МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

**Факультет «Системы управления, информатика и электроэнергетика» Кафедра 304**

**Направление подготовки 230100 Информатика и вычислительная техника Группа 3О-410Б**

**Квалификация (степень) бакалавр**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРА**

На тему: разработка программного обеспечения для взаимодействия с модулем GPS Trimble посредством спецоборудования

Автор квалификационной работы Сомов Дмитрий Николаевич ( )

(Фамилия, Имя, Отчество)

Руководитель Звонарева Галина Александровна ( )

(Фамилия, Имя, Отчество)

Консультант Звонарева Галина Александровна ( )

(Фамилия, Имя, Отчество)

Консультант Звонарева Галина Александровна ( )

(Фамилия, Имя, Отчество)

Рецензент Звонарева Галина Александровна ( )

(Фамилия, Имя, Отчество)

**К защите допустить**

Зав. кафедрой Брехов Олег Михайлович ( )

(Фамилия, И. О.)

“ ” 2018 г.

Москва 2018 г.

(здесь и далее будет окончательное задание)

Оглавление

[Список основных специальных терминов 5](#_Toc509507081)

[Введение 6](#_Toc509507082)

[Среда разработки приложений Qt Creator 8](#_Toc509507083)

[Подсистема формирования пакетов. Пользовательский графический интерфейс 11](#_Toc509507084)

[Создание графического интерфейса на языке QML 13](#_Toc509507085)

[I. Окна в Qt Quick 13](#_Toc509507086)

[II. Параграфы и стили элементов в Qt Quick 16](#_Toc509507087)

[III. Интерактивные элементы в Qt Quick. Блок отрисовки пользовательского интерфейса конфигурирования отправляемых пакетов 17](#_Toc509507088)

[IV. Сигналы и события в Qt Quick 21](#_Toc509507089)

[Способы взаимодействия между графическим интерфейсом QML и внутренними программными алгоритмами C++ 22](#_Toc509507090)

[I. Обращение к объектам и свойствам из С++ по их объектному имени 22](#_Toc509507091)

[II. Система слотов и сигналов между C++ и QML 24](#_Toc509507092)

[III. Использование объектных имен и ссылок, системы сигналов и слотов. Блок установки связей между подсистемой пользовательского ввода и подсистемой информационного обмена со спецоборудованием 25](#_Toc509507093)

[Блок отправки автоматически формируемых пакетов 30](#_Toc509507094)

[Блок отправки настраиваемых пакетов. Конфигурирование пакетов 32](#_Toc509507095)

[Подсистема информационного обмена с GPS-модулем. Интерфейс VirtualCOM 35](#_Toc509507096)

[Блок инициализации порта ввода/вывода VirtualCOM 38](#_Toc509507097)

[Блок интерфейса для обмена со спецоборудованием посредством порта VirtualCOM 40](#_Toc509507098)

[I. Блок чтения и логгирования данных, поступающих на порт VirtualCOM 40](#_Toc509507099)

[II. Блок записи данных в порт VirtualCOM 42](#_Toc509507100)

[Блоки подготовки информации к передаче в виде пакета 44](#_Toc509507101)

[Подсистема вывода приходящей информации на экран 47](#_Toc509507102)

[Блок текстового интерфейса вывода 49](#_Toc509507103)

[Блок разбора приходящих пакетов 52](#_Toc509507104)

[Блоки конвертации пришедшей информации перед ее выводом. Строение пакетов TSIP 54](#_Toc509507105)

[Тестирование готового программного продукта 57](#_Toc509507106)

[Тестирование графического интерфейса и его связи с программным кодом информационного обмена 59](#_Toc509507107)

[Тестирование взаимодействия с модулем GPS Trimble на стенде 61](#_Toc509507108)

[Исходные материалы и пособия 63](#_Toc509507109)

# Список основных специальных терминов

ВС – вычислительная система;

ПК – персональный компьютер;

ЦАП – цифро-аналоговый преобразователь;

COM-порт, Serial-порт – Communications port, порт последовательного интерфейса;

GPS – Global Positioning System, система глобального позиционирования;

QML – Qt Quick Markup Language, язык разметки Qt Quick;

OpenGL – Open Graphics Library, свободная графическая библиотека;

SV – space vehicle, космический спутник (имеется в виду спутник GPS);

