**ПРОЕКТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ**

**3. 1. Разработка структуры приложения**

Для представления принципа работы программы была составлена UML-диаграмма вариантов использования (use case diagram). Данная диаграмма отражает функциональность и поведение системы и является составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

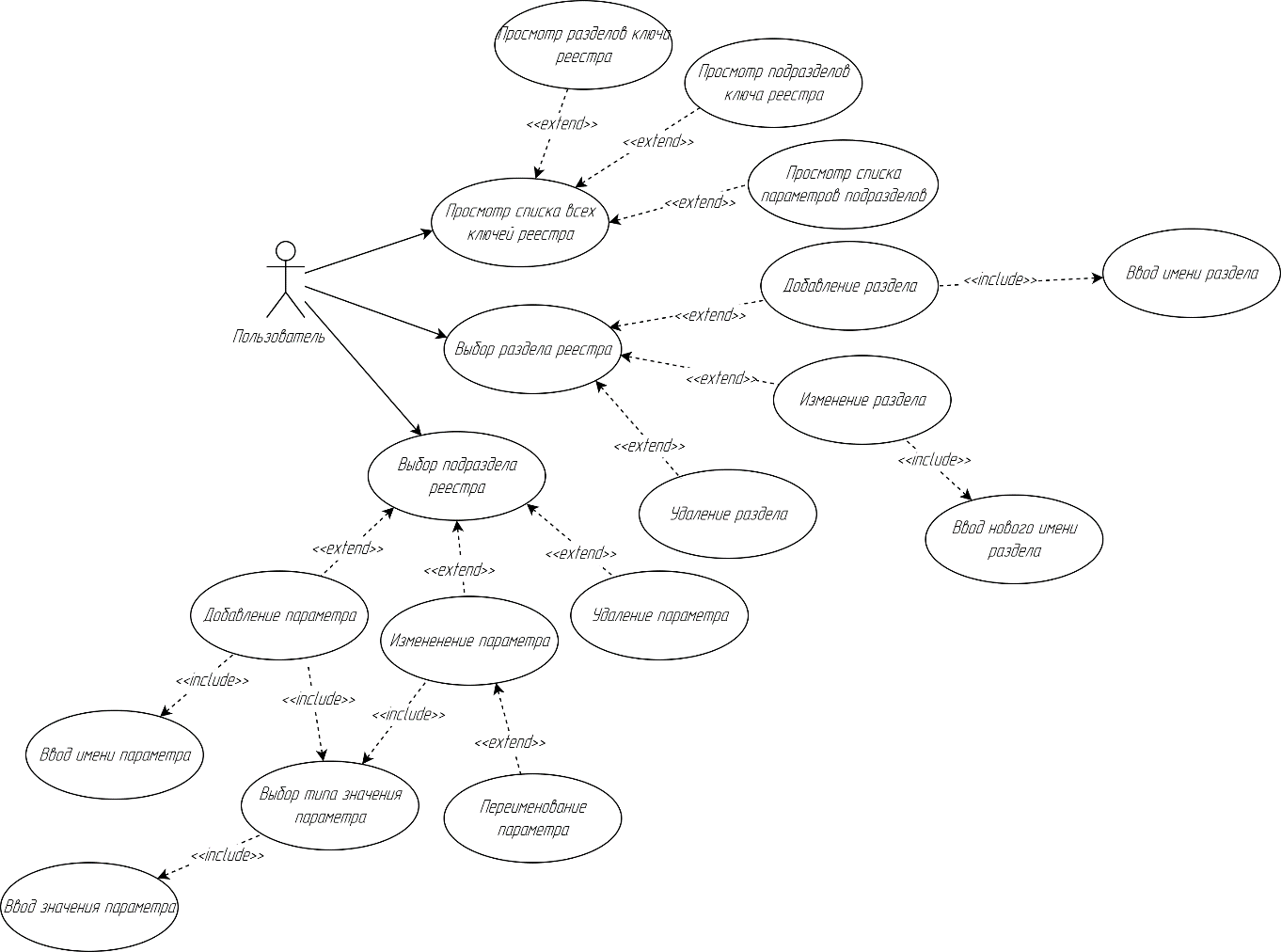


Рисунок 3.1 – Диаграмма вариантов использования программы «Редактор реестра»

Выбор раздела редактора реестра осуществляется из пяти имеющихся основных разделов:

HKEY\_CURRENT\_USER – раздел отвечает за данные пользователя, вошедшего в систему в настоящий момент. В нем хранятся папки пользователя, фон и цвет экрана рабочего стола, значки приложений, а также настройки панели управления и др. Эти данные являются профилем пользователя.

