

Linux Energy Monitor Application

1.0

Generated by Doxygen 1.9.7

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 raymii::Command Class Reference	7
4.1.1 Member Function Documentation	7
4.1.1.1 exec()	7
4.2 raymii::CommandResult Struct Reference	8
4.3 DateTime Class Reference	8
4.3.1 Detailed Description	8
4.3.2 Member Function Documentation	8
4.3.2.1 CurrentDate()	9
4.3.2.2 CurrentTime()	9
4.3.2.3 Day()	9
4.3.2.4 Hour()	9
4.3.2.5 Min()	10
4.3.2.6 Month()	10
4.3.2.7 Sec()	10
4.3.2.8 Year()	10
4.4 Format Class Reference	11
4.4.1 Detailed Description	11
4.4.2 Member Function Documentation	11
4.4.2.1 Date()	11
4.4.2.2 Decimal()	11
4.4.2.3 ElapsedTime()	12
4.4.2.4 Percentage()	12
4.4.2.5 Time()	13
4.5 MainWindow Class Reference	13
4.6 Memory Class Reference	13
4.6.1 Detailed Description	14
4.6.2 Member Function Documentation	14
4.6.2.1 TotalMemory()	14
4.6.2.2 UsedMemory()	14
4.6.2.3 Utilisation()	14
4.7 Power Class Reference	15
4.7.1 Detailed Description	15
4.7.2 Member Function Documentation	15

4.7.2.1 CurrHourEnergyUsage()	15
4.7.2.2 CurrPowerUsage()	15
4.7.2.3 HoursEnergyUsages() [1/2]	16
4.7.2.4 HoursEnergyUsages() [2/2]	16
4.7.2.5 ResetLogVector()	16
4.7.2.6 SetExtra()	16
4.7.2.7 SetLogVector()	17
4.7.2.8 TotalEnergyUsage()	17
4.7.2.9 UpdateLogVector()	17
4.7.2.10 UpdatePowerAndEnergyUsage()	17
4.7.2.11 UpdatePrevHoursEnergy()	18
4.8 PowerDAO Class Reference	18
4.8.1 Detailed Description	18
4.8.2 Member Function Documentation	18
4.8.2.1 HoursEnergyUsages()	18
4.8.2.2 InitDaysLogFile()	18
4.8.2.3 InitHoursLogFile()	18
4.8.2.4 LastLoggedDate()	19
4.8.2.5 LastNDaysEnergyUsage()	19
4.8.2.6 UpdateDaysLogFile()	19
4.8.2.7 UpdateHoursLogFile()	20
4.9 Process Class Reference	20
4.9.1 Detailed Description	20
4.9.2 Member Function Documentation	21
4.9.2.1 Command()	21
4.9.2.2 CpuUtilisation()	21
4.9.2.3 operator<()	21
4.9.2.4 operator>()	22
4.9.2.5 Pid()	22
4.9.2.6 Ram()	22
4.9.2.7 SetCommand()	22
4.9.2.8 SetCpuUtilisation()	23
4.9.2.9 SetPid()	23
4.9.2.10 SetRam()	23
4.9.2.11 SetUpTime()	24
4.9.2.12 SetUser()	24
4.9.2.13 UpTime()	24
4.9.2.14 User()	24
4.10 Processor Class Reference	25
4.10.1 Detailed Description	25
4.10.2 Member Function Documentation	25
4.10.2.1 ECores()	25