TSIP – Trimble Standard Interface Protocol, стандартизированный протокол передачи данных Trimble;

VirtualCOM – программно эмулируемый последовательный порт

# Введение

В настоящее время технологии GPS являются одними из инновационных и пользуются значительной популярностью не только в гражданской сфере – GPS-модуль в мобильном телефоне позволяет в большинстве случаев быстро узнать своё местоположение и сориентироваться, оказавшись на незнакомой территории – но и в оборонных технологиях. Новейшая военная техника оборудуется системами глобального позиционирования, обеспечивающими достаточную точность, чтобы машины могли гарантированно выполнять свои прямые функции по обеспечению безопасности.

Но перед использованием такого оборудования в условиях действительности его необходимо подключить к ВС боевой машины с помощью специального оборудования, способного связывать несколько разнотипных устройств, а также протестировать взаимодействие спецоборудования и GPS-модуля. Эти операции позволяют с весьма маленькой погрешностью утверждать, как именно поведет себя готовая к боевым условиям ВС в критических ситуациях.

Конечно, набор команд для тестирования может быть и более тривиальным, так как необходимо оценить не только реакцию вычислительных систем на внешнюю угрозу, но и проверить общее взаимодействие различных модулей, могут ли они понимать друг друга и взаимодействовать с достаточными значениями параметров для успешного функционирования всего комплекса.

В данной работе речь пойдет о тестировании взаимодействия GPS-приемника и спецоборудования. Конкретная роль и строение спецоборудования в рамках данной работы неизвестно; считается лишь, что оно способно принимать и отправлять пакеты данных от ПК к GPS-модулю и от модуля GPS – персональному компьютеру, используя особый интерфейс.

В качестве GPS-модуля в техническом задании указан GPS-модуль Trimble Mini-T Thunderbolt (далее – GPS-модуль). Данный GPS-модуль является одним из недавних продуктов компании Trimble, и тестирование его взаимодействия со спецоборудования в рамках вышеуказанного технического задания является актуальной задачей для достижения поставленных целей.

Главной целью данной работы является установка связи с GPS-модулем посредством спецоборудования через интерфейс виртуального последовательного порта VirtualCOM, отладка взаимодействия с GPS-модулем посредством спецоборудования путем передачи пакетов информации различного содержания по каналу от GPS-модуля к ПК и обратно.

Задачи, поставленные в рамках данной работы:

* освоение среды разработки графический приложений Qt Creator и языка программирования С++, использующегося при программировании логики в Qt Creator;
* получение навыков работы с внешним оборудованием, подключенным через интерфейс последовательного порта; в качестве оборудования выбран GPS-модуль;
* проектирование программного обеспечения, обеспечивающего удобное для оператора взаимодействие с GPS-модулем посредством спецоборудования: получение полезной нагрузки с GPS-модуля и сведений о его состоянии, отправка конфигурационных пакетов (конфигурирование и управление GPS-модулем).

# Среда разработки приложений Qt Creator

Для создания программного продукта, который бы обеспечивал тестирование взаимодействия GPS-модуля и спецоборудования, подключенного через интерфейс VirtualCOM-порта к ПК, которым управляет оператор, была выбрана среда разработки Qt Creator. Это среда для пакета библиотек Qt, которая имеет множество возможностей по созданию пользовательских приложений c мощным набором функционала.

В Qt Creator имеется один из самых дружелюбных по отношению к пользователю редакторов интерфейсов. Он позволяет создавать графические интерфейсы приложений, собирая их из заранее подготовленных элементов. Примеры элементов:

--- ----- ---

Конечно, собранный интерфейс не может храниться как набор нарисованных элементов, которые видит обыватель, запуская программу. На низком уровне интерфейс представляет собой также программу, написанную на языке разметки, созданном специально для Qt Creator и сопутствующих библиотек, объединенных в один большой модуль разработки графических интерфейсов из компонентов – Qt Quick. Язык, который используется при программировании интерфейсов Qt Quick называется QML, и он весьма широко используется в приложениях, созданных в Qt Creator, имеющих графический интерфейс. О возможностях языка QML будет рассказано позже.

