Linux Energy Monitor Application

1.0

Generated by Doxygen 1.9.7

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 raymii::Command Class Reference

Static Public Member Functions

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

3.1.1 Member Function Documentation

3.1.1.1 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:

- src/command.h

3.2 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:

- src/command.h

3.3 Memory Class Reference

`#include <memory.h>`

Public Member Functions

- float TotalMemory ()
- float UsedMemory ()
- float Utilisation ()
- void UpdateUsedMemory ()

3.3.1 Detailed Description

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

3.3.2 Member Function Documentation

3.3.2.1 TotalMemory()

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

Getter method for total_memory.

Returns

The amount of used memory in Kb.

3.3.2.2 UpdateUsedMemory()

```
void Memory::UpdateUsedMemory ( )
```

Update the amount of used memory.

3.3.2.3 UsedMemory()

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

Getter method for used_memory.

Returns

The amount of used memory in Kb.

3.3.2.4 Utilisation()

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

Calculates and returns the memory utilization as a percentage.

Returns

The memory utilization as a percentage.

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

- src/memory.h
- src/memory.cpp

3.4 Power Class Reference

Public Member Functions

- void UpdatePowerAndEnergyUsage ()
- double CurrHourEnergyUsage ()
- double CurrPowerUsage ()
- double TodaysTotalEnergyUsage ()
- std::vector< double > HoursEnergyUsages ()
- std::map< std::string, double > LastNDaysEnergyUsage (int n)

Static Public Member Functions

- static Power * Instance ()

3.4.1 Member Function Documentation

3.4.1.1 CurrHourEnergyUsage()

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

Getter method for curr_hour_energy_usage.

Returns

The energy usage in Wh in current hour.

3.4.1.2 CurrPowerUsage()

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

Getter method for curr_power_usage.

Returns

The real-time power usage in watts.

3.4.1.3 HoursEnergyUsages()

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

Getter method for hours_energy_usages.

Returns

The vector containing energy usage in Wh in every hour.

3.4.1.4 Instance()

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

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

Returns

The single instance of Power class.

3.4.1.5 LastNDaysEnergyUsage()

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

Get the energy usage drawn in the last n days.

Parameters

n	The last n days.

Returns

The date and corresponding energy usage.

3.4.1.6 TodaysTotalEnergyUsage()

```
double Power::TodaysTotalEnergyUsage ( )
```

Get the total energy usage that is drawn in the current date.

Returns

The total energy usage in Wh.

3.4.1.7 UpdatePowerAndEnergyUsage()

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

Keep updating and logging power and energy usage.

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

- src/power.h
- src/power.cpp

3.5 Process Class Reference

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 SetCpuUtilization (int pid, long curr_total_jiffies)
- void SetRam (int pid)
- void SetUpTime (int pid)

3.5.1 Member Function Documentation

3.5.1.1 Command()

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

Getter method for command.

Returns

The command that generated the process.

3.5.1.2 CpuUtilisation()

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

Getter method for cpu utilisation.

Returns

The cpu utilisation of the process as a percentage.

3.5.1.3 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.

3.5.1.4 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.

3.5.1.5 Pid()

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

Getter method for pid.

Returns

The process id.

3.5.1.6 Ram()

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

Getter method for memory.

Returns

The memory used by the process in Mb.

3.5.1.7 SetCommand()

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

Setter method for process command.

Parameters

pid	The process id.

3.5.1.8 SetCpuUtilization()

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

Setter method for cpu utilisation.

Parameters

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

3.5.1.9 SetPid()

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

Setter method for process id.

Parameters

pid	The process id.

3.5.1.10 SetRam()

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

Setter method for memory usage.

Parameters

pid	The process id.

3.5.1.11 SetUpTime()

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

Setter method for up time of the process.

Parameters

pid	The process id.

3.5.1.12 SetUser()

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

Setter method for process user.

Parameters

pid	The process id.

3.5.1.13 UpTime()

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

Getter method for up time.

Returns

The age of the process in seconds.