4.10.2.2 HyperThreadedCores()	25
4.10.2.3 LogicalCores()	26
4.10.2.4 PCores()	26
4.10.2.5 PhysicalCores()	26
4.10.2.6 Temperature()	26
4.10.2.7 UpdateUtilisations()	27
4.10.2.8 Utilisations()	27
4.11 System Class Reference	27
4.11.1 Detailed Description	28
4.11.2 Member Function Documentation	28
4.11.2.1 BindToAllCores()	28
4.11.2.2 BindToECores()	28
4.11.2.3 BindToPAndECores()	28
4.11.2.4 BindToPCores()	28
4.11.2.5 CpuTemperature()	28
4.11.2.6 CpuUtilisations()	29
4.11.2.7 EnergyCap()	29
4.11.2.8 HoursEnergyUsages()	29
4.11.2.9 Instance()	29
4.11.2.10 Kernel()	30
4.11.2.11 LastWeekEnergyUsage()	30
4.11.2.12 MemoryUtilisation()	30
4.11.2.13 OperatingSystem()	30
4.11.2.14 PowerUsage()	31
4.11.2.15 RunningProcesses()	31
4.11.2.16 SetEnergyCap()	31
4.11.2.17 SortedProcesses()	31
4.11.2.18 TotalEnergyUsage()	32
4.11.2.19 TotalEnergyUsageLastWeek()	32
4.11.2.20 TotalMemory()	32
4.11.2.21 TotalProcesses()	32
4.11.2.22 UpTime()	33
4.11.2.23 UsedMemory()	33
4.12 SystemParser Class Reference	33
5 File Documentation	35
5.1 command.h	35
5.2 date_time.h	36
5.3 format.h	37
5.4 mainwindow.h	37
5.5 memory.h	38
5.6 power.h	38

5.7 power_dao.h	39
5.8 process.h	39
5.9 processor.h	40
5.10 system.h	41
5.11 system_parser.h	42
Index	43

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

raymii::Command	7
raymii::CommandResult	8
DateTime	8
Format	11
Memory	13
Power	15
PowerDAO	18
Process	20
Processor	25
QMainWindow	
MainWindow	13
System	27
SystemParser	33

Chapter 2

Class Index

2.1 Class List

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

raymii::Command	7
raymii::CommandResult	8
DateTime	8
Format	11
MainWindow	13
Memory	13
Power	15
PowerDAO	18
Process	20
Processor	25
System	27
SystemParser	33

Chapter 3

File Index

3.1 File List

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

include/command.h	35
include/date_time.h	36
include/format.h	37
include/mainwindow.h	37
include/memory.h	38
include/power.h	38
include/power_dao.h	39
include/process.h	39
include/processor.h	40
include/system.h	41
include/system_parser.h	42

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

<i>command</i>	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

<i>year</i>	The given year.
<i>month</i>	The given month.
<i>day</i>	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

<i>value</i>	The given fraction number.
<i>precison</i>	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

<i>times</i>	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

<i>percent</i>	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

<i>hour</i>	The given hour.
<i>min</i>	The given minute.
<i>sec</i>	The given second.

Returns

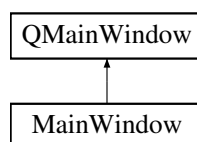
The formatted time in hh:mm:ss.

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

<i>hour</i>	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

<i>value</i>	The value of extra.
--------------	---------------------

4.7.2.7 SetLogVector()

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

Setter method for hourly energy usages.

Parameters

<i>datas</i>	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

<i>hour</i>	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

<i>curr_date</i>	The current date.
<i>usages</i>	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

<i>n</i>	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

<i>last_logged_date</i>	The last logged date.
<i>total_usage</i>	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

<i>curr_date</i>	The current date.
<i>usages</i>	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 attributes 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

<i>a</i>	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

<i>a</i>	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

<i>pid</i>	The process id.
------------	-----------------

4.9.2.8 SetCpuUtilisation()

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

Setter method for cpu utilisation.

Parameters

<i>pid</i>	The process id.
<i>curr_total_jiffies</i>	The current total cpu jiffies.

4.9.2.9 SetPid()

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

Setter method for process id.

Parameters

<i>pid</i>	The process id.
------------	-----------------

4.9.2.10 SetRam()

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

Setter method for memory usage.

Parameters

<i>pid</i>	The process id.
------------	-----------------

4.9.2.11 SetUpTime()

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

Setter method for up time of the process.

Parameters

<i>pid</i>	The process id.
------------	-----------------

4.9.2.12 SetUser()

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

Setter method for process user.

