|  |
| --- |
| Сопроводительный документ |
| RegistryManager |
| https://github.com/BadCatsss/RegistryManager |

Оглавление

[Введение 2](#_Toc26533853)

[Приложение RegistryManager 2](#_Toc26533854)

[Назначение приложения 2](#_Toc26533855)

[Особенности использования и работы 2](#_Toc26533856)

[Выполняемые операции 2](#_Toc26533857)

[Операция чтения 2](#_Toc26533858)

[Операция записи 4](#_Toc26533859)

[Пример работы приложения 4](#_Toc26533860)

[Технические подробности 6](#_Toc26533861)

[Файл RegistryManager.h 6](#_Toc26533862)

[Подробности реалезации 7](#_Toc26533863)

[Файл RegistryManager.cpp 8](#_Toc26533864)

[Подробности реалезации 13](#_Toc26533865)

# Введение

## Приложение RegistryManager

## Назначение приложения

Приложение RegistryManager создано с целью предоставления возможности произведения операций чтения и/или записи значений в иерархической базе данных в среде ОС Microsoft Windows – именуемой «Системный реестр» или же – «Реестр Windows».

# Особенности использования и работы

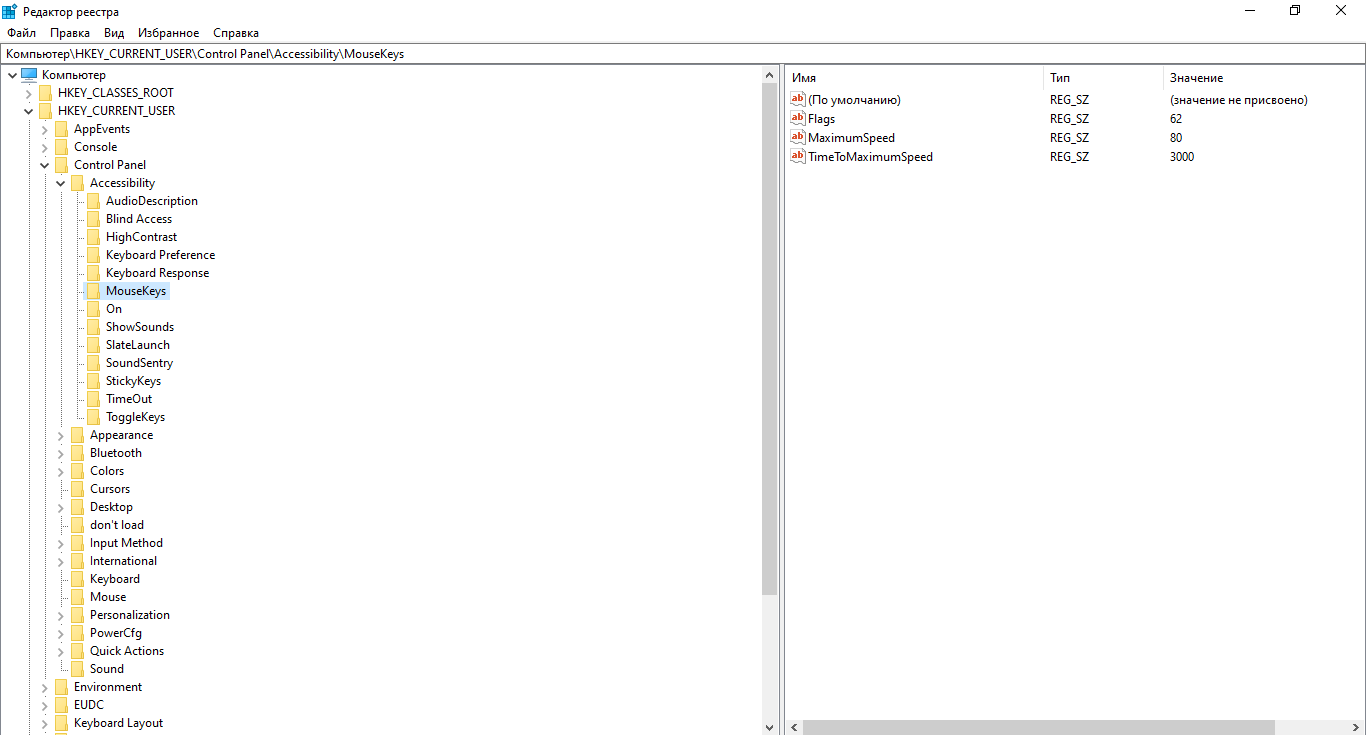
Приложение представляет собой реализацию шаблона проектирования Singelton, точнее, его усовершенствованной версии - синглтон Майерса.