HKEY\_USERS – раздел содержит информацию обо всех профилях пользователей на компьютере.

HKEY\_LOCAL\_MACHINE – раздел хранит конфигурацию аппаратного и программного обеспечения. В нем содержатся данные настроек, относящихся к конкретному компьютеру.

HKEY\_CLASSES\_ROOT – раздел содержит сведения о расширениях всех зарегистрированных в системе типов файлов и ассоциациях (отвечает за запуск необходимой программы при открытии файла с помощью «Проводника Windows»).

HKEY\_CURRENT\_CONFIG – раздел содержит аппаратные параметры, необходимые для загрузки системы. В нем хранятся сведения о профиле оборудования.

После выбора основного раздела редактора реестра осуществляется выбор следующего раздела, в который представляется возможным добавить новый раздел, изменить его название, а также впоследствии удалить его. В добавленный раздел также можно добавить подраздел.

Добавление, изменение и удаление раздела или подраздела выполняется посредством нажатия соответствующих кнопок графического интерфейса.

В выбранном разделе или подразделе осуществляется добавление параметра. Для того, чтобы добавить новый параметр, необходимо выбрать тип значения параметра, ввести его название и затем значение.

Графический интерфейс предлагает выбор из шести типов значений параметра: двоичные данные, DWORD, QWORD, строка, многострочная строка и расширяемая строка.

Двоичные типы данных отображаются в реестре в виде последовательности байтов, записанных в шестнадцатеричной системе исчисления с использованием цифр и букв от «A» до «F».

Данные типа DWORD – это тип двоичных данных, содержащих 4 байта. Этот тип отображается в редакторе реестра в шестнадцатеричном и десятичном форматах. Предназначен для работы с 32-битными системами, но также используется в 64-битных системах.

Данные типа QWORD – это данные, представленные в виде 64-разрядного целого. Этот параметр несет любое целое число в диапазоне от 64 бит. Предназначен для работы с 64-битными системами.

Строка – тип данных, представляющий собой текст постоянной длины. Данные могут содержать буквы, цифры и символы.

Многострочная строка – тип данных, который может содержать список значений вместо одной строки. Не все многострочные значения в реестре будут иметь более одной записи. Некоторые функционируют так же, как и одиночные строковые значения, но имеют дополнительное место для большего количества записей, если им это потребуется.

Расширяемая строка – тип данных, представляющий собой текст переменной длины. Значение этого типа мо­жет включать переменные среды окружения, а программа, использу­ющая эти значения, берет эти переменные из реестра и преобразует перед тем, как использовать их.

После того, как указан тип значения параметра, задано его название и записано значение, полученный параметр будет добавлен в раздел или подраздел редактора реестра – в зависимости от того, что до этого выбрал пользователь.

В графическом интерфейсе будут отображаться имя, тип и значение параметра. При необходимости возможно изменение названия или значения параметра, а также его удаления, для чего следует нажать соответствующие кнопки интерфейса приложения.

**3. 2. Используемые функции Win32API**

Используемые библиотеки Win32API, функции и их аргументы:

**1. Библиотека winuser.h**

* DlgProc – функция, которая обрабатывает сообщения, отправляемые модальному диалоговому окну. Представляет собой функцию-обработчик сообщений главного окна.

INT\_PTR CALLBACK DialogProc(

HWND hwndDlg,

UINT uMsg,

WPARAM wParam,

LPARAM lParam

);

hwndDlg – дескриптор диалогового окна.

uMsg – устанавливает сообщение.

wParam – определяет дополнительную информацию, относящуюся к сообщению.

lParam – определяет дополнительную информацию, относящуюся к сообщению.

* EndDialog – функция, которая уничтожает модальное диалоговое окно, заставляя систему закончить любую обработку информации диалогового окна.

BOOL EndDialog(

HWND hDlg,

INT\_PTR nResult

);

hDlg – дескриптор уничтожаемого диалогового окна.

nResult – значение, которое будет возвращено приложению из функции, создавшей диалоговое окно.

* DialogBox – функция, которая создает модальное диалоговое окно из ресурса шаблона блока диалога. DialogBox не возвращает управления до тех пор, пока заданная функция обратного вызова не завершит работу модального диалогового окна путем вызова функции EndDialog.

INT\_PTR DialogBox(

hInstance,

lpTemplate,

hWndParent,

lpDialogFunc

);

hInstance – дескриптор модуля, исполняемый файл которого содержит шаблон диалогового окна.

lpTemplate – определяет шаблон диалогового окна.

hWndParent – дескриптор окна, которое владеет диалоговым окном.

lpDialogFunc – указатель на процедуру диалогового окна.

