СОДЕРЖАНИЕ

[ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ 5](#_Toc136359114)

[ВВЕДЕНИЕ 6](#_Toc136359115)

[1 ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc136359116)

[2 ОБЗОР АНАЛОГОВ 8](#_Toc136359117)

[3 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ 10](#_Toc136359118)

[4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 14](#_Toc136359119)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 20](#_Toc136359120)

[ЗАКЛЮЧЕНИЕ 26](#_Toc136359121)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 27](#_Toc136359122)

[ПРИЛОЖЕНИЕ А 28](#_Toc136359123)

[ПРИЛОЖЕНИЕ Б 42](#_Toc136359124)

[ПРИЛОЖЕНИЕ В 43](#_Toc136359125)

[ПРИЛОЖЕНИЕ Г 44](#_Toc136359126)

# ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

Задание на курсовой проект – редактор реестра Windows. Такое приложение должно обладать удобным пользовательским интерфейсом и позволять пользователю проводить различные изменения в реестре Windows. К таким функциям можно просмотр всех разделов и параметров, создание разделов, создание параметров основных типов (REG\_SZ, REG\_DWORD, REG\_QWORD), редактирование параметров (переименование и изменения значения.

# ВВЕДЕНИЕ

Реестр как [древовидная иерархическая база данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) впервые появился в Windows 3.1. Это был всего один двоичный файл, который назывался REG.DAT. Одновременно c появлением реестра в Windows 3.1 появилась программа REGEDIT.EXE для просмотра и редактирования реестра. Эта версия имела только одну ветку, а именно HKEY\_CLASSES\_ROOT. Но у этой версии реестра был существенный недостаток: он имел ограничение на размер файла REG.DAT в 64 к5. Если реестр превышал этот размер, то его приходилось удалять и собирать из \*.REG файлов либо вводить данные вручную.

Свое развитие реестр получил в Windows NT 3.1. В этой версии реестр имел 4 корневых раздела: HKEY\_LOCAL\_MACHINE, HKEY\_CURRENT\_USER, HKEY\_CLASSES\_ROOT и HKEY\_USERS. Программа REGEDIT.EXE все также осталась, но позволяла редактировать только ветку HKEY\_CLASSES\_ROOT. Но появилась программа REGEDT32.EXE, которая позволяла редактировать все ветки реестра.

Далее технология и назначение реестра уже не менялись. Все последующие версии Windows (NT 3.5, 95, NT 4.0, 98, 2000, XP, Vista, 7, 8, 10) использовали реестр как основную БД, содержащую все основные данные по конфигурации как самой ОС, так и прикладных программ. Далее менялись названия файлов реестра и их расположение, а также название и назначение ключей.

# 1 ОБЗОР ЛИТЕРАТУРЫ

Реестр Windows, или системный реестр – это иерархически построенная база данных Windows, которая содержит информацию об аппаратном обеспечении, установленных программах и настройках, а также о профилях учетных записей персонального компьютера. Обычно изменять реестр вручную не требуется, поскольку приложения вносят все необходимые изменения автоматически.

Редактор реестра – это программа, позволяющая просматривать и редактировать реестр Windows. В редакторе реестра можно создавать разделы и параметры, а также их редактировать. Это может быть полезно в тех случаях, когда необходимо изменить какой-либо скрытый по умолчанию параметр в операционной системе.

Редактор реестра содержит 5 основных реестров. Раздел HKEY\_CURRENT\_USER содержит настройки текущего активного пользователя, вошедшего в систему. Раздел HKEY\_USERS содержит информацию о профилях всех пользователей данного компьютера. Раздел HKEY\_LOCAL\_MACHINE содержит параметры конфигурации, относящиеся к данному компьютеру (для всех пользователей). Раздел HKEY\_CLASSES\_ROOT является подразделом HKEY\_LOCAL\_MACHINE\ Software\Classes. В основном, содержит информацию о зарегистрированных типах файлов и объектах [COM](https://ru.wikipedia.org/wiki/Component_Object_Model) и [ActiveX](https://ru.wikipedia.org/wiki/ActiveX). Раздел HKEY CURRENT\_CONFIG содержит сведения о профиле оборудования, используемом локальным компьютером при запуске системы. Является ссылкой на HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Hardware\Profiles\ Current.

Для написания такой программы был выбран фреймворк Qt, так как он позволяет создать удобный графический интерфейс. Также было решено использовать WinAPI, так как этот программный интерфейс позволяет работать с реестром Windows.

Для написания логики приложения был выбран язык программирования C++, так как он обладает высокой скоростью и поддерживает парадигму объектно-ориентированного программирования.

# 2 ОБЗОР АНАЛОГОВ

В данном разделе приведены существующие аналоги программы, которую необходимо разработать в рамках данного курсового проекта.

**2.1 Regedit.exe**

Программа Regedit.exe – стандартное средство для работы с реестром Window. Она позволяет создавать разделы, параметры различных типов, а также редактировать их. Также программа позволяет сохранять все параметры реестра в файл, чтобы, в случае возникшей ошибки, вернуть настройки в предыдущее состояние. Интерфейс программы приведен на рисунке 2.1.1.

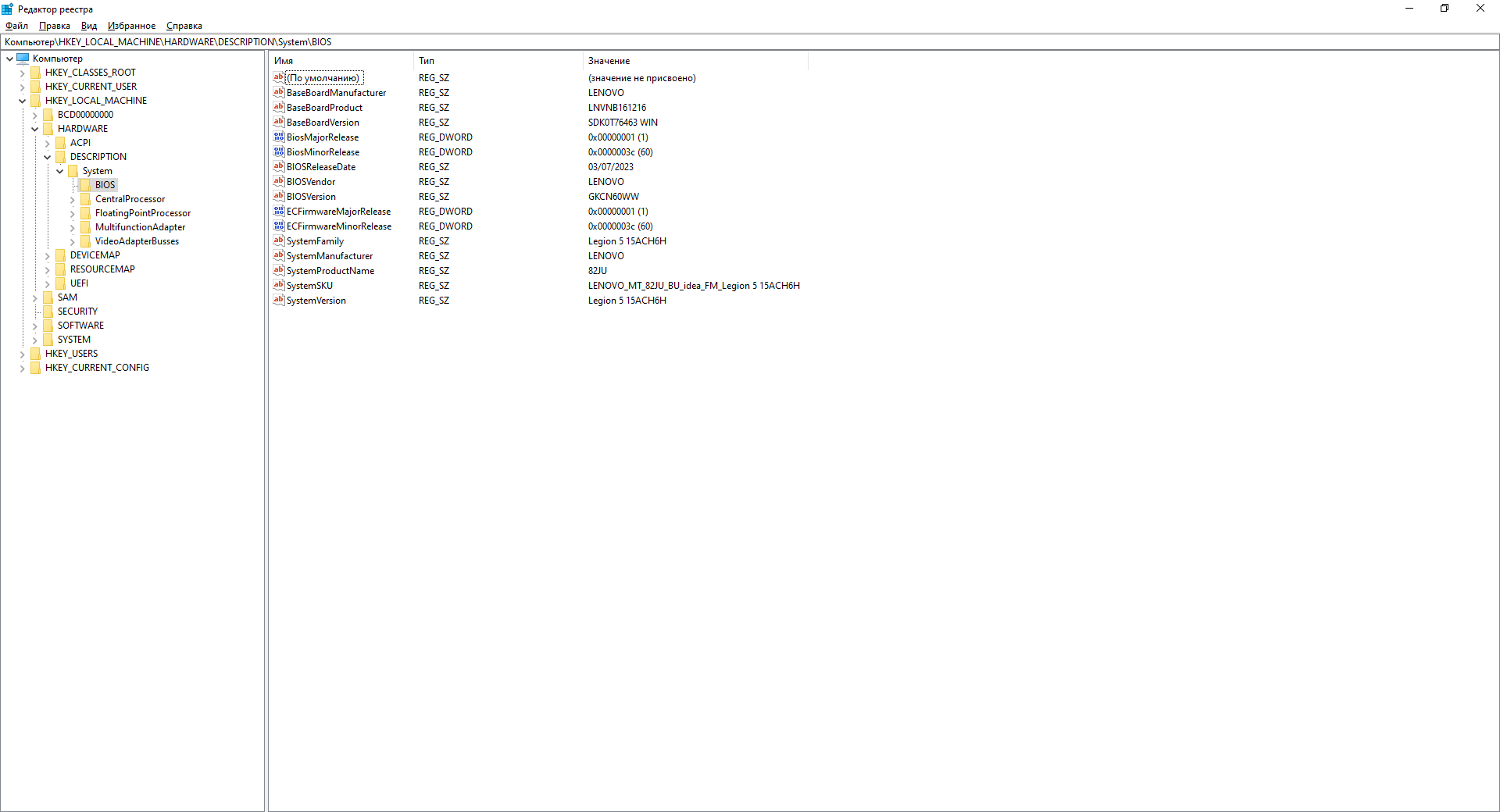


Рисунок 2.1.1 – Интерфейс приложения Regedit.exe.