Parameters

<i>pid</i>	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

<i>cap</i>	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

```
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:
00042         static CommandResult exec(const std::string &command) {
00043             int exitcode = 0;
00044             std::array<char, 8192> buffer{};
00045             std::string result;
00046 #ifdef _WIN32
00047             #define popen _popen
00048             #define pclose _pclose
00049             #define WEXITSTATUS
00050             #endif
00051             FILE *pipe = popen(command.c_str(), "r");
00052             if (pipe == nullptr) {
00053                 throw std::runtime_error("popen() failed!");
00054             }
00055             try {
00056                 std::size_t bytesread;
00057                 while ((bytesread = std::fread(buffer.data(), sizeof(buffer.at(0)), sizeof(buffer),
00058                     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

```

5.2 date_time.h

```

00001 #ifndef DATE_TIME_H
00002 #define DATE_TIME_H
00003
00004 #include <string>
00005
00009 class DateTime {
00010
00011     public:
00016         static int Year();
00017
00022         static int Month();
00023
00028         static int Day();
00029
00034         static int Hour();
00035
00040         static int Min();
00041
00046         static int Sec();
00047
00052         static std::string CurrentDate();
00053
00058         static std::string CurrentTime();
00059
00060     private:
00061         static time_t now;
00062 };
00063
00064 #endif

```

5.3 format.h

```

00001 #ifndef FORMAT_H
00002 #define FORMAT_H
00003
00004 #include <string>
00005
00009 class Format {
00010     public:
00016         static std::string ElapsedTime(long times);
00017
00025         static std::string Date(int year, int month, int day);
00026
00034         static std::string Time(int hour, int min, int sec);
00035
00042         static std::string Decimal(double value, int precision);
00043
00049         static std::string Percentage(double percent);
00050
00051     private:
00052         static std::string AABBC(int aa, int bb, int cc, std::string delimiter);
00053 };
00054
00055 #endif

```

5.4 mainwindow.h

```

00001 #ifndef MAINWINDOW_H
00002 #define MAINWINDOW_H
00003
00004 #include <QMainWindow>
00005 #include <QTreeWidgetItem>
00006 #include <QInputDialog>
00007 #include <QWidget>
00008 #include <QLabel>
00009 #include <QtCharts>
00010 #include <QTimer>
00011
00012 #include "include/system.h"
00013
00014 namespace Ui {
00015     class MainWindow;
00016 }
00017
00018 class MainWindow : public QMainWindow
00019 {
00020     Q_OBJECT
00021
00022     public:
00023         explicit MainWindow(QWidget *parent = nullptr);
00024         ~MainWindow();
00025
00026         void setWindowStyle();
00027
00028     private slots:
00029         void on_treeWidget_currentItemChanged(QTreeWidgetItem *current);
00030
00031         void displayDate();
00032
00033         void displayTime();
00034
00035         void displayMemoryChart();
00036
00037         void displayCpuChart();
00038
00039         void displayTemperatureChart();
00040
00041         void displayPowerUsage();
00042
00043         void displayWeekEnergyUsage();
00044
00045         void displayEnergyUsage();
00046
00047         void maxTurboMode();
00048
00049         void highPerformMode();
00050
00051         void powerSaverMode();
00052
00053         void balancedMode();
00054
00055         void displayProcesses();
00056

```

```

00057     void clickEnergyReportButtons();
00058
00059     void displayDayReportGraph();
00060
00061     void displayAccumDayReportGraph();
00062
00063     void displayWeekReportGraph();
00064
00065     void budgetInputReturn();
00066
00067 private:
00068
00069     void createDonutChart(float value, std::string title, QChartView *chartView, std::string unit, int
precision);
00070
00071     Ui::MainWindow *ui;
00072     QTimer* timer = new QTimer(this);
00073     System* system_ = System::Instance();
00074 };
00075
00076 #endif // MAINWINDOW_H

```

5.5 memory.h

```

00001 #ifndef MEMORY_H
00002 #define MEMORY_H
00003
00008 class Memory {
00009
00010     public:
00011         Memory();
00012
00017         float TotalMemory();
00018
00023         float UsedMemory();
00024
00029         float Utilisation();
00030
00031     private:
00032         float total_memory;
00033
00034 };
00035
00036 #endif // MEMORY_H