## Выполняемые операции

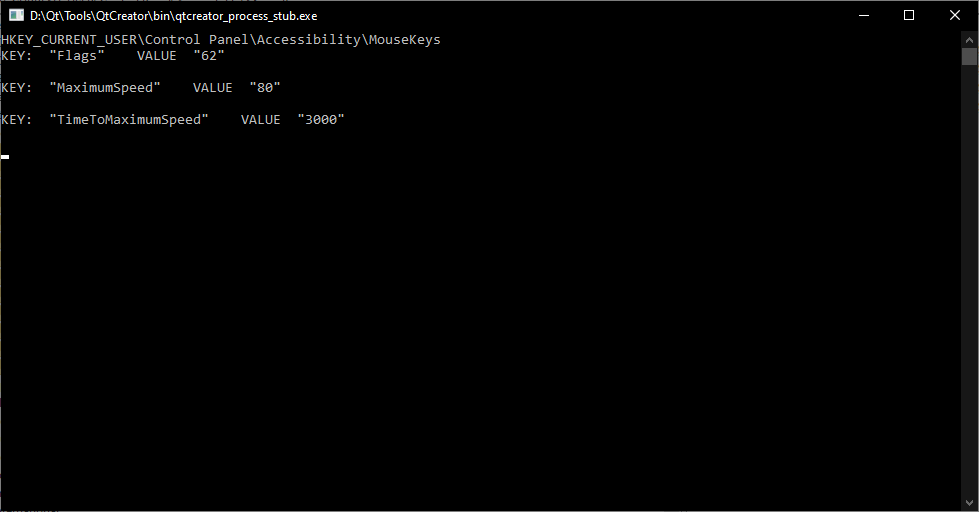
### Операция чтения

Операция чтения – производит чтение значения указанной записи из указанной ветки реестра. В случае, если указан корректный, но «частичный» путь – т.е такой, который не указывает на конкретную запись, а указывает на дочернюю «ветку»(«подветки») указанной «ветки» – в окно консоли будут выведены значения всех ключей и значений всех «подветок» - прелставляя тем самым – полную глубину вложенности в иерархической структуре.

Пример иерархической структуры :



Выводимые значения:



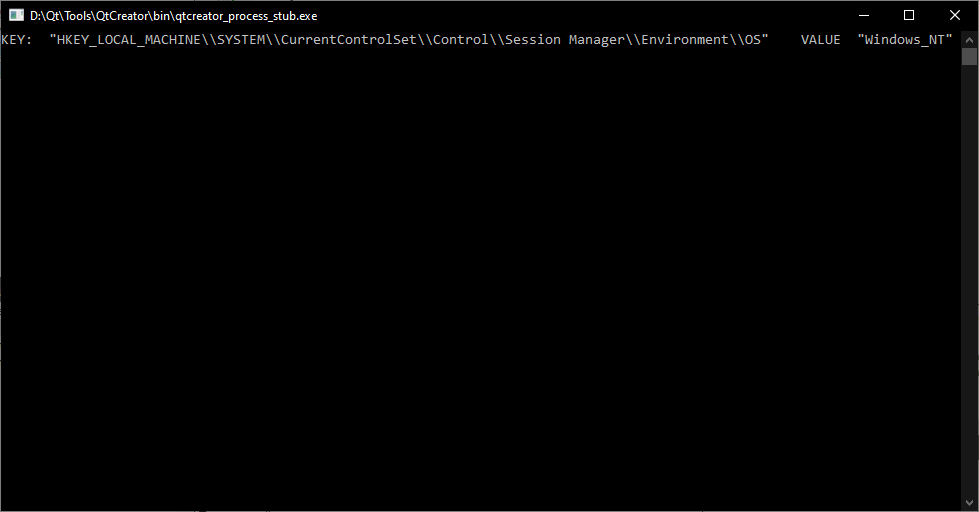
В таком случае, в качестве возвращаемого значения функции read() – выступает первое из выведенных пар «ключ-значение».