**2.2 RegistryWorkshop**

RegistryWorkshop – стороння программа для работы с реестром. Она обладает всем функционалом стандартного редактор реестра с некоторыми улучшениями, которые опытным пользователям могут быть полезны. Например, по сравнению с Regedit.exe, поиск параметра в реестре происходит значительно быстрее и выдает все найденные параметры сразу, а не по-одному. Также RegistryWorkshop программа имеет несколько уровней отмены последних действий пользователя. Интерфейс программы приведен на рисунке 2.1.2.

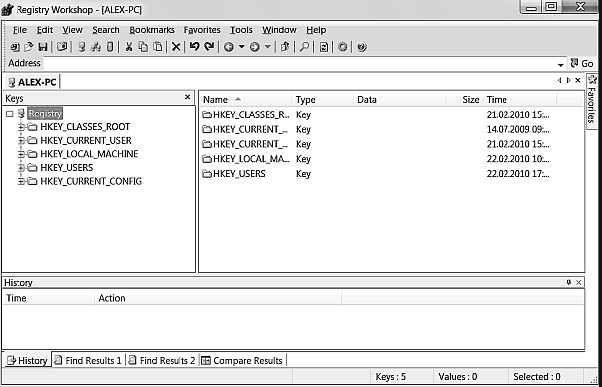


Рисунок 2.1.2 – Интерфейс приложения RegistryWorkshop.

# 3 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе приведено описание классов, созданных в процессе разработки программы для редактирования реестра Windows.

**3.1 MainWindow**

Данный класс наследуется от класса QMainWindow. Он является графическим представлением окна, в котором проходит работа приложения. В конструкторе вызываются методы, отвечающие за внешний вид окна.

Поля:

- Ui::MainWindow\* ui – указатель, позволяющий обращаться к виджетам главного окна;

- Registry registry – объект класса Registry, позволяющий работать с реестром.

Методы:

- void MainWindow::ConfigureMainWIndow() – задает атрибуты отображаемого окна;

- void MainWindow::ConfigureRegistryTreeWidget()– задает атрибуты виджету, который отвечает за представление разделов реестра;

- void MainWindow::ConfigureGroupKeysWidget() – задает атрибуты виджету, который отвечает за отображение параметров раздела реестра;

- void MainWindow::ConfigureCurrentPathWidget(QString path)– задает текст виджету, который отвечает за отображение текущего пути в дереве реестра;

- void MainWindow::ConfigureCreateButtons(QString path) – меняет состояние кнопок создания раздела или параметра в реестре на активное либо неактивное в зависимости от текущего пути в реестре;

- void MainWindow::ConfigureRemoveKeyButton(QTableWidgetItem \*currentItem) – меняет состояние кнопки удаления параметра из раздела на неактивное, если параметр не выбран;

- QTreeWidgetItem\* MainWindow::CreateRegistryBranch(QString groupName, QTreeWidgetItem\* parentItem) – отображает в дереве реестра подразделы с заданным именем и заданным родительским разделом;

- void MainWindow::on\_registryTree\_itemExpanded (QTreeWidgetItem \*group) – продолжает отображение ветки реестра при раскрытии подразделов текущего раздела;

- void MainWindow::on\_registryTree\_itemClicked (QTreeWidgetItem \*item, int column) – отображает параметры выбранного раздела;

- void MainWindow::on\_currentPath\_returnPressed() – при создании раздела открывает его;

- void MainWindow::on\_createKeyAction\_triggered() – вызывает диалоговое окно создания параметра;

- void MainWindow::on\_groupKeys\_itemClicked(QTableWidgetItem \*item)– вызывает метод, позволяющий удалить параметр в реестра при его выборе;

- void MainWindow::on\_removeKeyAction\_triggered()– удаляет параметр из реестра;

- void MainWindow::on\_createGroupAction\_triggered()– создает раздел в реестре;

- void MainWindow::on\_removeGroupAction\_triggered()– удаляет раздел из реестра;

- void MainWindow::on\_groupKeys\_cellDoubleClicked(int row, int column) – при двойном нажатии по параметру вызывает диалоговое окно для редактирование параметра.

**3.2 Registry**

Данный класс хранит в себе поля и методы для работы с реестром.

Поля:

- const QString PATH\_SEPARATOR = "\\" – разделитель для пути реестра.

Методы:

- void Registry::ParseRegistryBranch(QTreeWidgetItem \*root, QSettings \*settings, QIcon \*icon) – рекурсивно считывает подразделы из реестра для заданного корневого элементы и задает им иконку;

- QString Registry::FindPathForGroup(QTreeWidgetItem \*group) – находит путь до заданного раздела в реестре;

- QTreeWidgetItem \*Registry::FindGroupByPath(QString pathForGroup, QTreeWidgetItem\* registryTreeRoot) – находит раздел по указанному пути;

- QMap<QString, QString> Registry:: CreateGroupKeys(QTreeWidgetItem \*selectedGroup, QList<QTableWidgetItem\*>& valuesTypes) – создает пары вида параметр-значение и их типы;

- HKEY Registry::GetGroupHkeyByName(QString baseGroupName) – возвращает дескриптор базового раздела.

**3.3 CreateKeyWindow**

Данный класс наследуется от класса QDialog. Он является графическим представлением диалогового окна, в котором пользователь может создать новый параметр в реестре. В конструкторе вызываются методы, отвечающие за создание кнопок на окне.

Поля:

- Ui::CreateKeyWindow \*ui – указатель, позволяющий обращаться к виджетам диалогового окна;

- QString keyName – имя для нового параметра;

- QVariant keyValue – значение нового параметры и его тип.

Методы:

- void CreateKeyWindow::ConfigureWindow() – задает атрибуты отображаемого окна;

- void CreateKeyWindow::on\_okButton\_clicked()– проверяет правильность данных для нового параметры. В случае ошибки предупреждает пользователя о ней, иначе закрывает окно;

- QString CreateKeyWindow::GetKeyName() const – возвращает имя нового параметра;

- QVariant CreateKeyWindow::GetKeyValue() const– возвращает значение и тип нового параметра.

**3.4 CreateGroupWindow**

Данный класс наследуется от класса QDialog. Он является графическим представлением диалогового окна, в котором пользователь может создать новый раздел в реестре. В конструкторе вызываются методы, отвечающие за создание кнопок на окне.

Поля:

- Ui::CreateGroupWindow \*ui – указатель, позволяющий обращаться к виджетам диалогового окна;

- QString currentPath – путь до текущего раздела;

- QString groupName – имя нового раздела.

Методы:

- void CreateGroupWindow::ConfigureWindow() – задает атрибуты отображаемого окна;

- void CreateGroupWindow::on\_okButton\_clicked() - проверяет правильность данных для нового раздела. В случае ошибки предупреждает пользователя о ней, иначе закрывает окно;

- QString CreateGroupWindow::GetGroupName() const– возвращает имя нового раздела.

**3.5 ChangeKeyWindow**

Данный класс наследуется от класса QDialog. Он является графическим представлением диалогового окна, в котором пользователь может изменить имя либо значение выбранного параметра. В конструкторе вызываются методы, отвечающие за создание кнопок на окне.