* MessageBox – функция, которая создает, показывает на экране и использует окно сообщения. Окно сообщения содержит определяемое программой сообщение и заголовок, а также любую комбинацию предопределенных значков и командных кнопок.

int MessageBox(

HWND hWnd,

LPCTSTR lpText,

LPCTSTR lpCaption,

UINT uType

);

hWnd – дескриптор окна владельца создаваемого окна сообщения.

lpText – сообщение, которое будет отображаться.

lpCaption – заголовок диалогового окна.

uType – содержание и поведение диалогового окна.

* SendMessage – функция, которая отправляет указанное сообщение в окно или окна. Она вызывает оконную процедуру для указанного окна и не возвращается, пока оконная процедура не обработает сообщение.

LRESULT SendMessage(

HWND hWnd,

UINT Msg,

WPARAM wParam,

LPARAM lParam

);

hWnd – дескриптор окна, оконная процедура которого получит сообщение.

Msg – сообщение для отправки.

wParam – дополнительная информация, относящаяся к сообщению.

lParam – дополнительная информация, относящаяся к сообщению.

* GetDlgItem – функция, которая извлекает дескриптор элемента управления в указанном диалоговом окне.

HWND GetDlgItem(

HWND hDlg,

int nIDDlgItem

);

hDlg – дескриптор диалогового окна, содержащего элемент управления.

nIDDlgItem – идентификатор извлекаемого элемента управления.

**2. Библиотека winbase.h**

* WinMain – функция создаваемого приложения. Определяет класс окна, создает главное окно приложения и «присоединяет к нему» функцию обработки сообщений Windows.

int \_\_clrcall WinMain(

HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nShowCmd

);

hInstance – дескриптор текущего экземпляра приложения.

hPrevInstance – дескриптор предыдущего экземпляра приложения.

lpCmdLine – командная строка приложения, за исключением имени программы.

nShowCmd – управление отображением окна.

* FormatMessage – функция, форматирующая строку сообщения. Она требует определения сообщения как вводимых данных.

DWORD FormatMessage(

DWORD dwFlags,

LPCVOID lpSource,

DWORD dwMessageId,

DWORD dwLanguageId,

LPTSTR lpBuffer,

DWORD nSize,

va\_list \*Arguments

);

dwFlags – параметры форматирования и способы интерпретации параметра lpSource.

lpSource – расположение определения сообщения.

dwMessageId – идентификатор запрошенного сообщения.

dwLanguageId – идентификатор языка запрошенного сообщения.

lpBuffer – указатель на буфер, который получает строку с завершающим нулем, определяющую форматированное сообщение.

Arguments – массив значений, которые используются в качестве значений вставки в форматированном сообщении.

**3. Библиотека processsthreadsapi.h**

* ExitProcess – функция, которая заканчивает работу процесса и всех его потоков, обеспечивая чистое отключение процесса.

void ExitProcess(

UINT uExitCode

);

uExitCode – код выхода для процесса и всех потоков.

**4. Библиотека winreg.h**

* RegQueryValueEx – функция получает значение параметра из раздела реестра. Это может быть число, строка и другие типы данных. Она извлекает тип и данные для указанного имени значения, связанного с открытым ключом реестра.

LSTATUS RegQueryValueExA(

HKEY hKey,

LPCSTR lpValueName,

LPDWORD lpReserved,

LPDWORD lpType,

LPBYTE lpData,

LPDWORD lpcbData

);

hKey – дескриптор открытого раздела реестра.

lpValueName – имя значения реестра.

lpReserved – этот параметр зарезервирован и должен иметь значение NULL.

lpType – указатель на переменную, которая получает код, указывающий тип данных, хранящихся в указанном значении.

lpData – указатель на буфер, который принимает данные значения.

lpcbData – указатель на переменную, определяющую размер буфера, на который указывает параметр lpData, в байтах.

* RegSetValueEx – функция корректно работает с нулевым указателем. Она устанавливает данные и тип указанного значения в разделе реестра.

LSTATUS RegSetValueExA(

HKEY hKey,

LPCSTR lpValueName,

DWORD Reserved,

DWORD dwType,

const BYTE \*lpData,

DWORD cbData

);

hKey – дескриптор открытого раздела реестра.

lpValueName – имя устанавливаемого значения.

Reserved – этот параметр зарезервирован и должен быть равен нулю.

dwType – тип данных, на который указывает параметр lpData.

lpData – данные для хранения.

cbData – размер информации, на которую указывает параметр lpData, в байтах.