Графический интерфейс может быть эргономичным и обладать большим количеством функций, но зачастую им одним и функциями-помощниками языка QML обойтись не удается. В Qt Creator существует возможность писать основной программный код приложения на объектно-ориентированном языке программирования C++. Файлы исходного кода С++ логически отделены от файлов, наполненных языком разметки интерфейса, но разработано несколько программных механизмов для связи первых и вторых, о чём также будет рассказано позже. Не предполагается никаких ограничений на сложность алгоритмов, описанных на С++; более того, в Qt Creator (аналогично с набором библиотек Qt Quick) существует собственная библиотека для программирования на С++ с множеством классов, так или иначе имеющих отношение к реализации логики в данной среде: запуск графических интерфейсов, многопоточность, эмуляция последовательного порта, собственные типы данных и структуры, и так далее.

Пример использования класса QByteArray из одной из этих библиотек:

--- ----- ---

Собственные типы данных Qt:

--- ------ ---

Особая их реализация позволяет удостовериться, что на разных платформах скомпилированная программа будет исполняться с одинаковым результатом. Это важное условие для данной среды разработки, так как она позиционируется как среда для кросс-платформенного создания приложений, то есть, программный продукт, изготовленный под операционные системы настольных ПК, может быть легко и быстро переконфигурирован для выпуска на мобильные операционные системы, например, Android. Кросс-платформенность библиотек Qt является, несомненно, большим плюсом и привлекает специалистов по разработке мобильных и встраиваемых приложений; но в рамках данной работы эта функция использоваться не будет. Программное обеспечение для взаимодействия с GPS-модулем посредством спецоборудования, о котором будет изложено в данной работе, изначально проектировалось под операционную систему Windows для настольного ПК.

Другой немаловажной особенностью Qt Creator является его надстройка из графических библиотек OpenGL. Вообще говоря, элементы интерфейса Qt Quick уже (неявно для программиста) используют OpenGL для отрисовки своего содержимого на дисплее. Но при подключении определенного набора библиотек из этой надстройки возникает возможность проектировать интерфейсы и их элементы, отличные от тех, что собраны из стандартных элементов Qt Quick, и насыщенные современной двухмерной и трехмерной графикой. Это могут быть формы для отображения данных, таких, как спектрограммы и трехмерные таблицы; имитация полета спутников и других летательных аппаратов в пространстве; загрузка изображений, видеороликов и другой медиа-информации; и многое другое.

Стоит заметить, что, несмотря на все свои преимущества, выражаемые в возможностях и разнообразии графических форм для проектирования дизайна приложения, написание кода с использованием графических библиотек OpenGL представляет собой нетривиальную задачу. В рамках данной работы программирование интерфейсов с помощью OpenGL не рассматривается.

Можно долго перечислять всевозможные аспекты работы в Qt Creator – многопоточность, конфигурирование проектов, отладка и прочее – но в данной работе следует остановиться лишь на некоторых из них, которые повлияли на процесс проектирования и разработки программного продукта больше всего. О них речь и пойдет далее.

# Подсистема формирования пакетов. Пользовательский графический интерфейс

Программное обеспечение для взаимодействия с модулем GPS посредством спецоборудования требует наличия подсистемы пользовательского ввода, так как оператор, управляющий персональной ЭВМ с запущенным данным ПО, должен иметь возможность, вводя данные в программу и вызывая те или иные процедуры, осуществлять взаимодействие с GPS-модулем.

В рассматриваемом программном обеспечении подсистема пользовательского ввода представлена графическим пользовательским интерфейсом и несколькими файлами программного кода, решающего алгоритмические задачи и задачи по обработке данных скрытно от глаз оператора.

Графический пользовательский интерфейс состоит из двух окон. Основное окно содержит текстовую область, в которой логгируются события, происходящие с каналом информационного обмена (прием/передача пакетов, содержимое некоторых пакетов и так далее); вкладочное меню, содержащее несколько вкладок, каждая из которых предоставляет кнопки для отправки пакетов GPS-модулю и поля ввода и/или выпадающие списки для ввода пользовательских данных, которые используются перед отправкой настраиваемых пакетов для их конфигурирования; поля с текстовой информацией, которую графический интерфейс получает от программного кода. Дополнительное окно – это окно конфигурирования и установки подключения с GPS-модулем, имеет все необходимые поля для решения данной задачи. Графический интерфейс написан на языке QML.