Поля:

- Ui::CreateGroupWindow \*ui – указатель, позволяющий обращаться к виджетам диалогового окна;

- QString keyName – имя параметры;

- QString\* stringValue – указатель, хранящий значения параметра типа REG\_SZ;

- uint\* intValue – указатель, хранящий значение параметра типа DWORD;

**-**qulonglong\* longValue **–** указатель, хранящий значение параметра типа QWORD**.**

Методы:

- void ChangeKeyWindow::ConfigureWindow() – задает атрибуты отображаемого окна;

- void ChangeKeyWindow::on\_okButton\_clicked() - проверяет правильность данных для выбранного параметры и в случае ошибки предупреждает пользователя о ней;

- void ChangeKeyWindow::ConfigureLineEdits() – отображает

текущее имя и значение параметра в виджете;

- QString \*ChangeKeyWindow::GetStringValue() const – возвращает параметр типа REG\_SZ;

- void ChangeKeyWindow::SetStringValue(QString \*newStringValue) – устанавливает параметр типа REG\_SZ;

- uint \*ChangeKeyWindow::GetIntValue() const– возвращает параметр типа DWORD;

- void ChangeKeyWindow::SetIntValue(uint \*newIntValue) – задает параметр типа DWORD;

- qulonglong \*ChangeKeyWindow::GetLongValue() const – возвращает значение параметра типа QWORD;

- void ChangeKeyWindow::SetLongValue(qulonglong \*newLongValue) – задает параметр типа QWORD;

- QString ChangeKeyWindow::GetKeyName() const – возвращает имя параметра.

Диаграмма классов приведена в приложении В.

# 4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

**4.1 Алгоритм по шагам метода QMap<QString, QString> Registry:: CreateGroupKeys(QTreeWidgetItem\* selectedGroup, QList <QTableWidgetItem\*>& valuesTypes)**

Шаг 1. Объявить объект класса QString pathToGroup, приравнять его к возвращаемому значению метода FindPathForGroup(), в который передается переменная selectedGroup.

Шаг 2. Объявить объект класса QSettings selectedGroupSettings, в конструктор передать pathToGroup.

Шаг 3. Объявить переменную типа auto childKeys, приравнять ее к возвращаемому значению метода selectedGroupSettings.childKeys().

Шаг 4. Объявить объект класс QMap<QString, QString> keysAndValues.

Шаг 5. Цикл foreach с элементом auto key в контейнере childKeys.

Шаг 6. Объявить переменную auto keyValue, прировнять ее к возвращаемому значению метода selectedGroupSettings.value(key).

Шаг 7. Объявить переменную типа auto typeName, прировнять ее возвращаемому значению метода QString::fromUtf8(), в который передается возвращаемое значение метода keyValue. typeName() и вызвать метод toLower().

Шаг 8. Если typeName == "qstringlist", перейти к шагу 9, иначе перейти к шагу 12.

Шаг 9. Переменная keysAndValues[key] равна возвращаемому значению метода keyValue. toStringList() и вызвать метод join(), в который передается “ “.

Шаг 10. Вызвать метод valuesTypes.append(), передать в него объект класса QTableWidgetItem, в конструктор которого передается строка "REG\_MULTI\_SZ".

Шаг 11. Перейти к Шагу 26.

Шаг 12. Если typeName == "int", перейти к шагу 13, иначе перейти к шагу 16.

Шаг 13. Переменная keysAndValues[key] равна возвращаемому значению метода QString::number(), в который передается возвращаемое значение метода keyValue.toUInt().

Шаг 14. Вызвать метод valuesTypes.append, передать в него объект класса QTableWidgetItem, в конструктор передать строку "REG\_DWORD".

Шаг 15. Перейти к шагу 26.

Шаг 16. Если typeName == "qlonglong", перейти к шагу 17, иначе перейти к шагу 20.

Шаг 17. Переменная keysAndValues[key] равна возвращаемому значению метода QString::number(), в который передается возвращаемое значение метода keyValue.toULongLong().

Шаг 18. Вызвать метод valuesTypes.append(), передать в него объект класса QTableWidgetItem, в конструктор передать строку "REG\_QWORD".

Шаг 19. Перейти к шагу 26.

Шаг 20. Если typeName == "qstring" перейти к шагу 21, иначе перейти к шагу 24.

Шаг 21. Переменная keysAndValues[key] равна возвращаемому значению метода keyValue.toString().

Шаг 22. Вызвать метод valuesTypes.append(), передать в него объект класса QTableWidgetItem, в конструктор которого передается строка "REG\_SZ".

Шаг 23. Перейти к шагу 26.

Шаг 24. Переменная keysAndValues[key] равна "Can not display this key's value!".

Шаг 25. Вызвать метод valuesTypes.append(), передать в него объект класса QTableWidgetItem, в конструктор которого передается строка "Unknown type of registry key!".

Шаг 26. Перейти к шагу 7.

Шаг 27. Конец.

**4.2 Алгоритм по шагам метода void CreateKeyWindow:: on\_okButton\_clicked()**

Шаг 1. Переменная keyName равна возвращаемому значению метода ui->inputNameLineEdit->text().

Шаг 2. Переменная keyValue равна возвращаемому значению метода ui->inputValueLineEdit->text().

Шаг 3. Если !ui->stringKey->isChecked() && !ui->intKey->isChecked() && !ui->longKey->isChecked() перейти к шагу 4, иначе перейти к шагу 6.

Шаг 4. Вызвать метод QMessageBox::critical(), передать в него следующие параметры: this, "Error!", "You should choose key type!".

Шаг 5. Конец.

Шаг 6. Если keyName.isEmpty() || keyValue.toString(). isEmpty() перейти к шагу 7, иначе перейти к шагу 9.

Шаг 7. Вызвать метод QMessageBox::critical(), передать в него следующие параметры: this, "Error!", "Error! Name or value can't be empty!".

Шаг 8. Конец.

Шаг 9. Объявить переменную bool isNumber.

Шаг 10. Если ui->intKey->isChecked() перейти к шагу 11, иначе перейти к шагу 17.

Шаг 11. Объявить перемнную типа uint number, приравнять ее к возвращаемому значению метода keyValue.toUInt(), в который передается ссылка на переменную isNumber.

Шаг 12. Если !isNumber перейти к шагу 13, иначе перейти к шагу 15.

Шаг 13. Вызвать метод QMessageBox::warning(), передать в него следующие параметры: this, "Warning!", "DWORD key can't has a string value!".

Шаг 14. Конец.

Шаг 15. Переменная keyValue равна объекту класса QVariant, в конструктор которого передается number.

Шаг 16. Перейти к шагу 30.

Шаг 17. Если ui->longKey->isChecked()перейти к шагу 18, иначе перейти к шагу 23.

Шаг 18. Объявить переменную типа qulonglong number, приравнять ее к возвращаемому значению метода keyValue.toULongLong(), в который передается ссылка на переменную isNumber.

Шаг 19. Если !isNumber перейти к шагу 20, иначе перейти к шагу 22.

Шаг 20. Вызвать метод QMessageBox::warning(), передать в него следующие параметры: this, "Warning!", "QWORD key can't be a string!".

Шаг 21. Конец.

Шаг 22. Переменная keyValue равна объекту класса QVariant, в конструктор которого передается переменная number.

Шаг 23. Вызвать метод close().

Шаг 24. Конец.

**4.3 Алгоритм по шагам метода void MainWindow::on\_registryTree\_**

**itemClicked(QTreeWidgetItem \*item, int column)**

Шаг 1. Создать переменную типа auto path, равную возвращаемому значению метода registry.FindPathForGroup(), в который передается переменная item.

Шаг 2. Вызвать метод ConfigureCurrentPathWidget(), передать в него переменную path.

Шаг 3. Вызвать метод ConfigureCreateButtons() передать в него переменную path.

Шаг 4. Вызвать метод ConfigureRemoveKeyButton().

Шаг 5. Объявить объект класса QList<QTableWidgetItem\*> valuesTypes.

Шаг 6. Объявить переменную типа auto keysAndValues, равную возвращаемому значению метода registry.CreateGroupKeys, в который передаются следующие параметры: item, valuesTypes.

Шаг 7. Создать объект класса QMap<QString, QString>::const\_iterator iterator, равный возвращаемому значению метода keysAndValues.constBegin().

Шаг 8. Вызвать метод ui->groupKeys->setRowCount(), передать в него возвращаемое значение метода keysAndValues.size().

Шаг 9. Объявить переменную типа int rowIndex, равную 0.

Шаг 10. Если iterator != keysAndValues.constEnd() перейти к шагу 11, иначе перейти к шагу 22.

Шаг 11. Объявить переменную типа auto\* keyItem, равную объекту класса QTableWidgetItem(iterator.key()).

Шаг 12. Вызвать метод keyItem->setFlags(), передать в него возвращаемое значение метода keyItem->flags() & ~Qt::ItemIsEditable.