### Операция записи

Операция записи – производит запись указанного значения для конкретной записи из определенной ветки реестра. В случае, если в указанной ветке -отсутствует требуемая запись – она будет создана автоматически, после чего получит указанное значение. В ином случае, если требуемая запись имеется в реестре – ее значение будет перезаписано.

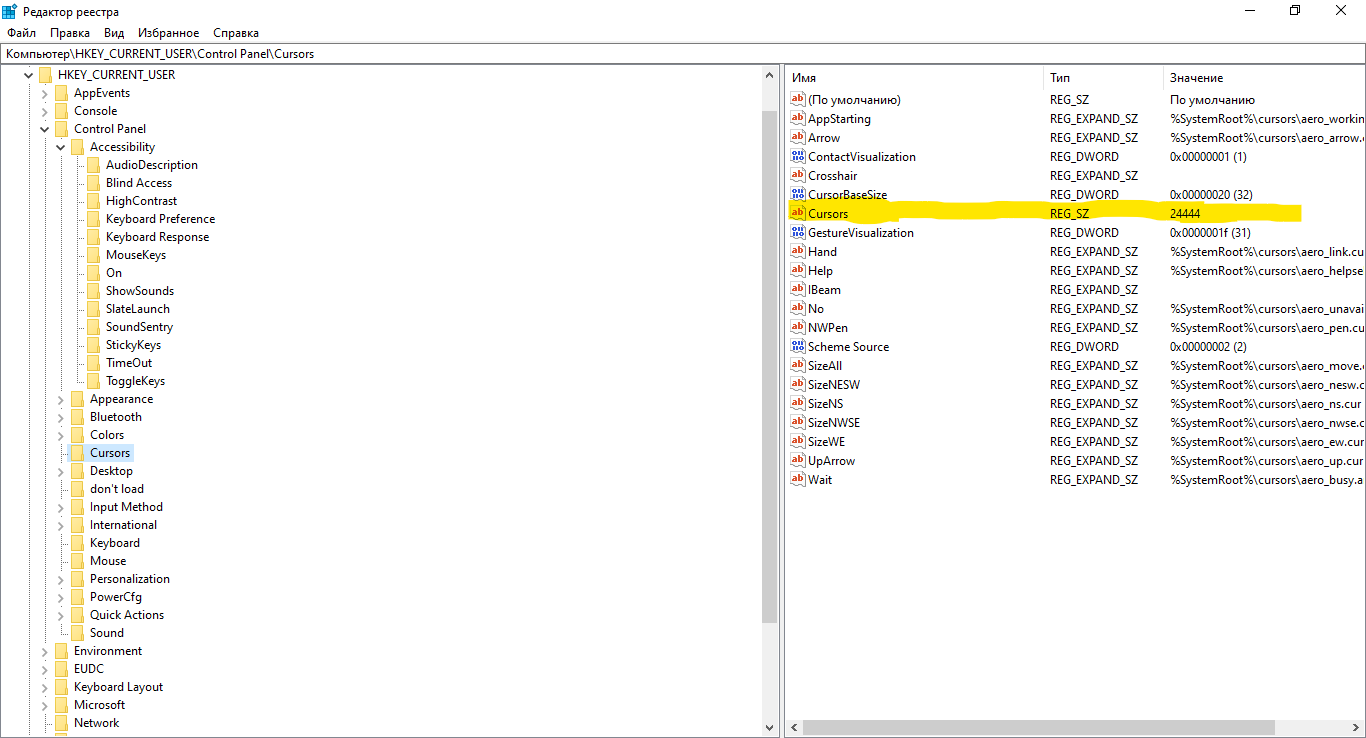
## Пример работы приложения

Чтение:

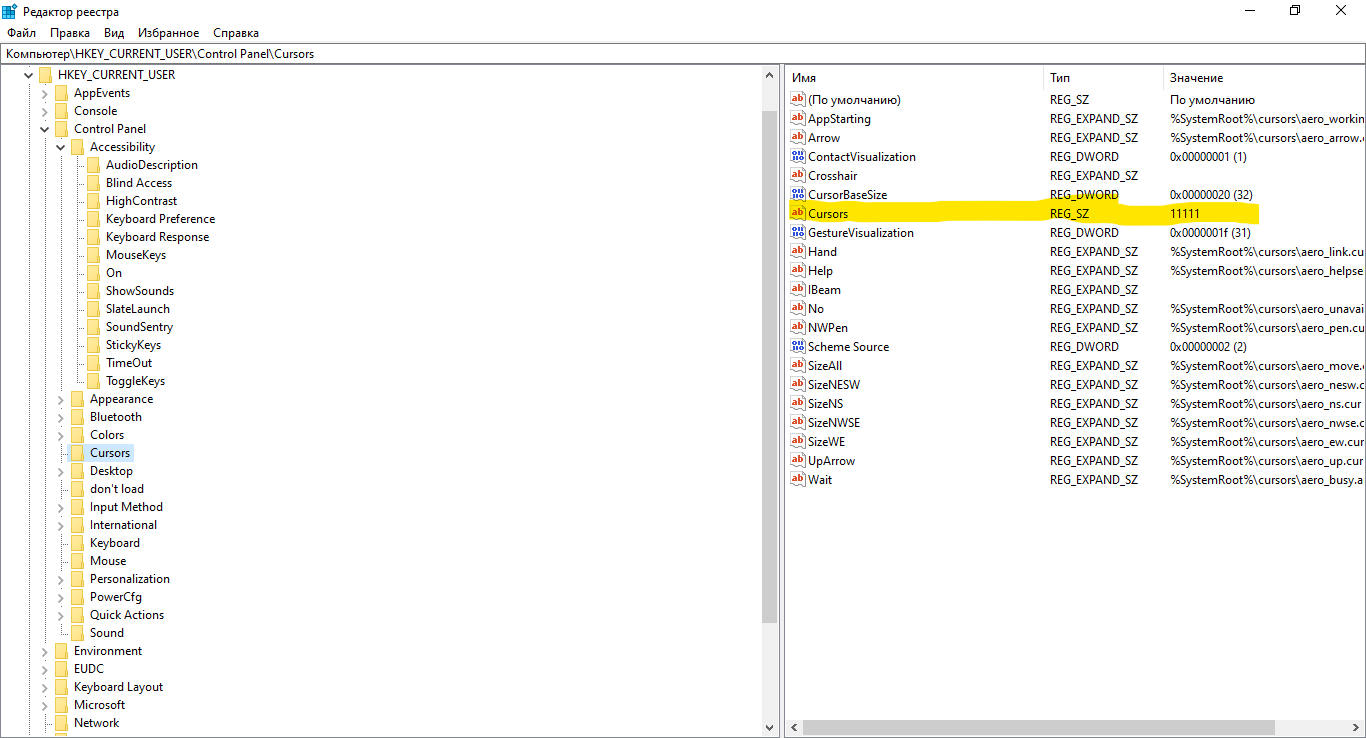


Запись:

Значение в реестре до выполнения операции записи:



Значение в реестре после выполнения операции записи:



# Технические подробности

## Файл RegistryManager.h

#pragma once

#include <qstring.h>

#include <thread>

#include <QMutex>

#include <QSystemSemaphore>

#include <QSharedMemory>

#include <QSettings>

#include<QDir>

#include <QtDebug>

class **RegistryManager**

{

private:

**RegistryManager**(const RegistryManager& param) = delete;

**RegistryManager**();

RegistryManager& operator = (RegistryManager& param) = delete ;

bool **checkHiveKeyCorrectnes**(const QString& path);

bool **checkKeyCorrectnes**(const QString& path);

void **addErrorToList**(const QString& param);

const QStringList& **getErrorsList**();

QVector<QString> hiveKeys;

QStringList errorsList;

QStringList pathParts;

public:

static RegistryManager& **instance**();

bool **write**(const QString& key, const QString& value);

const QString **read**(const QString& key);

void **printErrorslist**();

};

### Подробности реалезации

**RegistryManager**(const RegistryManager& param) = delete; - Запрет (удаление конструктора копирования).

**RegistryManager**(); - конструктор по-умолчанию

RegistryManager& operator = (RegistryManager& param) = delete ; - Запрет копирования через оператор присваивания.

bool **checkHiveKeyCorrectnes**(const QString& path); - Метод для первичной «грубой» проверки корректности переданного в функции чтения\записи – пути.

bool **checkKeyCorrectnes**(const QString& path); - Метод для проверки корректности переданного в функции чтения\записи – пути.

void **addErrorToList**(const QString& param); - Метод, добавляющий возможные возникшие ошибки в список, который будет выведен перед завершением программы, если таковы имели место быть.

const QStringList& **getErrorsList**(); - Метод, возвращающий список ошибок.

QVector<QString> hiveKeys; - Массив значений, представленный в виде QVector – для хранения названий 5 основных веток реестра:

"HKEY\_CLASSES\_ROOT"

"HKEY\_CURRENT\_USER"

"HKEY\_LOCAL\_MACHINE"

"HKEY\_USERS"

"HKEY\_CURRENT\_CONFIG"

Используется при первичной, «грубой» проверке.

QStringList errorsList; - список, хранящий возможные, возникши ошибки при выпонении операций чтения/записи.

QStringList pathParts; - список, в котором будут хранится лексемы из указанной строки – пути в ветви реестра, разделенные по токену «слеш» - пример:

"HKEY\_LOCAL\_MACHINE\\SYSTEM\\CurrentControlSet\\Control"

Лексемы:

* HKEY\_LOCAL\_MACHINE
* SYSTEM
* CurrentControlSet
* Control

public:

static RegistryManager& **instance**(); - Метод, реализующий паттерн «Синглтон Майерса».

bool **write**(const QString& key, const QString& value); - Метод, выпоняющий операцию записи.

const QString **read**(const QString& key); - - Метод, выпоняющий операцию чтения.

void **printErrorslist**(); - Метод, производящий вывод возможных, возникших ошибок в консоль.

## Файл RegistryManager.cpp

#include "RegistryManager.h"

RegistryManager::**RegistryManager**()

{

this->hiveKeys.push\_back("HKEY\_CLASSES\_ROOT");

this->hiveKeys.push\_back("HKEY\_CURRENT\_USER");

this->hiveKeys.push\_back("HKEY\_LOCAL\_MACHINE");

this->hiveKeys.push\_back("HKEY\_USERS");

this->hiveKeys.push\_back("HKEY\_CURRENT\_CONFIG");

}

RegistryManager& RegistryManager::**instance**()

{

static RegistryManager instance;

return instance;

}

bool RegistryManager::**checkHiveKeyCorrectnes**(const QString& path )

{

for (int i = 0; i < 5; ++i)

{

if ( path.toStdString().find( RegistryManager::hiveKeys[i].toStdString() ) != std::string::npos )