3.5.1.14 User()

```
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 files:

- src/process.h
- src/process.cpp

3.6 Processor Class Reference

Public Member Functions

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

3.6.1 Member Function Documentation

3.6.1.1 ECores()

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

Getter method for e_cores.

Returns

The number of efficiency cores of the processor.

3.6.1.2 HyperThreadedCores()

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

Getter method for hyperthreaded_cores.

Returns

The number of hyperthreaded cores of the processor.

3.6.1.3 LogicalCores()

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

Getter method for logical_cores.

Returns

The number of logical cores of the processor.

3.6.1.4 PCores()

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

Getter method for e_cores.

Returns

The number of performance cores of the processor.

3.6.1.5 PhysicalCores()

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

Getter method for physical_cores.

Returns

The number of physical cores of the processor.

3.6.1.6 Temperature()

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

Getter method for temperature.

Returns

The cpu temperature in degree Celsius.

3.6.1.7 UpdateTemperature()

```
void Processor::UpdateTemperature ( )
```

Update the cpu temperature.

3.6.1.8 UpdateUtilisations()

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

Update the utilisation of each cpu core and overall usage in percentage.

3.6.1.9 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 files:

- src/processor.h
- src/processor.cpp

3.7 System Class Reference

Public Member Functions

- std::string OperatingSystem ()
- std::string Kernel ()
- long UpTime ()
- int TotalProcesses ()
- int RunningProcesses ()
- float TotalMemory ()
- float UsedMemory ()
- float MemoryUtilisation ()
- int CpuTemperature ()
- std::vector< float > CpuUtilisations ()
- std::vector< Process > SortedProcesses ()
- void UpdateCpuAndMemory ()
- void UpdateProcesses ()
- void BindToPCores ()
- void BindToAllCores ()
- void BindToECores ()

Static Public Member Functions

- static System * Instance ()

3.7.1 Member Function Documentation

3.7.1.1 BindToAllCores()

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

Bind the most cpu consuming processes to all cores.

3.7.1.2 BindToECores()

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

Bind the most cpu consuming processes to efficiency cores.

3.7.1.3 BindToPCores()

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

Bind the most cpu consuming processes to performance cores.

3.7.1.4 CpuTemperature()

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

Getter method for cpu temperature.

Returns

The cpu temperature in degree Celsius.

3.7.1.5 CpuUtilisations()

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

Getter method for cpu utilisations.

Returns

The vector containing the utilisation of each cpu core.

3.7.1.6 Instance()

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

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

Returns

The single instance of System class.

3.7.1.7 Kernel()

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

Getter method for kernel.

Returns

The kernel of the operating system.

3.7.1.8 MemoryUtilisation()

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

Getter method for memory utilisation as a percentage.

Returns

The live memory utilisation as a percentage.

3.7.1.9 OperatingSystem()

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

Getter method for operating system.

Returns

The type of the operating system.

3.7.1.10 RunningProcesses()

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

Getter method for running processes.

Returns

The number of running processes.

3.7.1.11 SortedProcesses()

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

Getter method for processes in descending order on cpu usage.

Returns

The sorted vector containing all processes.

3.7.1.12 TotalMemory()

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

Getter method for total memory.

Returns

The amount of system total memory in Kb.

3.7.1.13 TotalProcesses()

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

Getter method for total processes.

Returns

The number of total processes.

3.7.1.14 UpdateCpuAndMemory()

```
void System::UpdateCpuAndMemory ( )
```

Update cpu utilisations & temperature and memory usage.

3.7.1.15 UpdateProcesses()

```
void System::UpdateProcesses ( )
```

Update the processes.

3.7.1.16 UpTime()

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

Getter method for uptime.

Returns

The uptime of the system.

3.7.1.17 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 files:

- src/system.h
- src/system.cpp

3.8 View Class Reference

Public Member Functions

- void **ServiceSelect** ()

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

- src/view.h
- src/view.cpp

Chapter 4

File Documentation

4.1 command.h