Шаг 13. Вызвать метод ui->groupKeys->setItem(), передать в него следующие параметры: rowIndex, 0, keyItem.

Шаг 14. Вызвать метод valuesTypes[rowIndex]->setFlags(), передать в него возвращаемое значение метода valuesTypes[rowIndex]->flags() & ~Qt::ItemIsEditable.

Шаг 15. Вызвать метод ui->groupKeys->setItem(), передать в него следующие параметры: rowIndex, 1, valuesTypes[rowIndex]).

Шаг 16. Объявить переменную типа auto\* valueItem, равную объекту класса QTableWidgetItem(iterator.value()).

Шаг 17. Вызвать метод valueItem->setFlags, передать в него valueItem->flags() & ~Qt::ItemIsEditable.

Шаг 18. Вызвать метод ui->groupKeys->setItem, передать в него следующие параметры: rowIndex, 2, valueItem.

Шаг 19. Увеличить переменную iterator на 1.

Шаг 20. Увеличить переменную rowIndex на 1.

Шаг 21. Перейти к шагу 10.

Шаг 22. Конец.

**4.4 Алгоритм по шагам метода void MainWindow::on\_removeGroup Action\_triggered()**

Шаг 1. Объявить переменную типа auto currentPath, равную возвращаемому значению метода registry.FindPathForGroup(ui->registryTree->currentItem()).

Шаг 2. Объявить переменную типа auto currentPathElements, равную возвращаемому значению метода currentPath.split(), в который передается "\\".

Шаг 3. Объявить переменную типа QString baseGroupName, равную currentPathElements[0].

Шаг 4. Объявить переменную типа HKEY baseGroup, равную возвращаемому значению метода registry.GetGroupHkeyByName(), в который передается переменная baseGroupName.

Шаг 5. Вызвать метод currentPathElements.pop\_front().

Шаг 6. Переменная currentPath равна возвращаемому значению метода currentPathElements.join(), в который передается "\\".

Шаг 7. Если возвращаемое значение функции RegDeleteKeyEx(), в которую передаются следующие параметры: baseGroup, возвращаемое значение метода currentPath.utf16(), преобразованное к типу wchar\_t\*, KEY\_WOW64\_64KEY, 0, не равно ERROR\_SUCCESS перейти к шагу 8, иначе перейти к шагу 10.

Шаг 8. Вызвать метод QMessageBox::warning(), в который передаются следующие параметры: this, "Warning!", "Couldn't delete this group!".

Шаг 9. Конец.

Шаг 10. Объявить переменную типа int indexToRemove.

Шаг 11. Объявить переменную типа auto\* parentOfCurrentItem, равную возвращаемому значению метода ui->registryTree->currentItem()->parent().

Шаг 12. Цикл по переменной indexToRemove, равной нулю пока она меньше возвращаемого значение метода parentOfCurrentItem->childCount() с шагом 1.

Шаг 13. Если возвращаемое значение метода ui->registryTree->currentItem()->text(), в который передается 0, равно возвращаемому значению метода parentOfCurrentItem->child(indexToRemove)->text(), в который передается 0, перейти к шагу 14, иначе перейти к шагу 12.

Шаг 14. Прервать цикл.

Шаг 15. Вызвать метод ui->registryTree->currentItem()->parent()->takeChild(), в который передается переменная indexToRemove.

Шаг 16. Конец.

**4.5 Блок-схема метода void Registry::ParseRegistryBranch (QTreeWidgetItem \*root, QSettings \*settings, QIcon \*icon)**

Данный метод предназначен для того, чтобы отобразить на экране часть дерева реестра.

Блок-схема данного метода приведена в приложении Г на чертеже ГУИР.400201.425 Г.1.

**4.6 Блок-схема метода QTreeWidgetItem \*Registry::FindGroup ByPath(QString pathForGroup, QTreeWidgetItem\* registryTreeRoot)**

Данный метод предназначен для поиска раздела реестра в дереве по заданному пути.

Блок-схема данного метода приведена в приложении Г на чертеже ГУИР.400201.425 Г.2.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При запуске программы пользователь попадает на главное окно (рис. 5.1).



Рисунок 5.1 – Главное окно приложения.

Либо при двойном нажатии по разделу реестра, либо по нажатию на значок стрелочки отображаются подраздела текущего раздела (рис. 5.2). Если выбранный раздел содержит какие-либо параметры, то их имена, типы и значения по возможности отобразятся в левой части окна (рис 5.3).

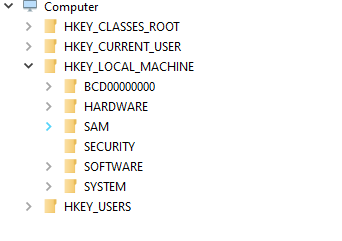


Рисунок 5.2 – Отображение подразделов выбранного раздела.

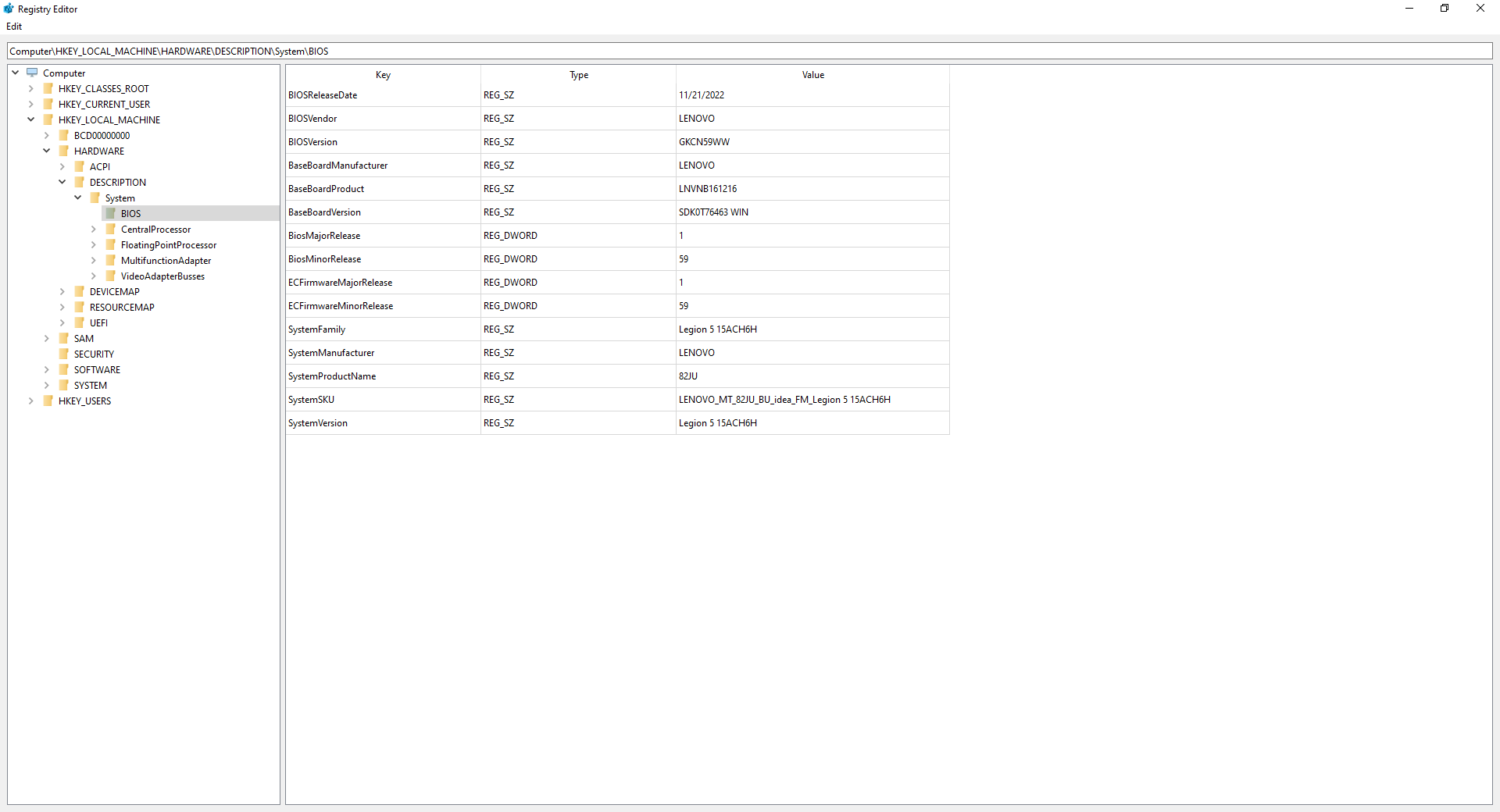


Рисунок 5.3 – Отображение параметров раздела реестра.