* RegCloseKey – функция, закрывающая дескриптор указанного раздела реестра.

LSTATUS RegCloseKey(

HKEY hKey

);

hKey – дескриптор открываемого ключа, который нужно закрыть.

* RegQueryInfoKey – функция, которая получает информацию об указанном разделе реестра.

LSTATUS RegQueryInfoKeyA(

HKEY hKey,

LPSTR lpClass,

LPDWORD lpcchClass,

LPDWORD lpReserved,

LPDWORD lpcSubKeys,

LPDWORD lpcbMaxSubKeyLen,

LPDWORD lpcbMaxClassLen,

LPDWORD lpcValues,

LPDWORD lpcbMaxValueNameLen,

LPDWORD lpcbMaxValueLen,

LPDWORD lpcbSecurityDescriptor,

PFILETIME lpftLastWriteTime

);

hKey – дескриптор открытого раздела реестра.

lpClass – указатель на буфер, который принимает определенный пользователем класс ключа.

lpcchClass – указатель на переменную, определяющую размер буфера, на который указывает параметр lpClass, в символах.

lpReserved – этот параметр зарезервирован и должен иметь значение NULL.

lpcSubKeys – указатель на переменную, которая получает количество подключей, содержащихся в указанном ключе.

lpcbMaxSubKeyLen – указатель на переменную, которая получает размер подраздела ключа с самым длинным именем в символах Юникода, не включая завершающий нулевой символ.

lpcbMaxClassLen – указатель на переменную, которая получает размер самой длинной строки, определяющей класс подключа, в символах Юникода.

lpcValues – указатель на переменную, которая получает количество значений, связанных с ключом.

lpcbMaxValueNameLen – указатель на переменную, которая получает размер самого длинного имени значения ключа в символах Юникода.

lpcbMaxValueLen – указатель на переменную, которая получает размер самого длинного компонента данных среди значений ключа в байтах.

lpcbSecurityDescriptor – указатель на переменную, которая получает размер дескриптора безопасности ключа в байтах.

lpftLastWriteTime – указатель на структуру FILETIME, которая получает время последней записи.

* RegEnumValue – функция, перечисляющая значения для указанного открытого раздела реестра. Она копирует одно имя индексированного значения и блок данных для ключа при каждом вызове.

LSTATUS RegEnumValueA(

HKEY hKey,

DWORD dwIndex,

LPSTR lpValueName,

LPDWORD lpcchValueName,

LPDWORD lpReserved,

LPDWORD lpType,

LPBYTE lpData,

LPDWORD lpcbData

);

hKey – дескриптор открытого раздела реестра.

dwIndex – индекс значения, которое нужно получить.

lpValueName – указатель на буфер, который получает имя значения в виде строки с завершающим нулем.

lpcchValueName – указатель на переменную, определяющую размер буфера, на который указывает параметр lpValueName, в символах.

lpReserved – этот параметр зарезервирован и должен иметь значение NULL.

lpType – указатель на переменную, которая получает код, указывающий тип данных, хранящихся в указанном значении.

lpData – указатель на буфер, который получает данные для записи значения.

lpcbData – указатель на переменную, определяющую размер буфера, на который указывает параметр lpData, в байтах.

* RegDeleteValue – функция, удаляющая указанное значение из указанного раздела и подраздела реестра.

LSTATUS RegDeleteKeyValue(

HKEY hKey,

LPCSTR lpSubKey,

LPCSTR lpValueName

);

hKey – дескриптор открытого раздела реестра.

lpSubKey – имя раздела реестра.

lpValueName – значение реестра, которое нужно удалить из ключа.

**5. Функции-обработчики сообщений**

* ChooseParamTypeDlgProc – функция-обработчик сообщений диалога выбора типа параметра реестра.

INT\_PTR CALLBACK ChooseParamTypeDlgProc(

HWND hwnd,

UINT msg,

WPARAM wp,

LPARAM lp

);

* AddEditParamNameDlgProc – функция-обработчик сообщений диалога добавления/редактирования имени параметра реестра.

INT\_PTR CALLBACK AddEditParamNameDlgProc(

HWND hwnd,

UINT msg,

WPARAM wp,

LPARAM lp

);

* AddEditParamValueDlgProc – функция-обработчик сообщений диалога добавления/редактирования значения параметра реестра.

INT\_PTR CALLBACK AddEditParamValueDlgProc(

HWND hwnd,

UINT msg,

WPARAM wp,

LPARAM lp

);

* AddEditMultistringValueDlgProc – функция-обработчик сообщений диалога редактирования многострочного параметра реестра.