```
00001 #ifndef COMMAND_H
00002 #define COMMAND_H
00003 // Copyright (C) 2021 Remy van Elst
00004 //
00005 //     This program is free software: you can redistribute it and/or modify
00006 //     it under the terms of the GNU General Public License as published by
00007 //     the Free Software Foundation, either version 3 of the License, or
00008 //     (at your option) any later version.
00009 //
00010 //     This program is distributed in the hope that it will be useful,
00011 //     but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 //     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 //     GNU General Public License for more details.
00014 //
00015 //     You should have received a copy of the GNU General Public License
00016 //     along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 #include <array>
00018 #include <ostream>
00019 #include <string>
00020 #include <cstdio>
00021
00022 namespace raymii {
00023
00024     struct CommandResult {
00025         std::string output;
00026         int exitstatus;
00027         friend std::ostream &operator«(std::ostream &os, const CommandResult &result) {
00028             os « "command exitstatus: " « result.exitstatus « " output: " « result.output;
00029             return os;
00030         }
00031         bool operator==(const CommandResult &rhs) const {
00032             return output == rhs.output &&
00033                    exitstatus == rhs.exitstatus;
00034         }
00035         bool operator!=(const CommandResult &rhs) const {
00036             return !(rhs == *this);
00037         }
00038     };
00039
00040     class Command {
00041     public:
00050         static CommandResult exec(const std::string &command) {
00051             int exitcode = 0;
00052             std::array<char, 8192> buffer{};
00053             std::string result;
00054 #ifdef _WIN32
00055 #define popen _popen
00056 #define pclose _pclose
00057 #define WEXITSTATUS
00058 #endif
00059             FILE *pipe = popen(command.c_str(), "r");
00060             if (pipe == nullptr) {
00061                 throw std::runtime_error("popen() failed!");
00062             }
00063             try {
00064                 std::size_t bytesread;
00065                 while ((bytesread = std::fread(buffer.data(), sizeof(buffer.at(0)), sizeof(buffer),
      pipe)) != 0) {
```

```
00066                     result += std::string(buffer.data(), bytesread);
00067                 }
00068             } catch (...) {
00069                 pclose(pipe);
00070                 throw;
00071             }
00072             // Workaround "error: cannot take the address of an rvalue of type 'int'" on MacOS
00073             // see e.g.
      https://github.com/BestImageViewer/geeqie/commit/75c7df8b96592e10f7936dc1a28983be4089578c
00074             int res = pclose(pipe);
00075             exitcode = WEXITSTATUS(res);
00076             return CommandResult{result, exitcode};
00077         }
00078
00079         // Only for reference in the article. Use regular ::exec.
00080         static CommandResult execFgets(const std::string &command) {
00081             int exitcode = 0;
00082             std::array<char, 8192> buffer{};
00083             std::string result;
00084 #ifdef _WIN32
00085 #define popen _popen
00086 #define pclose _pclose
00087 #define WEXITSTATUS
00088 #endif
00089             FILE *pipe = popen(command.c_str(), "r");
00090             if (pipe == nullptr) {
00091                 throw std::runtime_error("popen() failed!");
00092             }
00093             try {
00094                 while (std::fgets(buffer.data(), buffer.size(), pipe) != nullptr) {
00095                     result += buffer.data();
00096                 }
00097             } catch (...) {
00098                 pclose(pipe);
00099                 throw;
00100             }
00101             // Workaround "error: cannot take the address of an rvalue of type 'int'" on MacOS
00102             // see e.g.
      https://github.com/BestImageViewer/geeqie/commit/75c7df8b96592e10f7936dc1a28983be4089578c
00103             int res = pclose(pipe);
00104             exitcode = WEXITSTATUS(res);
00105             return CommandResult{result, exitcode};
00106         }
00107     };
00108
00109 }// namespace raymii
00110 #endif//COMMAND_H
```

4.2 memory.h

```
00001 #ifndef MEMORY_H
00002 #define MEMORY_H
00003
00008 class Memory {
00009
00010     public:
00015         float TotalMemory();
00016
00021         float UsedMemory();
00022
00027         float Utilisation();
00028
00032         void UpdateUsedMemory();
00033
00034         Memory();
00035
00036     private:
00037         float total_memory;
00038         float used_memory;
00039
00040 };
00041
00042 #endif // MEMORY_H
```

4.3 power.h