При выборе пункта меню “Edit” в верхней части окна появится возможность создания или удаления раздела или параметра из реестра (рис 5.4). При выборе пункта “Create” появится возможность выбрать, что именно необходимо создать (рис 5. 5).

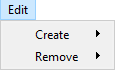


Рисунок 5.4 – Выбора пункта меню для создания или удаление параметра или раздела из реестра.

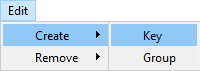


Рисунок 5.5 – Выбор объекта для создания (параметр или раздел).

При отсутствии возможности создать параметр или раздел (если, например, не выбран раздел, в котором необходимо создать параметр), то соответствующий пункт меню не будет доступен (рис 5.6).

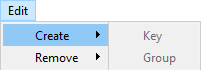


Рисунок 5.6 – Отсутствие возможности создать параметр и раздел.

Если же возможность создать параметр имеется, то при выборе пункта “Key” появится диалоговое окно для создания параметра (рис 5.7). При попытке создать параметр с некорректными данными (не выбран тип, имя или значение параметра) появится окно с предупреждением (рис 5.8 – 5.10).

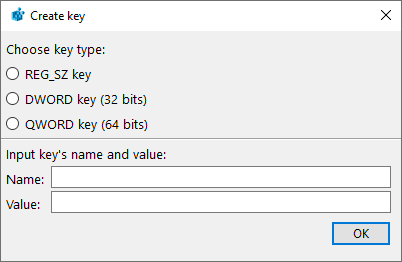


Рисунок 5.7 – Диалоговое окно для создания параметра.

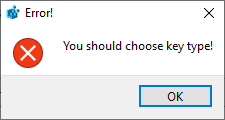


Рисунок 5.8 – Попытка создать параметр, не выбрав его тип.

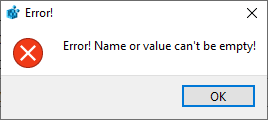


Рисунок 5.9 – Попытка создать параметр, не выбрав имя или значение.

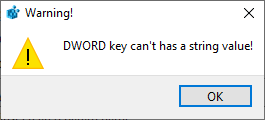


Рисунок 5.10 – Попытка создать параметр типа dword со строковым значением.

При успешном создании нового параметра его имя, тип и значение

отобразятся в правой части окна (рис 5.11).



Рисунок 5.11 – Отображение нового параметра в реестре.

При выборе подпункта меню “Group” появится диалоговое окно для создания раздела (рис 5.12).

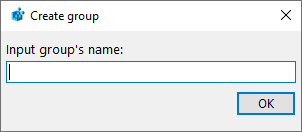


Рисунок 5.12 – Диалоговое окно для создания раздела.

При попытке создать раздел без имени появится окно об ошибке (рис. 5.13).

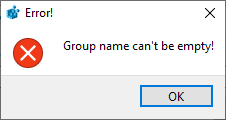


Рисунок 5.13 – Диалоговое окно с предупреждением о том, что имя раздела не может быть пустым.

При успешном создании новый раздел отобразится в дереве реестра (рис. 5.14).



Рисунок 5.14 – Отображение нового раздела в реестре.

Для удаления параметра или раздела из реестра необходимо выбрать пункт “Edit”, затем “Remove”. После этого необходимо выбрать, что именно необходимо удалить (рис. 5.15).

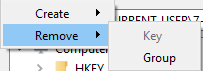


Рисунок 5.15 – Меню для удаления параметра или раздела.

После выбора пункта “Group” выбранный раздел удалится, если это возможно (рис. 5.16).

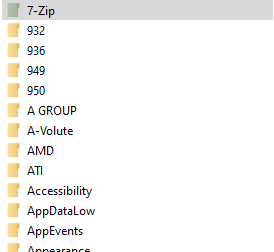


Рисунок 5.16 – дерево реестра после удаления раздела “new group” из раздела “7-Zip”.

При выборе пункта “Key” удалится выбранный параметр (рис. 5.17).

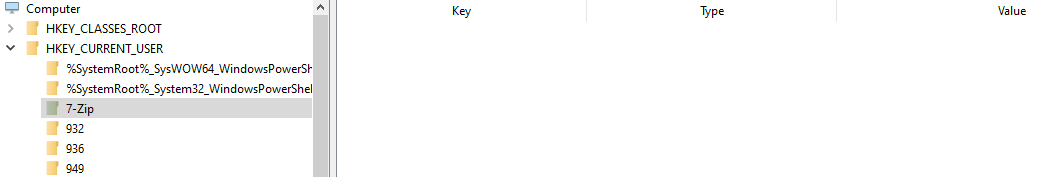


Рисунок 5.17 – Параметры раздела “7-Zip” после удаления из него параметры “just created param name”.

При двойном нажатии по параметру появится диалоговое окно для изменения имени или значения параметра (рис. 5.18).

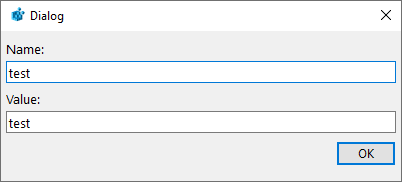


Рисунок 5.18 – Диалоговое окно для редактирования параметра.

После нажатия кнопки “OK” параметр изменит свое имя или значение на новое (рис 5.19).



Рисунок 5.19 – Изменение значения параметра с “test” на “new\_test”.

# ЗАКЛЮЧЕНИЕ

В ходе данного курсового проекта было разработано приложение, позволяющие просматривать дерево реестра, а также создавать и редактировать разделы и параметры. Для написания программы были изучены фреймворк Qt, с помощью которого создавался пользовательский интерфейс приложения и часть программной логики, программный интерфейс WinAPI, также позволяющий работать с реестром, и были углублены знания языка программирования C++ и навыки в области объектно-ориентированного программирования.

Несмотря на то, что программа имеет довольно удобный пользовательский интерфейс, позволяет просматривать разделы в реестре, параметры в разделах, а также создавать и удалять разделы и параметры, ее можно улучшить следующим образом:

- увеличить количество типов параметров, доступных для создания

(например, REG\_MULTI\_SZ);

- увеличить количество типов параметров, доступных для просмотра

(например, REG\_FULL\_RESOURCE\_DESCRIPTOR);

- увеличить количество типов параметров, доступных для

редактирования (например, REG\_MULTI\_SZ).

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Официальный сайт документации Qt [Электронный ресурс]. —

Режим доступа - <https://doc.qt.io/.> – [Дата доступа] – 04.05.2023.

[2] Сайт Wikipedia [Электронный ресурс]. —

Режим доступа - [https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B5 %D1%81%D1%82%D1%80\_Windows](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B5%20%D1%81%D1%82%D1%80_Windows). – [Дата доступа] – 04.05.2023.

[3] Официальный сайт документации WinAPI [Электронный ресурс]. — Режим доступа - <https://learn.microsoft.com/ru-ru/windows/win32/api/winreg/nf-winreg-regcreatekeya>. – [Дата доступа] – 15.04.2023.