```

5.6 power.h

```

00001 #ifndef POWER_H
00002 #define POWER_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include <map>
00007
00008 using std::string;
00009 using std::vector;
00010 using std::map;
00011
00016 class Power {
00017     public:
00018         Power();
00019
00024         double CurrHourEnergyUsage();
00025
00030         double CurrPowerUsage();
00031
00036         double TotalEnergyUsage();
00037
00042         vector<double> HoursEnergyUsages();
00043
00049         double HoursEnergyUsages(int hour);
00050
00054         void ResetLogVector();
00055
00060         void SetLogVector(vector<double> datas);
00061
00066         void UpdateLogVector(int hour);
00067
00071         void UpdatePrevHoursEnergy();

```

```

00072
00077     void SetExtra(double value);
00078
00082     void UpdatePowerAndEnergyUsage();
00083
00084     private:
00089         long long EnergyUsageInUj();
00090
00091         // The range of the energy counter
00092         static const long long MAX_ENERGY;
00093
00094         vector<double> hours_energy_usages;
00095         double curr_hour_energy_usage = 0;
00096         double total_energy_usage = 0; // in a day
00097         double curr_power_usage = 0;
00098
00099         // total energy usage (in uj) in the previous hours
00100         // since the system is booted
00101         long long prev_hours_energy = 0;
00102
00103         // total energy usage since the pc is booted
00104         long long accum_energy_usage = 0;
00105
00106         // current energy counter extracted from the system file
00107         long long energy = 0;
00108
00109         // energy counter extracted in the last logging time
00110         long long prev_energy = 0;
00111
00112         // number of times when the capped energy amount
00113         // is reached
00114         long long capped_times = 0;
00115
00116         // The extra energy usage in the current hour
00117         // (in case the pc is rebooted)
00118         double extra = 0;
00119 };
00120
00121 #endif

```

5.7 power_dao.h

```

00001 #ifndef POWER_DAO_H
00002 #define POWER_DAO_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include <map>
00007
00011 class PowerDAO {
00012
00013     public:
00019         void InitHoursLogFile(std::string curr_date, std::vector<double> usages);
00020
00024         void InitDaysLogFile();
00025
00030         std::string LastLoggedDate();
00031
00037         void UpdateDaysLogFile(std::string last_logged_date, double total_usage);
00038
00044         void UpdateHoursLogFile(std::string curr_date, std::vector<double> usages);
00045
00050         std::vector<double> HoursEnergyUsages();
00051
00057         std::map<std::string, double> LastNDaysEnergyUsage(int n);
00058
00059     private:
00060         static const std::string HOURS_LOG_FILE;
00061         static const std::string DAYS_LOG_FILE;
00062 };
00063
00064 #endif

```

5.8 process.h

```

00001 #ifndef PROCESS_H
00002 #define PROCESS_H
00003
00004 #include <string>

```

```

00005
00010 class Process {
00011     public:
00017         bool operator<(Process const& a) const;
00018
00024         bool operator>(Process const& a) const;
00025
00030         int Pid();
00031
00036         std::string User();
00037
00042         std::string Command();
00043
00048         float CpuUtilisation();
00049
00054         std::string Ram();
00055
00060         long int UpTime();
00061
00066         void SetPid(int pid);
00067
00072         void SetUser(int pid);
00073
00078         void SetCommand(int pid);
00079
00085         void SetCpuUtilisation(int pid, long curr_total_jiffies);
00086
00091         void SetRam(int pid);
00092
00097         void SetUpTime(int pid);
00098
00099     private:
00100         int pid;
00101         std::string user;
00102         std::string command;
00103         float cpu_utilisation;
00104         std::string ram;
00105         long up_time;
00106
00107         // previous cpu jiffies of a proces
00108         long prev_jiffies_on_process = 0;
00109         // previous cpu total jiffies
00110         long prev_total_jiffies = 0;
00111 };
00112
00113 #endif // PROCESS_H