INT\_PTR CALLBACK AddEditMultistringValueDlgProc(

HWND hwnd,

UINT msg,

WPARAM wp,

LPARAM lp

);

**3.3. Разработка и реализация алгоритмов приложения**

Общий алгоритм работы приложения представлен на рисунке 3.2.

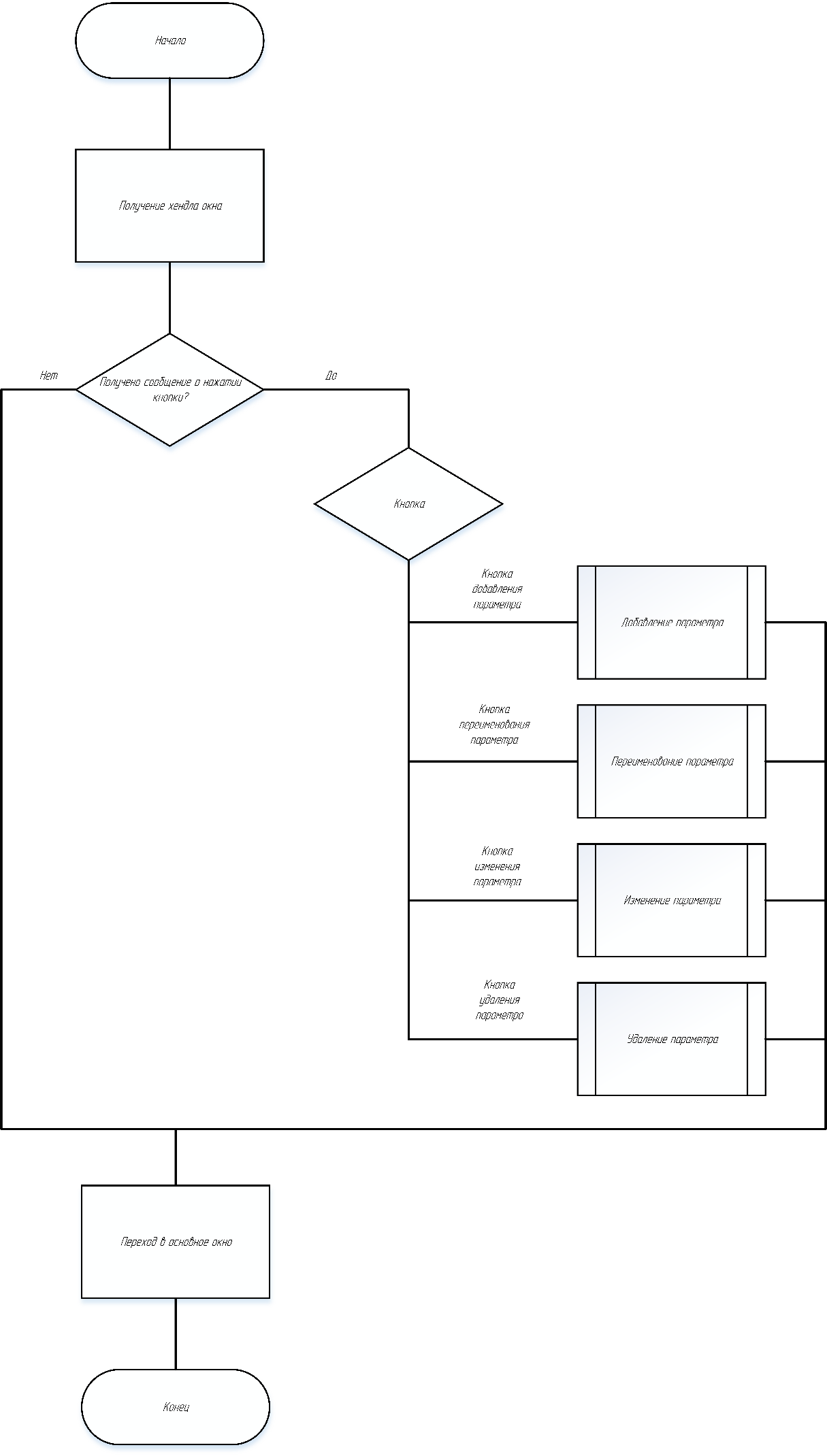


Рисунок 3.2 – Алгоритм работы приложения

Алгоритм выполнения программы при нажатии кнопки добавления параметра представлен на рисунке 3.3.