# ПРИЛОЖЕНИЕ А

(*обязательное*)

Листинг программы с комментариями

// ChangeKeyWindow.h

#ifndef CHANGEKEYWINDOW\_H

#define CHANGEKEYWINDOW\_H

#include <QDialog>

namespace Ui {

class ChangeKeyWindow;

}

class ChangeKeyWindow : public QDialog

{

Q\_OBJECT

public:

ChangeKeyWindow(QString, QWidget \*parent = nullptr);

~ChangeKeyWindow();

void ConfigureLineEdits();

QString \*GetStringValue() const;

void SetStringValue(QString \*newStringValue);

uint \*GetIntValue() const;

void SetIntValue(uint \*newIntValue);

qulonglong \*GetLongValue() const;

void SetLongValue(qulonglong \*newLongValue);

QString GetKeyName() const;

private slots:

void on\_okButton\_clicked();

private:

void ConfigureWindow();

QString\* stringValue;

uint\* intValue;

qulonglong\* longValue;

QString keyName;

Ui::ChangeKeyWindow \*ui;

};

#endif // CHANGEKEYWINDOW\_H

// ChangeKeyWindow.cpp

#include "../include/ChangeKeyWindow.h"

#include "ui\_ChangeKeyWindow.h"

#include <QMessageBox>

ChangeKeyWindow::ChangeKeyWindow(QString keyName, QWidget \*parent): QDialog(parent), stringValue(nullptr), intValue(nullptr), longValue(nullptr), keyName(keyName),

ui(new Ui::ChangeKeyWindow)

{

ui->setupUi(this);

ConfigureWindow();

}

ChangeKeyWindow::~ChangeKeyWindow()

{

delete ui;

}

void ChangeKeyWindow::ConfigureWindow()

{

setFixedSize(400, 150);

setWindowModality(Qt::WindowModal);

}

QString ChangeKeyWindow::GetKeyName() const

{

return keyName;

}

void ChangeKeyWindow::ConfigureLineEdits()

{

ui->nameLineEdit->setText(keyName); // set selected key name

if (stringValue) // set selected key value according to it type

{

ui->valueLineEdit->setText(\*stringValue);

}

else if (intValue)

{

ui->valueLineEdit->setText(QString::number(\*intValue));

}

else

{

ui->valueLineEdit->setText(QString::number(\*longValue));

}

}

qulonglong \*ChangeKeyWindow::GetLongValue() const

{

return longValue;

}

void ChangeKeyWindow::SetLongValue(qulonglong \*newLongValue)

{

longValue = newLongValue;

}

uint \*ChangeKeyWindow::GetIntValue() const

{

return intValue;

}

void ChangeKeyWindow::SetIntValue(uint \*newIntValue)

{

intValue = newIntValue;

}

QString \*ChangeKeyWindow::GetStringValue() const

{

return stringValue;

}

void ChangeKeyWindow::SetStringValue(QString \*newStringValue)

{

stringValue = newStringValue;

}

void ChangeKeyWindow::on\_okButton\_clicked()

{

auto newValue = ui->valueLineEdit->text(); // new value for selected key

auto newName = ui->nameLineEdit->text(); // new name for selected key

if (newValue.isEmpty() || newName.isEmpty())

{

QMessageBox::critical(this, "Error!", "Error! New name or value can't be empty!");

return;

}

keyName = newName; // set new name

// set new value for the key according to it type

if (stringValue)

{

\*stringValue = newValue;

}

else if (intValue)

{

bool isOk;

\*intValue = newValue.toUInt(&isOk);

if (!isOk)

{

QMessageBox::warning(this, "Warning!", "DWORD key can't has a string value!"); // if string was entered for dword key

return;

}

}

else

{

bool isOk;

\*longValue = newValue.toULongLong(&isOk);

if (!isOk)

{

QMessageBox::warning(this, "Warning!", "QWORD key can't has a string value!"); // if string was entered for qword key

return;

}

}

close();

}

// CreateGroupWindow.h

#ifndef CREATEGROUPWINDOW\_H

#define CREATEGROUPWINDOW\_H

#include <QDialog>

namespace Ui {

class CreateGroupWindow;

}

class CreateGroupWindow : public QDialog

{

Q\_OBJECT

public:

CreateGroupWindow(QString, QWidget \*parent = nullptr);

~CreateGroupWindow();

QString GetGroupName() const;

private slots:

void on\_okButton\_clicked();

private:

void ConfigureWindow();

QString currentPath;

QString groupName;

Ui::CreateGroupWindow \*ui;

};

#endif

// CreateGroupWindow.cpp

#include "../include/CreateGroupWindow.h"

#include "ui\_CreateGroupWindow.h"

#include "../include/Registry.h"

#include <QMessageBox>

CreateGroupWindow::CreateGroupWindow(QString currentPath, QWidget \*parent): QDialog(parent), currentPath(currentPath), ui(new Ui::CreateGroupWindow)

{

ui->setupUi(this);

ConfigureWindow();

}

CreateGroupWindow::~CreateGroupWindow()

{

delete ui;

}

void CreateGroupWindow::ConfigureWindow()

{

setWindowModality(Qt::ApplicationModal);

setWindowTitle("Create group");

setFixedSize(300, 100);

}

QString CreateGroupWindow::GetGroupName() const

{

return groupName;

}

void CreateGroupWindow::on\_okButton\_clicked()

{

groupName = ui->inputNameLineEdit->text();

if (groupName.isEmpty())

{

QMessageBox::critical(this, "Error!", "Group name can't be empty!");

}

auto pathElements = currentPath.split("\\"); // get all elements of the current path

pathElements.push\_back(groupName); // add new group name for the current path

HKEY baseGroup = Registry().GetGroupHkeyByName(pathElements[0]); // curent group descriptor

pathElements.pop\_front(); // remove "Computer" from the current path

HKEY hKey;

auto pathToNewGroup = pathElements.join("\\");

if (RegCreateKeyEx(baseGroup, (wchar\_t\*)pathToNewGroup.utf16(), 0, NULL, REG\_OPTION\_NON\_VOLATILE, KEY\_WRITE, NULL, &hKey, NULL) != ERROR\_SUCCESS) // attempt to create new group

{

QMessageBox::warning(this, "Warning!", "Couldn't create new group!");

return;

}

close();

}

// CreateKeyWindow.h

#ifndef CREATEKEYWINDOW\_H

#define CREATEKEYWINDOW\_H

#include <QDialog>

#include <QVariant>

namespace Ui {

class CreateKeyWindow;

}

class CreateKeyWindow : public QDialog

{

Q\_OBJECT

public:

CreateKeyWindow(QWidget \*parent = nullptr);

~CreateKeyWindow();

QString GetKeyName() const;

QVariant GetKeyValue() const;

private slots:

void on\_okButton\_clicked();

private:

void ConfigureWindow();

QString keyName;

QVariant keyValue;

Ui::CreateKeyWindow \*ui;

};

#endif

// CreateKeyWindow.cpp

#include "../include/CreateKeyWindow.h"

#include "ui\_CreateKeyWindow.h"

#include <QMessageBox>

#include <QSettings>

CreateKeyWindow::CreateKeyWindow(QWidget \*parent): QDialog(parent), ui(new Ui::CreateKeyWindow)

{

ui->setupUi(this);

ConfigureWindow();

}

CreateKeyWindow::~CreateKeyWindow()

{

delete ui;

}

QString CreateKeyWindow::GetKeyName() const

{

return keyName;

}

QVariant CreateKeyWindow::GetKeyValue() const

{

return keyValue;

}

void CreateKeyWindow::ConfigureWindow()

{

setWindowModality(Qt::ApplicationModal);

setFixedSize(400, 230);

setWindowTitle("Create key");

}

void CreateKeyWindow::on\_okButton\_clicked()

{

keyName = ui->inputNameLineEdit->text(); // name for the new key

keyValue = ui->inputValueLineEdit->text(); // value for the new key

if (!ui->stringKey->isChecked() && !ui->intKey->isChecked() && !ui->longKey->isChecked()) // if type for the new key wasn't selected

{

QMessageBox::critical(this, "Error!", "You should choose key type!");

return;

}

else if (keyName.isEmpty() || keyValue.toString().isEmpty()) // if name of value wasn't entered

{

QMessageBox::critical(this, "Error!", "Error! Name or value can't be empty!");

return;

}

bool isNumber;

if (ui->intKey->isChecked())

{

uint number = keyValue.toUInt(&isNumber);

if (!isNumber) // if string was entered for the dword key

{

QMessageBox::warning(this, "Warning!", "DWORD key can't has a string value!");

return;

}

keyValue = QVariant(number);

}

else if (ui->longKey->isChecked())

{

qulonglong number = keyValue.toULongLong(&isNumber);

if (!isNumber) // if string was entered for the qword key

{

QMessageBox::warning(this, "Warning!", "QWORD key can't be a string!");

return;

}

keyValue = QVariant(number);

}

close();

}

// main.cpp

#include "../include/MainWindow.h"

#include <QApplication>

#include <QSettings>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

// MainWindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTreeWidget>

#include "Registry.h"

namespace Ui { class MainWindow; }

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_registryTree\_itemExpanded(QTreeWidgetItem \*item);

void on\_registryTree\_itemClicked(QTreeWidgetItem \*item, int column);

void on\_currentPath\_returnPressed();

void on\_createKeyAction\_triggered();

void on\_groupKeys\_itemClicked(QTableWidgetItem \*item);

void on\_removeKeyAction\_triggered();

void on\_createGroupAction\_triggered();

void on\_removeGroupAction\_triggered();

void on\_groupKeys\_cellDoubleClicked(int row, int column);

private:

void ConfigureMainWIndow();

void ConfigureRegistryTreeWidget();

void ConfigureGroupKeysWidget();

void ConfigureCurrentPathWidget(QString path = "");

void ConfigureCreateButtons(QString path = "");

void ConfigureRemoveKeyButton(QTableWidgetItem\* currentItem = nullptr);