```

5.9 processor.h

```

00001 #ifndef PROCESSOR_H
00002 #define PROCESSOR_H
00003
00004 #include <string>
00005 #include <vector>
00006
00011 class Processor {
00012     public:
00013         Processor();
00014
00015         int PhysicalCores();
00020
00021         int LogicalCores();
00026
00032         int HyperThreadedCores();
00033
00038         int ECores();
00039
00044         int PCores();
00045
00050         std::vector<float> Utilisations();
00051
00056         int Temperature();
00057
00061         void UpdateUtilisations();
00062
00063     private:
00064         static const int PHYSICAL_CORES;
00065         static const int LOGICAL_CORES;
00066         static const int HYPERTHREADED_CORES;
00067         static const int E_CORES;
00068         static const int P_CORES;
00069

```

```

00070         // utilisation of all cpu cores
00071         std::vector<float> utilisations;
00072         // previous jiffies when cpu is not idle
00073         std::vector<long> prev_active_jiffies;
00074         // previous jiffies when cpu is in all states
00075         std::vector<long> prev_total_jiffies;
00076     };
00077
00078 #endif

```

5.10 system.h

```

00001 #ifndef SYSTEM_H
00002 #define SYSTEM_H
00003
00004 #include <unistd.h>
00005
00006 #include <string>
00007 #include <vector>
00008 #include <map>
00009
00010 #include "include/processor.h"
00011 #include "include/memory.h"
00012 #include "include/process.h"
00013 #include "include/power.h"
00014 #include "include/power_dao.h"
00015
00016 using std::string;
00017 using std::vector;
00018 using std::map;
00019
00025 class System {
00026
00027     public:
00033         static System* Instance();
00034
00039         string OperatingSystem();
00040
00045         string Kernel();
00046
00051         long UpTime();
00052
00057         int TotalProcesses();
00058
00063         int RunningProcesses();
00064
00069         float TotalMemory();
00070
00075         float UsedMemory();
00076
00081         float MemoryUtilisation();
00082
00087         int CpuTemperature();
00088
00093         vector<float> CpuUtilisations();
00094
00099         vector<Process> SortedProcesses();
00100
00105         double PowerUsage();
00106
00111         vector<double> HoursEnergyUsages();
00112
00117         double TotalEnergyUsage();
00118
00123         map<string, double> LastWeekEnergyUsage();
00124
00129         double TotalEnergyUsageLastWeek();
00130
00134         void BindToPCores();
00135
00139         void BindToAllCores();
00140
00144         void BindToECores();
00145
00149         void BindToPAndECores();
00150
00155         void SetEnergyCap(double cap);
00156
00161         double EnergyCap();
00162
00163     private:
00164         System();
00165

```

```

00169         void UpdateProcesses();
00170
00171         void UpdateEnergy();
00172
00173         void SetUpEnergyDataAndFiles();
00174
00179         vector<int> CpuConsumingProcesses();
00180
00187         void BindProcesses(vector<int> pids, int low, int high);
00188
00189         static System* instance;
00190
00191         Processor cpu;
00192         Memory memory;
00193         map<int, Process> processes;
00194         string operating_system;
00195         string kernel;
00196
00197         Power power;
00198         PowerDAO dao;
00199         double energy_cap = 500; // by default in Wh
00200
00201         int hour;
00202     };
00203
00204 #endif // SYSTEM_H