{

return true;

}

}

addErrorToList("HiveKey is incorrect");

return false;

}

bool RegistryManager::**checkKeyCorrectnes**(const QString& path)

{

pathParts = path.split( QRegExp( "[/\\\\]" ) );

QString pathh = pathParts[0];

if (pathParts.size() >= 2)

{

for (int var = 1; var < pathParts.size(); ++var)

{

QSettings st(pathh, QSettings::NativeFormat);

if ( st.childGroups().contains( pathParts[var] ) || st.childKeys().contains( pathParts[var] ) )

{

if ( ! ( st.childGroups().contains( pathParts[var] ) && st.childKeys().contains( pathParts[var] ) && var == pathParts.length() - 1 ) )

{

//если st.childGroups().contains(list[var]) && st.childKeys().contains(list[var])

//например: HKEY\_CURRENT\_USER\Control Panel\Cursors\Cursors\Cursors

// - иначе, если так, отдаем предпочтение каталогу и подкаталогу - не добавляя "\\" - и не переходя к одноименной записи

//var == list.length() - 1 - указан файл - т.е далее, нет подразделов в пути

pathh = pathh + "\\" + pathParts[var];

}

else if(st.childGroups().contains( pathParts[var] ) || var == pathParts.length() -1 )

{

if( st.allKeys().contains( pathParts[var] ) || st.allKeys().size() != 0)

{

return true;

}

return false;

}

else

{

addErrorToList("Key or subkey is wrong. Cant write value");

return false;

}

}

else return false;

}

return true;

}

addErrorToList("Too short path");

return false;

}

const QStringList& RegistryManager::**getErrorsList**()

{

return errorsList;

}

void RegistryManager::**addErrorToList**(const QString &param)

{

errorsList.push\_back(param);

}

void RegistryManager::**printErrorslist**()

{

if ( RegistryManager::instance().getErrorsList().length() != 0 )

{

for ( auto& i : RegistryManager::instance().getErrorsList() )

{

qWarning() << i << endl;

}

}

}

bool RegistryManager::**write**(const QString& key, const QString& value)

{

QSettings settings(QString::fromStdString(key.toStdString().erase( key.toStdString().find\_last\_of("\\"),key.length()) ),QSettings::NativeFormat);

if ( checkHiveKeyCorrectnes(key) )

{

if ( checkKeyCorrectnes(key) || !settings.contains(value))

{

QString slicePart = pathParts[0];

for (int i = 1; i < pathParts.size() - 1; ++i)

{

slicePart= slicePart + "\\" + pathParts[i] ;

}

QSettings settings1(slicePart,QSettings::NativeFormat);

settings1.setValue(QString::fromStdString(key.toStdString().erase(0, key.toStdString().find\_last\_of("\\") + 1) ),value);

}

else

{

addErrorToList("Key or subkey is wrong. Cant write value");

return false;

}

}

else

{

addErrorToList("Hive Key is wrong. Cant write value");

return false;

}

return true;

}

const QString RegistryManager::**read**(const QString& key)

{

QSettings settings(key, QSettings::NativeFormat);

if ( checkHiveKeyCorrectnes( key ) )

{

if ( checkKeyCorrectnes( key ) )

{

QMap<QString, QString> registryBranch;

QStringList keyList = settings.allKeys();

for ( int i = 0; i < keyList.size(); i++ )

{

registryBranch.insert(keyList[i], settings.value(keyList[i]).toString());

}

for (int var = 0; var < registryBranch.size(); ++var)

{

qDebug() << "KEY: " << keyList[var] << " " << "VALUE " << registryBranch.find( keyList[var] ).value() << endl;

}

if (keyList.length() == 0)

{

QString slicePart = pathParts[0];

for (int i = 1; i < pathParts.size() - 1; ++i)

{

slicePart= slicePart + "\\" + pathParts[i] ;

}

QSettings settings1(slicePart, QSettings::NativeFormat);