```
00001 #ifndef POWER_H
00002 #define POWER_H
00003
```

```
00004 #include <string>
00005 #include <vector>
00006 #include <map>
00007
00008 class Power {
00009     public:
00015         static Power* Instance();
00016
00020         void UpdatePowerAndEnergyUsage();
00021
00026         double CurrHourEnergyUsage();
00027
00032         double CurrPowerUsage();
00033
00038         double TodaysTotalEnergyUsage();
00039
00044         std::vector<double> HoursEnergyUsages();
00045
00051         std::map<std::string, double> LastNDaysEnergyUsage(int n);
00052
00053     private:
00054         Power();
00055
00059         void ResetLogVector();
00060
00068         std::string FormatDate(int year, int mon, int day);
00069
00074         std::string LastLoggedDate();
00075
00080         void UpdateDaysLogFile(std::string date);
00081
00086         long long EnergyUsageInUj();
00087
00092         void UpdateHoursLogFile(std::string date);
00093
00097         void UpdateLogVector();
00098
00099         static Power* instance;
00100         std::string hours_log_file = "../data/hours_power_usage.csv";
00101         std::string days_log_file = "../data/days_power_usage.csv";
00102         std::string energy_usage_path = "/sys/class/powercap/intel-rapl/intel-rapl:1/energy_uj";
00103         std::string mex_energy_path =
      "/sys/class/powercap/intel-rapl/intel-rapl:1/max_energy_range_uj";
00104
00105         // Record energy usages in every hour in a day
00106         std::vector<double> hours_energy_usages;
00107
00108         // The energy amount got from the system will
00109         // reset to zero when it exceeds this bound
00110         long long max_energy;
00111         double curr_hour_energy_usage;
00112         double curr_power_usage;
00113
00114 };
00115
00116 #endif
```

4.4 process.h

```
00001 #ifndef PROCESS_H
00002 #define PROCESS_H
00003
00004 #include <string>
00005
00006 class Process {
00007     public:
00013         bool operator<(Process const& a) const;
00014
00020         bool operator>(Process const& a) const;
00021
00026         int Pid();
00027
00032         std::string User();
00033
00038         std::string Command();
00039
00044         float CpuUtilisation();
00045
00050         std::string Ram();
00051
00056         long int UpTime();
00057
00062         void SetPid(int pid);
```

```
00063
00068         void SetUser(int pid);
00069
00074         void SetCommand(int pid);
00075
00081         void SetCpuUtilization(int pid, long curr_total_jiffies);
00082
00087         void SetRam(int pid);
00088
00093         void SetUpTime(int pid);
00094
00095     private:
00096         int pid;
00097         std::string user;
00098         std::string command;
00099         float cpu_utilisation;
00100         std::string ram;
00101         long up_time;
00102
00103         // Record previous cpu active/total jiffies to calculate
00104         // the live utilisation of each process in a short period
00105         long prev_active_jiffies{0};
00106         long prev_total_jiffies{0};
00107 };
00108
00109 #endif // PROCESS_H
```

4.5 processor.h

```
00001 #ifndef PROCESSOR_H
00002 #define PROCESSOR_H
00003
00004 #include <vector>
00005
00006 class Processor {
00007
00008     public:
00013         int PhysicalCores();
00014
00019         int LogicalCores();
00020
00025         int HyperThreadedCores();
00026
00031         int ECores();
00032
00037         int PCores();
00038
00043         std::vector<float> Utilisations();
00044
00049         int Temperature();
00050
00054         void UpdateUtilisations();
00055
00059         void UpdateTemperature();
00060
00061         Processor();
00062
00063     private:
00064         int physical_cores;
00065         int logical_cores;
00066         int hyperthreaded_cores;
00067         int e_cores;
00068         int p_cores;
00069         int temperature;
00070
00071         // First element represents general/overall cpu utilisation,
00072         // followed by that of all cores with index cid-1
00073         std::vector<float> utilisations;
00074
00075         // Record previous cpu active/total jiffies to calculate
00076         // the live utilisation of each core in a short period
00077         std::vector<long> prev_active_jiffies;
00078         std::vector<long> prev_total_jiffies;
00079 };
00080
00081 #endif
```

4.6 system.h