QTreeWidgetItem\* CreateRegistryBranch(QString, QTreeWidgetItem\*);

Ui::MainWindow \*ui;

Registry registry;

};

#endif

// MainWindow.cpp

#include "../include/MainWindow.h"

#include "../include/Registry.h"

#include "../include/CreateGroupWindow.h"

#include "../include/CreateKeyWindow.h"

#include "../include/ChangeKeyWindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

MainWindow::MainWindow(QWidget \*parent): QMainWindow(parent), ui(new Ui::MainWindow)

{

ui->setupUi(this);

ConfigureMainWIndow();

ConfigureRegistryTreeWidget();

ConfigureGroupKeysWidget();

ConfigureCurrentPathWidget();

ConfigureCreateButtons();

ConfigureRemoveKeyButton();

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::ConfigureMainWIndow()

{

setWindowIcon(QIcon(":/img/window\_icon.png"));

setWindowTitle("Registry Editor");

}

void MainWindow::ConfigureRegistryTreeWidget()

{

showMaximized();

QTreeWidgetItem\* root = new QTreeWidgetItem(QStringList() << "Computer");

root->setIcon(0, QIcon(":/img/computer\_icon.png"));

ui->registryTree->addTopLevelItem(root); // add top level item ("Computer") for the registry tree

QStringList baseGroups =

{

"HKEY\_CLASSES\_ROOT",

"HKEY\_CURRENT\_USER",

"HKEY\_LOCAL\_MACHINE",

"HKEY\_USERS"

};

foreach (auto groupName, baseGroups) // create subgroups for every base group

{

CreateRegistryBranch(groupName, ui->registryTree->topLevelItem(0));

}

}

void MainWindow::ConfigureGroupKeysWidget()

{ // creation table for key name, type and value

ui->groupKeys->setColumnCount(3);

ui->groupKeys->setHorizontalHeaderLabels(QStringList() << "Key" << "Type" << "Value");

ui->groupKeys->verticalHeader()->setVisible(false);

ui->groupKeys->horizontalHeader()->setSectionsClickable(false);

ui->groupKeys->horizontalHeader()->resizeSection(0, 250);

ui->groupKeys->horizontalHeader()->resizeSection(1, 250);

ui->groupKeys->horizontalHeader()->resizeSection(2, 350);

}

void MainWindow::ConfigureCurrentPathWidget(QString path)

{ // line edit which contains path for selected group

ui->currentPath->setText("Computer\\" + path);

ui->currentPath->setReadOnly(true);

}

void MainWindow::ConfigureCreateButtons(QString path)

{

if (path.isEmpty()) // create buttons can be shown only when current path isn't empty

{

ui->createGroupAction->setEnabled(false);

ui->createKeyAction->setEnabled(false);

}

else

{

ui->createGroupAction->setEnabled(true);

ui->createKeyAction->setEnabled(true);

}

}

void MainWindow::ConfigureRemoveKeyButton(QTableWidgetItem \*currentItem)

{

if (!currentItem) // remove key button can be shown only if there is selected table item

{

ui->removeKeyAction->setEnabled(false);

}

else

{

ui->removeKeyAction->setEnabled(true);

}

}

QTreeWidgetItem\* MainWindow::CreateRegistryBranch(QString groupName, QTreeWidgetItem\* parentItem)

{

QTreeWidgetItem\* item = new QTreeWidgetItem(QStringList() << groupName);

QIcon icon(":/img/group\_icon.ico");

item->setIcon(0, icon);

parentItem->addChild(item);

QSettings settings(groupName, QSettings::NativeFormat);

registry.ParseRegistryBranch(item, &settings, &icon);

return item;

}

void MainWindow::on\_registryTree\_itemExpanded(QTreeWidgetItem \*group)

{

Q\_UNUSED(group);

ui->registryTree->resizeColumnToContents(0);

QSettings settings(registry.FindPathForGroup(group), QSettings::NativeFormat);

QIcon icon(":/img/group\_icon.ico");

registry.ParseRegistryBranch(group, &settings, &icon);

}

void MainWindow::on\_registryTree\_itemClicked(QTreeWidgetItem \*item, int column)

{

Q\_UNUSED(column);

auto path = registry.FindPathForGroup(item);

ConfigureCurrentPathWidget(path); // set new path in the line edit

ConfigureCreateButtons(path); // enable create buttons enable

ConfigureRemoveKeyButton(); // disable remove key button

QList<QTableWidgetItem\*> valuesTypes;

auto keysAndValues = registry.CreateGroupKeys(item, valuesTypes); // get map which contains key names and values; get list of table items with key types

QMap<QString, QString>::const\_iterator iterator = keysAndValues.constBegin();

ui->groupKeys->setRowCount(keysAndValues.size()); // set row count for the table

int rowIndex = 0;

while (iterator != keysAndValues.constEnd())

{

// create table item for key name; make it immutable

auto\* keyItem = new QTableWidgetItem(iterator.key());

keyItem->setFlags(keyItem->flags() & ~Qt::ItemIsEditable);

ui->groupKeys->setItem(rowIndex, 0, keyItem);

// create table item for key type; make it immutable

valuesTypes[rowIndex]->setFlags(valuesTypes[rowIndex]->flags() & ~Qt::ItemIsEditable);

ui->groupKeys->setItem(rowIndex, 1, valuesTypes[rowIndex]);

// create table item for key value; make it immutable

auto\* valueItem = new QTableWidgetItem(iterator.value());

valueItem->setFlags(valueItem->flags() & ~Qt::ItemIsEditable);

ui->groupKeys->setItem(rowIndex, 2, valueItem);

// increase row number and iterator

iterator++;

rowIndex++;

}

}

void MainWindow::on\_currentPath\_returnPressed()

{

auto group = registry.FindGroupByPath(ui->currentPath->text(), ui->registryTree->topLevelItem(0)); // get registry tree item with current path

ui->registryTree->setCurrentItem(group); // make it current item

emit ui->registryTree->itemClicked(group, 0); // open it

}

void MainWindow::on\_createKeyAction\_triggered()

{

ConfigureRemoveKeyButton(); // disable remove key button

CreateKeyWindow createKeyWindow(this);

createKeyWindow.show();

createKeyWindow.exec();

auto currentItem = ui->registryTree->currentItem();

auto path = registry.FindPathForGroup(currentItem);

QSettings settings(path, QSettings::NativeFormat);

settings.setValue(createKeyWindow.GetKeyName(), createKeyWindow.GetKeyValue()); // create new key

on\_registryTree\_itemClicked(currentItem, 0); // display new key

}

void MainWindow::on\_groupKeys\_itemClicked(QTableWidgetItem \*item)

{

ConfigureRemoveKeyButton(item); // enable remove key button

}

void MainWindow::on\_removeKeyAction\_triggered()

{

auto keyName = ui->groupKeys->selectedItems()[0]->text(); // get name of selected key

auto currentGroup = ui->registryTree->currentItem();

auto groupName = registry.FindPathForGroup(currentGroup);

QSettings settings(groupName, QSettings::NativeFormat);

settings.remove(keyName); // remove selected key

on\_registryTree\_itemClicked(currentGroup, 0);

}

void MainWindow::on\_createGroupAction\_triggered()

{

CreateGroupWindow createGroupWindow(registry.FindPathForGroup(ui->registryTree->currentItem()), this);

createGroupWindow.show();

createGroupWindow.exec();

auto createdGroupName = createGroupWindow.GetGroupName(); // get name for the new group

if (createdGroupName.isEmpty())

{

return;

}

CreateRegistryBranch(createdGroupName, ui->registryTree->currentItem()); // create subbranch for the new item including it

ConfigureCurrentPathWidget(registry.FindPathForGroup(ui->registryTree->currentItem()) + "\\" + createdGroupName); // set new path

on\_currentPath\_returnPressed(); // open new group

}

void MainWindow::on\_removeGroupAction\_triggered()