```

5.11 system_parser.h

```

00001 #ifndef SYSTEM_PARSER_H
00002 #define SYSTEM_PARSER_H
00003
00004 #include <string>
00005 #include <vector>
00006
00007 class SystemParser {
00008
00009     public:
00010         // System
00011         static std::string OperatingSystem();
00012         static std::string Kernel();
00013         static std::vector<int> Pids();
00014         static std::vector<float> MemoryInfo();
00015         static float TotalMemory();
00016         static float AvalMemory();
00017         static int TotalProcesses();
00018         static int RunningProcesses();
00019         static long UpTime();
00020
00021         // CPU
00022         static std::vector<long> CpuTimes(int cid);
00023         static long TotalJiffies(int cid);
00024         static long IdleJiffiesC(int cid);
00025         static long ActiveJiffiesC(int cid);
00026
00027         // Processes
00028         static long ActiveJiffiesP(int pid);
00029         static std::string Command(int pid);
00030         static std::string Ram(int pid);
00031         static std::string Uid(int pid);
00032         static std::string User(int pid);
00033         static long int UpTime(int pid);
00034
00035     private:
00036         // Utils
00037         static std::string KeyValParser(std::string, std::string);
00038
00039         // Paths
00040         static const std::string kProcDirectory;
00041         static const std::string kCmdlineFilename;
00042         static const std::string kCpuinfoFilename;
00043         static const std::string kStatusFilename;
00044         static const std::string kStatFilename;
00045         static const std::string kUptimeFilename;
00046         static const std::string kMeminfoFilename;
00047         static const std::string kVersionFilename;
00048         static const std::string kOSPath;
00049         static const std::string kPasswordPath;
00050     };
00051
00052 #endif // SYSTEM_PARSER_H

```


Index

- BindToAllCores
 - System, [28](#)
- BindToECores
 - System, [28](#)
- BindToPAndECores
 - System, [28](#)
- BindToPCores
 - System, [28](#)
- Command
 - Process, [21](#)
- CpuTemperature
 - System, [28](#)
- CpuUtilisation
 - Process, [21](#)
- CpuUtilisations
 - System, [28](#)
- CurrentDate
 - DateTime, [8](#)
- CurrentTime
 - DateTime, [9](#)
- CurrHourEnergyUsage
 - Power, [15](#)
- CurrPowerUsage
 - Power, [15](#)
- Date
 - Format, [11](#)
- DateTime, [8](#)
 - CurrentDate, [8](#)
 - CurrentTime, [9](#)
 - Day, [9](#)
 - Hour, [9](#)
 - Min, [9](#)
 - Month, [10](#)
 - Sec, [10](#)
 - Year, [10](#)
- Day
 - DateTime, [9](#)
- Decimal
 - Format, [11](#)
- ECores
 - Processor, [25](#)
- ElapsedTime
 - Format, [12](#)
- EnergyCap
 - System, [29](#)
- exec
 - raymii::Command, [7](#)

- Format, [11](#)
 - Date, [11](#)
 - Decimal, [11](#)
 - ElapsedTime, [12](#)
 - Percentage, [12](#)
 - Time, [12](#)
- Hour
 - DateTime, [9](#)
- HoursEnergyUsages
 - Power, [15](#), [16](#)
 - PowerDAO, [18](#)
 - System, [29](#)
- HyperThreadedCores
 - Processor, [25](#)
- [include/command.h](#), [35](#)
- [include/date_time.h](#), [36](#)
- [include/format.h](#), [37](#)
- [include/mainwindow.h](#), [37](#)
- [include/memory.h](#), [38](#)
- [include/power.h](#), [38](#)
- [include/power_dao.h](#), [39](#)
- [include/process.h](#), [39](#)
- [include/processor.h](#), [40](#)
- [include/system.h](#), [41](#)
- [include/system_parser.h](#), [42](#)
- InitDaysLogFile
 - PowerDAO, [18](#)
- InitHoursLogFile
 - PowerDAO, [18](#)
- Instance
 - System, [29](#)
- Kernel
 - System, [29](#)
- LastLoggedDate
 - PowerDAO, [19](#)
- LastNDaysEnergyUsage
 - PowerDAO, [19](#)
- LastWeekEnergyUsage
 - System, [30](#)
- LogicalCores
 - Processor, [25](#)
- MainWindow, [13](#)
- Memory, [13](#)
 - TotalMemory, [14](#)
 - UsedMemory, [14](#)
 - Utilisation, [14](#)