QStringList keyList1 = settings1.allKeys();

for(int i = 0; i < keyList1.size(); i++)

{

registryBranch.insert( keyList1[i], settings.value( keyList1[i] ).toString() );

}

qDebug() << "KEY: " << key << " " << "VALUE " << settings1.value(pathParts[ pathParts.size() - 1 ], QSettings::NativeFormat).toString()<< endl;

return QString ( settings1.value(pathParts[ pathParts.size() - 1 ], QSettings::NativeFormat).toString() );

}

//qDebug() << "KEY: " << keyList[0] << " " << "VALUE " << registryBranch.find(keyList[0]).value()<< endl;

return QString ( registryBranch.find( keyList[0] ).value() );

}

else

{

addErrorToList("Key or subkey is wrong. Cant read value");

}

}

else

{

addErrorToList("Hive Key is wrong. Cant write value");

}

}

### Подробности реалезации

#include "RegistryManager.h"

#### **RegistryManager()**

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| private | Нет | Нет |

RegistryManager::**RegistryManager**()

{

this->hiveKeys.push\_back("HKEY\_CLASSES\_ROOT");

this->hiveKeys.push\_back("HKEY\_CURRENT\_USER");

this->hiveKeys.push\_back("HKEY\_LOCAL\_MACHINE");

this->hiveKeys.push\_back("HKEY\_USERS");

this->hiveKeys.push\_back("HKEY\_CURRENT\_CONFIG");

}

– конструктор класса **RegistryManager**

#### **RegistryManager& instance()**

- Метод, реализующий паттерн «Синглтон Майерса».

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| public | Нет | RegistryManager& |

RegistryManager& RegistryManager::**instance**()

{

static RegistryManager instance;

return instance;

}

#### **bool checkHiveKeyCorrectnes(const QString& path)**

- Метод для первичной «грубой» проверки корректности переданного в функции чтения\записи – пути.

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| private | const QString& path | bool |

bool RegistryManager::**checkHiveKeyCorrectnes**(const QString& path )

{

for (int i = 0; i < 5; ++i)

{

if ( path.toStdString().find( RegistryManager::hiveKeys[i].toStdString() ) != std::string::npos )

{

return true;

}

}

addErrorToList("HiveKey is incorrect");

return false;

}

#### **bool checkKeyCorrectnes(const QString& path)**

- Метод для проверки корректности переданного в функции чтения\записи – пути.

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| private | const QString& path | bool |

bool RegistryManager::**checkKeyCorrectnes**(const QString& path)

{

pathParts = path.split( QRegExp( "[/\\\\]" ) );

QString pathh = pathParts[0];

if (pathParts.size() >= 2)

{

for (int var = 1; var < pathParts.size(); ++var)

{

QSettings st(pathh, QSettings::NativeFormat);

if ( st.childGroups().contains( pathParts[var] ) || st.childKeys().contains( pathParts[var] ) )

{

if ( ! ( st.childGroups().contains( pathParts[var] ) && st.childKeys().contains( pathParts[var] ) && var == pathParts.length() - 1 ) )

{

//если st.childGroups().contains(list[var]) && st.childKeys().contains(list[var])

//например: HKEY\_CURRENT\_USER\Control Panel\Cursors\Cursors\Cursors

// - иначе, если так, отдаем предпочтение каталогу и подкаталогу - не добавляя "\\" - и не переходя к одноименной записи

//var == list.length() - 1 - указан файл - т.е далее, нет подразделов в пути

pathh = pathh + "\\" + pathParts[var];

}

else if(st.childGroups().contains( pathParts[var] ) || var == pathParts.length() -1 )

{

if( st.allKeys().contains( pathParts[var] ) || st.allKeys().size() != 0)

{

return true;

}

return false;

}

else

{

addErrorToList("Key or subkey is wrong. Cant write value");

return false;

}

}

else return false;

}

return true;

}

addErrorToList("Too short path");

return false;

}

#### **const QStringList& getErrorsList()**

- Метод, возвращающий список ошибок.