```
00001 #ifndef SYSTEM_H
```

```
00002 #define STSTEM_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include <map>
00007
00008 #include "processor.h"
00009 #include "memory.h"
00010 #include "process.h"
00011
00012 class System {
00013
00014     public:
00020         static System* Instance();
00021
00026         std::string OperatingSystem();
00027
00032         std::string Kernel();
00033
00038         long UpTime();
00039
00044         int TotalProcesses();
00045
00050         int RunningProcesses();
00051
00056         float TotalMemory();
00057
00062         float UsedMemory();
00063
00068         float MemoryUtilisation();
00069
00074         int CpuTemperature();
00075
00080         std::vector<float> CpuUtilisations();
00081
00086         std::vector<Process> SortedProcesses();
00087
00091         void UpdateCpuAndMemory();
00092
00096         void UpdateProcesses();
00097
00101         void BindToPCores();
00102
00106         void BindToAllCores();
00107
00111         void BindToECores();
00112
00113     private:
00114         System();
00115
00120         std::vector<int> CpuConsumingProcesses();
00121
00128         void BindProcesses(std::vector<int> pids, int low, int high);
00129
00130         static System* instance;
00131         std::string operating_system;
00132         std::string kernel;
00133         Processor cpu;
00134         Memory memory;
00135         std::map<int, Process> processes;
00136 };
00137
00138 #endif // SYSTEM_H
```

4.7 system_parser.h

```
00001 #ifndef SYSTEM_PARSER_H
00002 #define SYSTEM_PARSER_H
00003
00004 #include <string>
00005 #include <vector>
00006
00007 namespace SystemParser {
00008     // Paths
00009     const std::string kProcDirectory{"/proc/"};
00010     const std::string kCmdlineFilename{"/cmdline"};
00011     const std::string kCpuinfoFilename{"/cpuinfo"};
00012     const std::string kStatusFilename{"/status"};
00013     const std::string kStatFilename{"/stat"};
00014     const std::string kUptimeFilename{"/uptime"};
00015     const std::string kMeminfoFilename{"/meminfo"};
00016     const std::string kVersionFilename{"/version"};
00017     const std::string kOSPath{"/etc/os-release"};
```

```
00018     const std::string kPasswordPath{"/etc/passwd"};
00019
00020     // Utils
00021     std::string KeyValParser(std::string, std::string);
00022
00023     // System
00024     std::string OperatingSystem();
00025     std::string Kernel();
00026     std::vector<int> Pids();
00027     std::vector<float> MemoryInfo();
00028     float TotalMemory();
00029     float AvalMemory();
00030     int TotalProcesses();
00031     int RunningProcesses();
00032     long UpTime();
00033
00034     // CPU
00035     std::vector<long> CpuTimes(int cid);
00036     long TotalJiffies(int cid);
00037     long IdleJiffiesC(int cid);
00038     long ActiveJiffiesC(int cid);
00039
00040     // Processes
00041     long ActiveJiffiesP(int pid);
00042     std::string Command(int pid);
00043     std::string Ram(int pid);
00044     std::string Uid(int pid);
00045     std::string User(int pid);
00046     long int UpTime(int pid);
00047 };
00048
00049 #endif // SYSTEM_PARSER_H
```

4.8 view.h

```
00001 #ifndef VIEW_H
00002 #define VIEW_H
00003
00004 #include <string>
00005
00006 #include "system.h"
00007 #include "power.h"
00008
00009 class View {
00010
00011     public:
00012         void ServiceSelect();
00013
00014     private:
00015         System* system_ = System::Instance();
00016         Power* power_ = Power::Instance();
00017
00018         void DisplaySystemInfo();
00019         void DisplayProcesses();
00020         void DisplayTodaysEnergyUsage();
00021         void DisplayLastWeekEnergyUsage();
00022         void DisplayLivePowerUsage();
00023         void PowerUsageSelect();
00024         void PowerModeSelect();
00025
00026 };
00027
00028 #endif
```

Index