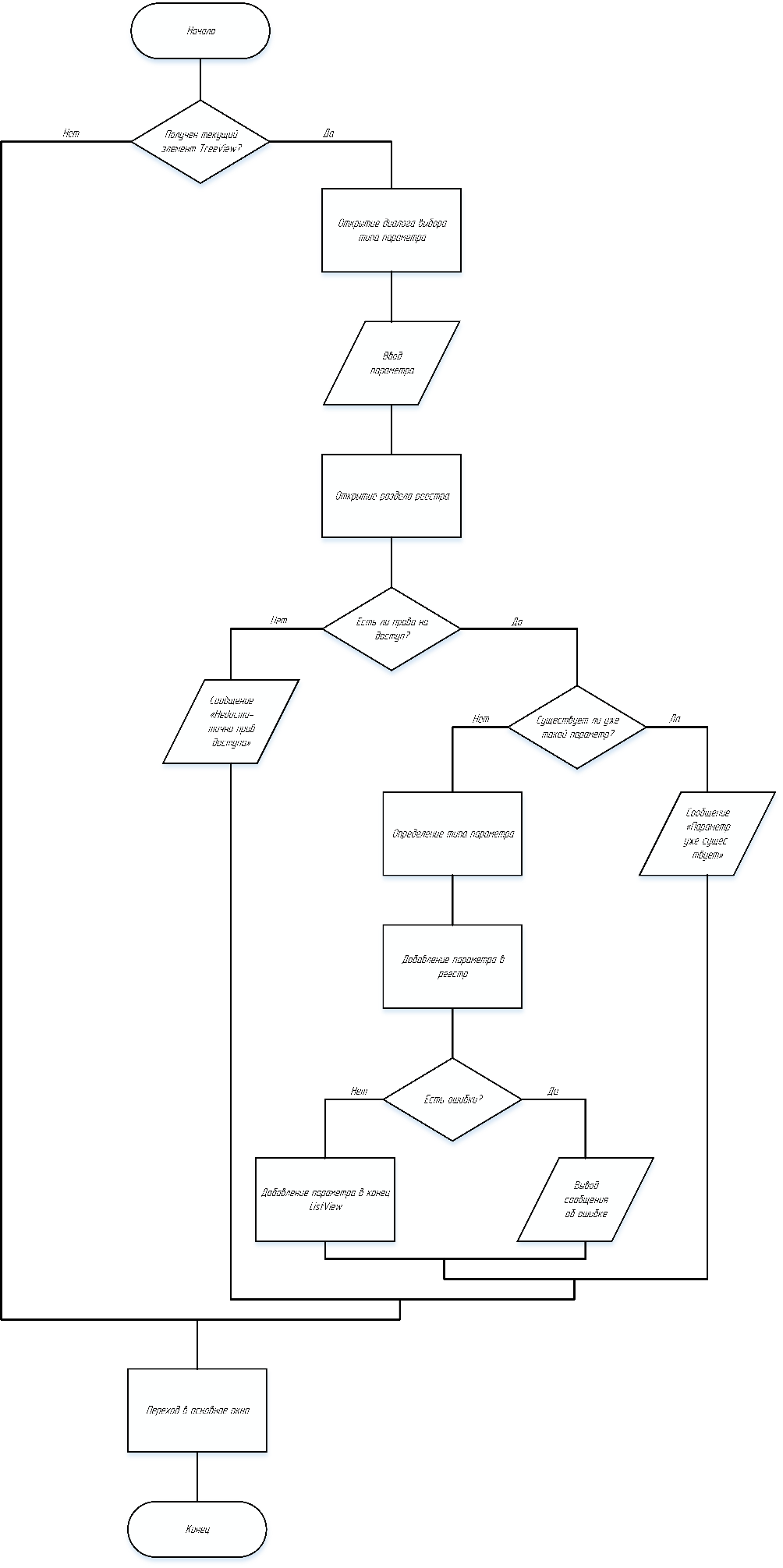


Рисунок 3.3 – Алгоритм работы функции добавления параметра

Алгоритм выполнения программы при нажатии кнопки переименования параметра представлен на рисунке 3.4.