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| private | Нет | QStringList& |

const QStringList& RegistryManager::**getErrorsList**()

{

return errorsList;

}

#### **void addErrorToList(const QString &param)**

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| private | const QString &param | void |

void RegistryManager::**addErrorToList**(const QString &param)

{

errorsList.push\_back(param);

}

#### **void printErrorslist()**

- Метод, добавляющий возможные возникшие ошибки в список, который будет выведен перед завершением программы, если таковы имели место быть.

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| public | Нет | Void |

void RegistryManager::**printErrorslist**()

{

if ( RegistryManager::instance().getErrorsList().length() != 0 )

{

for ( auto& i : RegistryManager::instance().getErrorsList() )

{

qWarning() << i << endl;

}

}

}

***bool write(const QString& key, const QString& value)***

- Метод, выпоняющий операцию записи.

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| public | const QString& key  const QString& value | bool |

bool RegistryManager::**write**(const QString& key, const QString& value)

{

QSettings settings(QString::fromStdString(key.toStdString().erase( key.toStdString().find\_last\_of("\\"),key.length()) ),QSettings::NativeFormat);

if ( checkHiveKeyCorrectnes(key) )

{

if ( checkKeyCorrectnes(key) || !settings.contains(value))

{

QString slicePart = pathParts[0];

for (int i = 1; i < pathParts.size() - 1; ++i)

{

slicePart= slicePart + "\\" + pathParts[i] ;

}

QSettings settings1(slicePart,QSettings::NativeFormat);

settings1.setValue(QString::fromStdString(key.toStdString().erase(0, key.toStdString().find\_last\_of("\\") + 1) ),value);

}

else

{

addErrorToList("Key or subkey is wrong. Cant write value");

return false;

}

}

else

{

addErrorToList("Hive Key is wrong. Cant write value");

return false;

}

return true;

}

***const QString read(const QString& key)***

- Метод, выпоняющий операцию чтения.

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Принимаемое значение | Возвращаемое значение |
| public | Нет | RegistryManager& |

const QString RegistryManager::**read**(const QString& key)

{

QSettings settings(key, QSettings::NativeFormat);

if ( checkHiveKeyCorrectnes( key ) )

{

if ( checkKeyCorrectnes( key ) )

{

QMap<QString, QString> registryBranch;

QStringList keyList = settings.allKeys();

for ( int i = 0; i < keyList.size(); i++ )

{

registryBranch.insert(keyList[i], settings.value(keyList[i]).toString());

}

for (int var = 0; var < registryBranch.size(); ++var)

{

qDebug() << "KEY: " << keyList[var] << " " << "VALUE " << registryBranch.find( keyList[var] ).value() << endl;

}

if (keyList.length() == 0)

{

QString slicePart = pathParts[0];

for (int i = 1; i < pathParts.size() - 1; ++i)

{

slicePart= slicePart + "\\" + pathParts[i] ;

}

QSettings settings1(slicePart, QSettings::NativeFormat);

QStringList keyList1 = settings1.allKeys();

for(int i = 0; i < keyList1.size(); i++)

{

registryBranch.insert( keyList1[i], settings.value( keyList1[i] ).toString() );

}

qDebug() << "KEY: " << key << " " << "VALUE " << settings1.value(pathParts[ pathParts.size() - 1 ], QSettings::NativeFormat).toString()<< endl;

return QString ( settings1.value(pathParts[ pathParts.size() - 1 ], QSettings::NativeFormat).toString() );

}

//qDebug() << "KEY: " << keyList[0] << " " << "VALUE " << registryBranch.find(keyList[0]).value()<< endl;

return QString ( registryBranch.find( keyList[0] ).value() );

}

else

{

addErrorToList("Key or subkey is wrong. Cant read value");

}

}

else

{

addErrorToList("Hive Key is wrong. Cant write value");

}

}