef POWER_DAO_H
#define POWER_DAO_H

#include <string>
#include <vector>
#include <map>

class PowerDAO {

    public:
        void InitHoursLogFile(std::string curr_date, std::vector<double> usages);

        void InitDaysLogFile();

        std::string LastLoggedDate();

        void UpdateDaysLogFile(std::string last_logged_date, double total_usage);

        void UpdateHoursLogFile(std::string curr_date, std::vector<double> usages);

        std::vector<double> HoursEnergyUsages();

        std::map<std::string, double> LastNDaysEnergyUsage(int n);

    private:
        static const std::string HOURS_LOG_FILE;
        static const std::string DAYS_LOG_FILE;
};

#endif
```

5.8 process.h

```
#ifndef PROCESS_H
#define PROCESS_H

#include <string>
```

```

class Process {
    public:
        bool operator<(Process const& a) const;

        bool operator>(Process const& a) const;

        int Pid();

        std::string User();

        std::string Command();

        float CpuUtilisation();

        std::string Ram();

        long int UpTime();

        void SetPid(int pid);

        void SetUser(int pid);

        void SetCommand(int pid);

        void SetCpuUtilisation(int pid, long curr_total_jiffies);

        void SetRam(int pid);

        void SetUpTime(int pid);

    private:
        int pid;
        std::string user;
        std::string command;
        float cpu_utilisation;
        std::string ram;
        long up_time;

        // previous cpu jiffies of a proces
        long prev_jiffies_on_process = 0;
        // previous cpu total jiffies
        long prev_total_jiffies = 0;
};

#endif // PROCESS_H
```

5.9 processor.h

```
#ifndef PROCESSOR_H
#define PROCESSOR_H

#include <string>
#include <vector>

class Processor {

    public:
        Processor();

        int PhysicalCores();

        int LogicalCores();

        int HyperThreadedCores();

        int ECores();

        int PCores();

        std::vector<float> Utilisations();

        int Temperature();

        void UpdateUtilisations();

    private:
        static const int PHYSICAL_CORES;
        static const int LOGICAL_CORES;
        static const int HYPERTHREADED_CORES;
        static const int E_CORES;
        static const int P_CORES;

        // utilisation of all cpu cores
        std::vector<float> utilisations;
        // previous jiffies when cpu is not idle
        std::vector<long> prev_active_jiffies;
        // previous jiffies when cpu is in all states
        std::vector<long> prev_total_jiffies;
};

#endif
```

5.10 system.h

```
#ifndef SYSTEM_H
#define STSTEM_H

#include <unistd.h>

#include <string>
#include <vector>
#include <map>

#include "include/processor.h"
#include "include/memory.h"
#include "include/process.h"
#include "include/power.h"
#include "include/power_dao.h"

using std::string;
using std::vector;
using std::map;

class System {

    public:
        static System* Instance();

        string OperatingSystem();

        string Kernel();

        long UpTime();

        int TotalProcesses();

        int RunningProcesses();

        float TotalMemory();

        float UsedMemory();

        float MemoryUtilisation();

        int CpuTemperature();

        vector<float> CpuUtilisations();

        vector<Process> SortedProcesses();

        double PowerUsage();

        vector<double> HoursEnergyUsages();

        double TotalEnergyUsage();

        map<string, double> LastWeekEnergyUsage();

        double TotalEnergyUsageLastWeek();

        void BindToPCores();

        void BindToAllCores();

        void BindToECores();

        void BindToPAndECores();

        void SetEnergyCap(double cap);

        double EnergyCap();

    private:
        System();

        void UpdateProcesses();

        void UpdateEnergy();

        void SetUpEnergyDataAndFiles();

        vector<int> CpuConsumingProcesses();

        void BindProcesses(vector<int> pids, int low, int high);

        static System* instance;

        Processor cpu;
        Memory memory;
        map<int, Process> processes;
        string operating_system;
        string kernel;

        Power power;
        PowerDAO dao;
        double energy_cap = 500; // by default in Wh

        int hour;
};

#endif // SYSTEM_H
```

5.11 system_parser.h

```
#ifndef SYSTEM_PARSER_H
#define SYSTEM_PARSER_H

#include <string>
#include <vector>

class SystemParser {

    public:
        // System
        static std::string OperatingSystem();
        static std::string Kernel();
        static std::vector<int> Pids();
        static std::vector<float> MemoryInfo();
        static float TotalMemory();
        static float AvalMemory();
        static int TotalProcesses();
        static int RunningProcesses();
        static long UpTime();

        // CPU
        static std::vector<long> CpuTimes(int cid);
        static long TotalJiffies(int cid);
        static long IdleJiffiesC(int cid);
        static long ActiveJiffiesC(int cid);

        // Processes
        static long ActiveJiffiesP(int pid);
        static std::string Command(int pid);
        static std::string Ram(int pid);
        static std::string Uid(int pid);
        static std::string User(int pid);
        static long int UpTime(int pid);


    private:
        // Utils
        static std::string KeyValParser(std::string, std::string);

        // Paths
        static const std::string kProcDirectory;
        static const std::string kCmdlineFilename;
        static const std::string kCpuinfoFilename;
        static const std::string kStatusFilename;
        static const std::string kStatFilename;
        static const std::string kUptimeFilename;
        static const std::string kMeminfoFilename;
        static const std::string kVersionFilename;
        static const std::string kOSPath;
        static const std::string kPasswordPath;
};

#endif // SYSTEM_PARSER_H
```

Index