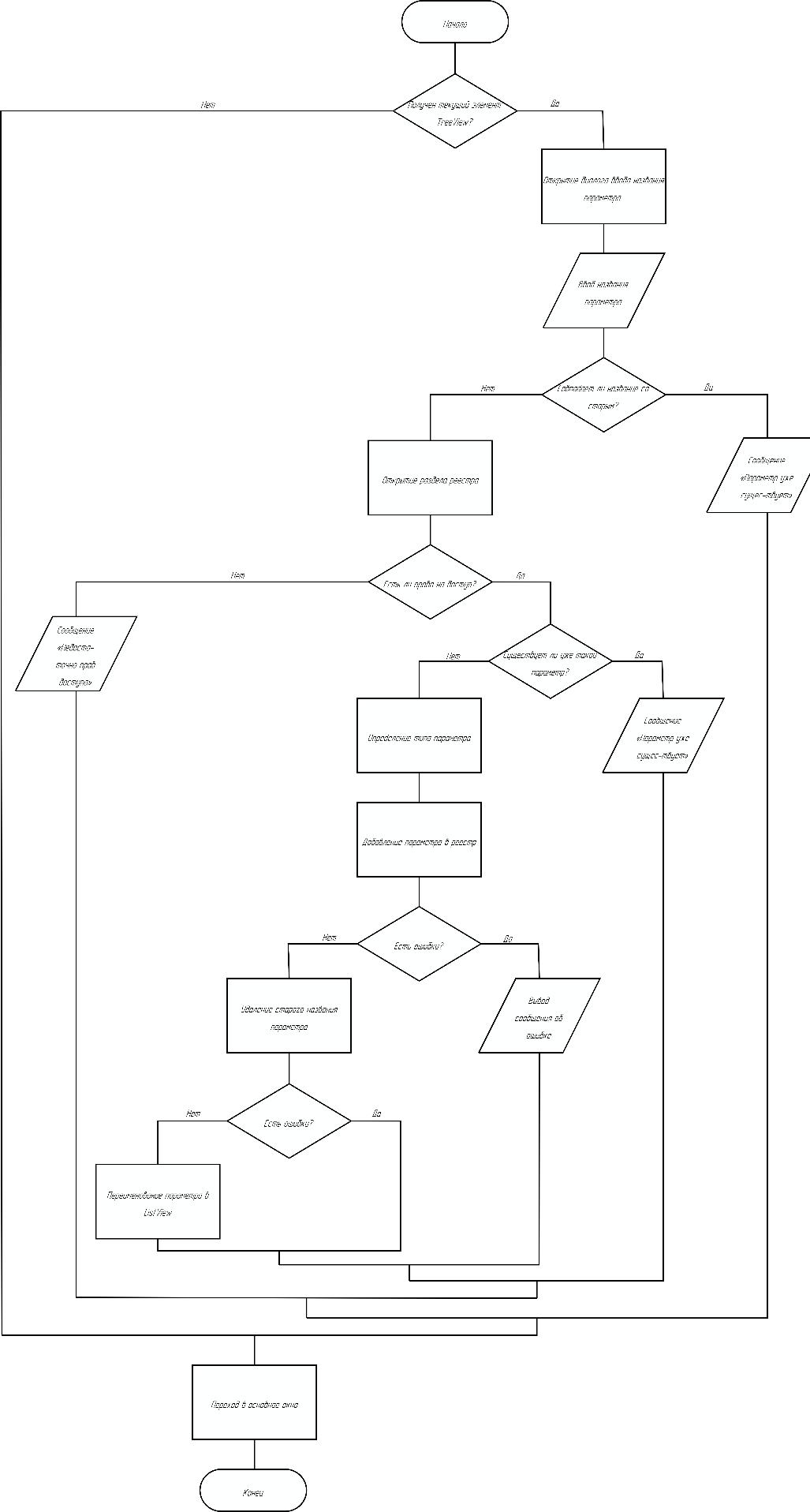


Рисунок 3.4 – Алгоритм работы функции переименования параметра

**3.4. Разработка интерфейса взаимодействия пользователя с системой**

Перед созданием интерфейса был разработан прототип, в котором было выбрано расположение кнопок и расположение меню. Прототип представлен на рисунке 3.5.