- MemoryUtilisation
 - System, 30
- Min
 - DateTime, 9
- Month
 - DateTime, 10
- OperatingSystem
 - System, 30
- operator<
 - Process, 21
- operator>
 - Process, 21
- PCores
 - Processor, 26
- Percentage
 - Format, 12
- PhysicalCores
 - Processor, 26
- Pid
 - Process, 22
- Power, 15
 - CurrHourEnergyUsage, 15
 - CurrPowerUsage, 15
 - HoursEnergyUsages, 15, 16
 - ResetLogVector, 16
 - SetExtra, 16
 - SetLogVector, 16
 - TotalEnergyUsage, 17
 - UpdateLogVector, 17
 - UpdatePowerAndEnergyUsage, 17
 - UpdatePrevHoursEnergy, 17
- PowerDAO, 18
 - HoursEnergyUsages, 18
 - InitDaysLogFile, 18
 - InitHoursLogFile, 18
 - LastLoggedDate, 19
 - LastNDaysEnergyUsage, 19
 - UpdateDaysLogFile, 19
 - UpdateHoursLogFile, 20
- PowerUsage
 - System, 30
- Process, 20
 - Command, 21
 - CpuUtilisation, 21
 - operator<, 21
 - operator>, 21
 - Pid, 22
 - Ram, 22
 - SetCommand, 22
 - SetCpuUtilisation, 23
 - SetPid, 23
 - SetRam, 23
 - SetUpTime, 23
 - SetUser, 24
 - UpTime, 24
 - User, 24
- Processor, 25
 - ECores, 25
 - HyperThreadedCores, 25
 - LogicalCores, 25
 - PCores, 26
 - PhysicalCores, 26
 - Temperature, 26
 - UpdateUtilisations, 26
 - Utilisations, 27
- Ram
 - Process, 22
- raymii::Command, 7
 - exec, 7
- raymii::CommandResult, 8
- ResetLogVector
 - Power, 16
- RunningProcesses
 - System, 31
- Sec
 - DateTime, 10
- SetCommand
 - Process, 22
- SetCpuUtilisation
 - Process, 23
- SetEnergyCap
 - System, 31
- SetExtra
 - Power, 16
- SetLogVector
 - Power, 16
- SetPid
 - Process, 23
- SetRam
 - Process, 23
- SetUpTime
 - Process, 23
- SetUser
 - Process, 24
- SortedProcesses
 - System, 31
- System, 27
 - BindToAllCores, 28
 - BindToECores, 28
 - BindToPAndECores, 28
 - BindToPCores, 28
 - CpuTemperature, 28
 - CpuUtilisations, 28
 - EnergyCap, 29
 - HoursEnergyUsages, 29
 - Instance, 29
 - Kernel, 29
 - LastWeekEnergyUsage, 30
 - MemoryUtilisation, 30
 - OperatingSystem, 30
 - PowerUsage, 30
 - RunningProcesses, 31
 - SetEnergyCap, 31
 - SortedProcesses, 31

- TotalEnergyUsage, [31](#)
- TotalEnergyUsageLastWeek, [32](#)
- TotalMemory, [32](#)
- TotalProcesses, [32](#)
- UpTime, [32](#)
- UsedMemory, [33](#)
- SystemParser, [33](#)
- Temperature
 - Processor, [26](#)
- Time
 - Format, [12](#)
- TotalEnergyUsage
 - Power, [17](#)
 - System, [31](#)
- TotalEnergyUsageLastWeek
 - System, [32](#)
- TotalMemory
 - Memory, [14](#)
 - System, [32](#)
- TotalProcesses
 - System, [32](#)
- UpdateDaysLogFile
 - PowerDAO, [19](#)
- UpdateHoursLogFile
 - PowerDAO, [20](#)
- UpdateLogVector
 - Power, [17](#)
- UpdatePowerAndEnergyUsage
 - Power, [17](#)
- UpdatePrevHoursEnergy
 - Power, [17](#)
- UpdateUtilisations
 - Processor, [26](#)
- UpTime
 - Process, [24](#)
 - System, [32](#)
- UsedMemory
 - Memory, [14](#)
 - System, [33](#)
- User
 - Process, [24](#)
- Utilisation
 - Memory, [14](#)
- Utilisations
 - Processor, [27](#)
- Year
 - DateTime, [10](#)