{

auto currentPath = registry.FindPathForGroup(ui->registryTree->currentItem());

auto currentPathElements = currentPath.split("\\");

QString baseGroupName = currentPathElements[0]; // get base group name

HKEY baseGroup = registry.GetGroupHkeyByName(baseGroupName); // get base group descriptor

currentPathElements.pop\_front(); // remobe base group name

currentPath = currentPathElements.join("\\"); // create path for the selected group

if (RegDeleteKeyEx(baseGroup, (wchar\_t\*)currentPath.utf16(), KEY\_WOW64\_64KEY, 0) != ERROR\_SUCCESS) // if group wasn't removed

{

QMessageBox::warning(this, "Warning!", "Couldn't delete this group!");

return;

}

int indexToRemove;

auto\* parentOfCurrentItem = ui->registryTree->currentItem()->parent();

for (indexToRemove = 0; indexToRemove < parentOfCurrentItem->childCount(); indexToRemove++)

{

if (ui->registryTree->currentItem()->text(0) == parentOfCurrentItem->child(indexToRemove)->text(0)) // get index of the removed group

{

break;

}

}

ui->registryTree->currentItem()->parent()->takeChild(indexToRemove); // remove group from the tree

}

void MainWindow::on\_groupKeys\_cellDoubleClicked(int row, int column)

{

Q\_UNUSED(row);

Q\_UNUSED(column);

auto currentKeyName = ui->groupKeys->selectedItems()[0]->text(); // get name of selected key

ChangeKeyWindow window(currentKeyName, this);

QString valueType = ui->groupKeys->selectedItems()[1]->text(); // get type of the selected type

// get value of the key and set it in the window

if (valueType == "REG\_SZ")

{

auto keyValue = ui->groupKeys->selectedItems()[2]->text();

window.SetStringValue(&keyValue);

}

else if (valueType == "REG\_DWORD")

{

auto keyValue = ui->groupKeys->selectedItems()[2]->text().toUInt();

window.SetIntValue(&keyValue);

}

else if (valueType == "REG\_QWORD")

{

auto keyValue = ui->groupKeys->selectedItems()[2]->text().toULongLong();

window.SetLongValue(&keyValue);

}

else

{

QMessageBox::warning(this, "Warning!", "Unable to change this key!");

return;

}

window.ConfigureLineEdits();

window.show();

window.exec();

// get key name and path for this key

auto currentItem = ui->registryTree->currentItem();

auto path = registry.FindPathForGroup(currentItem);

QSettings settings(path, QSettings::NativeFormat);

auto keyName = window.GetKeyName();

if (currentKeyName != keyName)

{

on\_removeKeyAction\_triggered();

}

// set new value in registry

if (valueType == "REG\_SZ")

{

settings.setValue(keyName, \*window.GetStringValue());

}

else if (valueType == "REG\_DWORD")

{

settings.setValue(keyName, \*window.GetIntValue());

}

else if (valueType == "REG\_QWORD")

{

settings.setValue(keyName, \*window.GetLongValue());

}

on\_registryTree\_itemClicked(currentItem, 0);

}

// Registry.h

#ifndef REGISTRY\_H

#define REGISTRY\_H

#include <QTreeWidget>

#include <QSettings>

#include <QIcon>

#include <QTableWidgetItem>

#include <QMap>

#include <windows.h>

#include <winreg.h>

class Registry

{

public:

void ParseRegistryBranch(QTreeWidgetItem\*, QSettings\*, QIcon\*);

QString FindPathForGroup(QTreeWidgetItem\*);

QTreeWidgetItem\* FindGroupByPath(QString, QTreeWidgetItem\*);

QMap<QString, QString> CreateGroupKeys(QTreeWidgetItem\*, QList<QTableWidgetItem\*>&);

HKEY GetGroupHkeyByName(QString);

private:

const QString PATH\_SEPARATOR = "\\";

};

#endif // REGISTRY\_H

// Registry.cpp

#include "../include/Registry.h"

void Registry::ParseRegistryBranch(QTreeWidgetItem \*root, QSettings \*settings, QIcon \*icon)

{ // get all group on 2 levels

static int depth = 0;

static const int MAX\_DEPTH = 2;

depth++;

if (root->childCount() > 0)

{

for (int i = 0; i < root->childCount(); i++)

{

auto child = root->child(i);

if (child->childCount() > 0)

{

break;

}

settings->beginGroup(child->text(0));

ParseRegistryBranch(child, settings, icon);

settings->endGroup();

}

}

else if (depth <= MAX\_DEPTH)

{

foreach (const auto& group, settings->childGroups())

{

auto\* item = new QTreeWidgetItem(QStringList() << group);

item->setIcon(0, \*icon);

root->addChild(item);

settings->beginGroup(group);

ParseRegistryBranch(item, settings, icon);

settings->endGroup();

}

}

depth--;

}

QString Registry::FindPathForGroup(QTreeWidgetItem \*group)

{

QStringList reversedPath;

for (; group->text(0) != "Computer"; group = group->parent())

{

reversedPath.append(group->text(0));

}

std::reverse(reversedPath.begin(), reversedPath.end());

return reversedPath.join(PATH\_SEPARATOR);

}

QTreeWidgetItem \*Registry::FindGroupByPath(QString pathForGroup, QTreeWidgetItem\* registryTreeRoot)

{

auto pathElements = pathForGroup.split(PATH\_SEPARATOR);

if (pathElements.last() == "")

{

pathElements.pop\_back();

}

if (registryTreeRoot->text(0) != pathElements[0])

{

return nullptr;

}

pathElements.pop\_front();

foreach (auto pathElement, pathElements)

{

for (int i = 0; i < registryTreeRoot->childCount(); i++)

{

if (pathElement == registryTreeRoot->child(i)->text(0))

{

registryTreeRoot = registryTreeRoot->child(i);

pathElements.pop\_front();

break;

}

}

}

if (pathElements.size() > 0)

{

return nullptr;

}

return registryTreeRoot;

}

QMap<QString, QString> Registry::CreateGroupKeys(QTreeWidgetItem \*selectedGroup, QList<QTableWidgetItem\*>& valuesTypes)

{

QString pathToGroup = FindPathForGroup(selectedGroup); // get path for selected group

QSettings selectedGroupSettings(pathToGroup, QSettings::NativeFormat);

auto childKeys = selectedGroupSettings.childKeys(); // get child keys

QMap<QString, QString> keysAndValues;

foreach (auto key, childKeys)

{

auto keyValue = selectedGroupSettings.value(key);

auto typeName = QString::fromUtf8(keyValue.typeName()).toLower();

if (typeName == "qstringlist")

{ // get string list key

keysAndValues[key] = keyValue.toStringList().join(" ");

valuesTypes.append(new QTableWidgetItem("REG\_MULTI\_SZ"));

}

else if (typeName == "int")

{ // get dword key

keysAndValues[key] = QString::number(keyValue.toUInt());

valuesTypes.append(new QTableWidgetItem("REG\_DWORD"));

}

else if (typeName == "qlonglong")

{ // get qword key

keysAndValues[key] = QString::number(keyValue.toULongLong());

valuesTypes.append(new QTableWidgetItem("REG\_QWORD"));

}

else if (typeName == "qstring")

{ // get string value

keysAndValues[key] = keyValue.toString();

valuesTypes.append(new QTableWidgetItem("REG\_SZ"));

}

else

{ // unable to read this key

keysAndValues[key] = "Can not display this key's value!";

valuesTypes.append(new QTableWidgetItem("Unknown type of registry key!"));

}

}

return keysAndValues;

}

HKEY Registry::GetGroupHkeyByName(QString baseGroupName)

{ // get base group desciptor according to it name

if (baseGroupName == "HKEY\_CLASSES\_ROOT")

{

return HKEY\_CLASSES\_ROOT;

}

else if (baseGroupName == "HKEY\_CURRENT\_USER")

{

return HKEY\_CURRENT\_USER;

}

else if (baseGroupName == "HKEY\_LOCAL\_MACHINE")

{

return HKEY\_LOCAL\_MACHINE;

}

else

{

return HKEY\_USERS;

}

}

ПРИЛОЖЕНИЕ Б  
(о*бязательное*)  
  
Диаграмма классов

ПРИЛОЖЕНИЕ В  
(о*бязательное*)  
  
Блок-схема алгоритмов

# ПРИЛОЖЕНИЕ Г

(о*бязательное*)

Ведомость документов