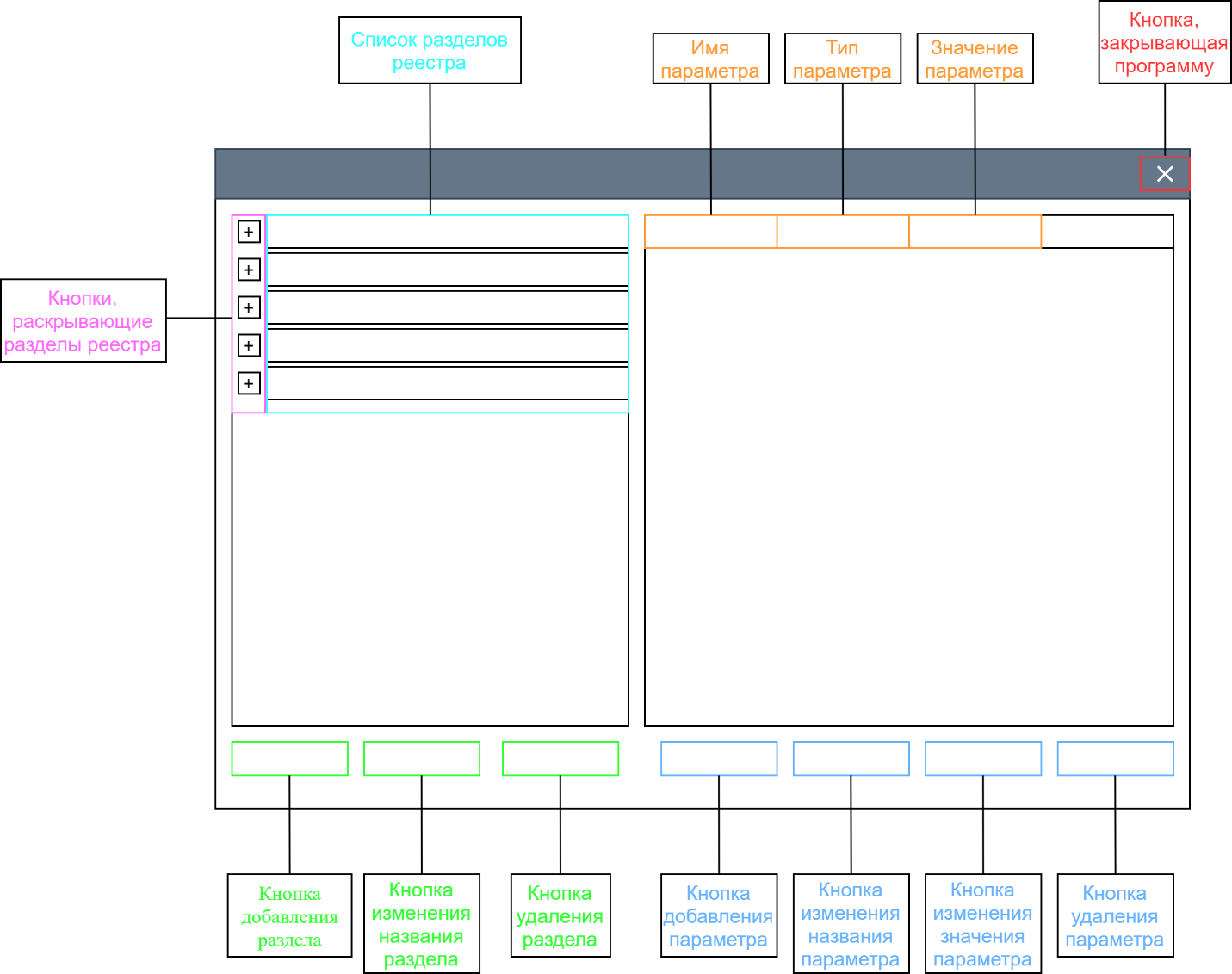


Рисунок 3.5 – Окно программы

При нажатии кнопки выбора типа параметра реестра вызывается функция ChooseParamTypeDlgProc. Данная функция позволяет осуществить выбор типа параметра: двоичные данные, DWORD, QWORD, строка, многострочная строка и расширяемая строка.

При нажатии кнопки редактирования названия параметра реестра вызывается функция AddEditParamNameDlgProc. Данная функция предоставляет возможность добавить название нового параметра или изменить название старого.

При нажатии кнопки редактирования значения параметра реестра вызывается функция AddEditParamValueDlgProc. Данная функция дает возможность добавить новое значение параметра реестра или изменить старое.

В ходе выполнения проектно-конструкторской части была разработана UML-диаграмма вариантов использования программы «Редактор реестра», разработаны и представлены алгоритмы приложения: общий алгоритм работы приложения, алгоритм работы функции добавления параметра и алгоритм работы функции переименования параметра, а также создан макет программы с выбранным расположением основного меню и кнопок для реализации интерфейса приложения.

В проектно-конструкторской части были использованы функции Win32API для работы с реестром: RegQueryValueEx, RegSetValueEx, RegCloseKey, RegQueryInfoKey, RegEnumValue, RegDeleteValue (библиотека winreg.h). Также были использованы функции-обработчики сообщений: ChooseParamTypeDlgProc, AddEditParamNameDlgProc, AddEditParamValueDlgProc, AddEditMultistringValueDlgProc.