Вышеупомянутый программный код написан на С++ и представлен несколькими блоками: блок автоматически настраиваемых пакетов, не требующих от пользователя ввода конфигурационных данных; блок отправки настраиваемых пакетов, которые используют введенные ранее оператором данные для изменения отправляемых байтов данных соответственно назначению пакета; блок установки связей между С++-кодовыми алгоритмами и графическим интерфейсом на языке QML.

Представляется разумным сначала подробно описать создание графического интерфейса и, в частности, язык QML, который был использован при этой разработке, а затем рассмотреть алгоритмы С++, используемые в описываемой подсистеме для обработки данных и команд пользовательского ввода.

# Создание графического интерфейса на языке QML

Язык разметки Qt Quick, или QML – особый язык программирования, разработанный для проектирования интерфейсов в среде разработки Qt Creator при использовании сборки библиотек элементов графического интерфейса. Данная сборка имеет название Qt Quick и довольно внушительна по своим размерам, поэтому будут приведены лишь те элементы, которые были изучены и использовались при создании программного продукта, описываемого в данной работе.

Так как язык разметки QML используется для программирования интерфейсов, состоящих из элементов сборки Qt Quick (а на низком уровне эти элементы – ничто иное, как такой же код на языке QML), можно считать эти два понятия тесно взаимосвязанными и употреблять их взаимозаменяемо.

Итак, далее речь идет о различных элементах Qt Quick, применявшихся при разработке графического пользовательского интерфейса программного продукта тестирования взаимодействия с GPS-приемником посредством спецоборудования, и приемы, которые были применены при данном проектировании и разработке. Приведенные на QML примеры также являются частью этого программного продукта.

## Окна в Qt Quick

Главная часть приложения с графическим интерфейсом Qt Quick – это окно приложения, в котором происходит отображение всех добавленных элементов. Могут также быть созданы дополнительные окна. Любое окно может быть открыто, скрыто, закрыто, перемещено по экрану.

Примеры окон Qt Quick: так выглядит главное окно ПО для взаимодействия с GPS-модулем посредством спецоборудования:

-------

А это - дополнительное окно для выбора конфигурации соединения со спецоборудованием и его установкой:

--------

Основное окно существует всегда, оно является зрительным изображением запущенной и функционирующей программы. В частности, с главным окном связаны некоторые важные методы взаимодействия алгоритмического кода (С++) и кода интерфейса (QML). Основное окно, тем не менее, может быть скрыто (важно то, что при этом оно не перестает существовать как программный объект). Дополнительное же окно может быть открыто и закрыто (а на уровне программного кода это означает «создано» и «уничтожено» соответственно) в любой момент по запросу извне, из другой части программы.

Итак, окно объявляется в QML следующим образом:

-------------------

Каждое объявление, приведенное подобным образом, порождает новый элемент интерфейса – объект соответствующего типа. Следуя этой логике, дополнительные окна – даже если они скрыты в данный момент – должны быть объявлены заранее (если не предполагается их динамическое создание):

--------------------

После запуска приложения главное окно отображается по умолчанию. Дополнительные окна могут быть показаны с помощью вызова функции show() и закрыты с помощью вызова функции close() (конечно, если окно больше не нужно; в противном случае имеет смысл скрыть его функцией hide(), а не закрывать):

------------------

В языке QML используется некоторое подобие объектно‑ориентированного стиля написания программного кода, поэтому объекты‑элементы интерфейса могут иметь свой id, зная который, возможно получить доступ к свойствам объекта и вызывать методы этого объекта с помощью оператора доступа по ссылке – “.”. Именно использование оператора доступа и можно заметить в приведенных выше примерах обращения с объектами‑окнами.

Как и любой элемент Qt Quick, каждый экземпляр Window имеет набор свойств, определяющих внешний вид и поведение данного объекта. Например, у главного окна задана ширина и высота по умолчанию,

-----------

а у дополнительного окна – вид модальности.

-----------------

Модальное (иначе - диалоговое) окно блокирует любое взаимодействие пользователя с остальными окнами приложения до тех пор, пока оно само не будет закрыто, т.е. пока диалог между ним и пользователем не будет закончен.

Язык QML приветствует построение интерфейса по иерархическому принципу. Основной является окно (Window) или параграф (Item), которые задаются каждый в отдельном файле. Если название файла разметки QML начинается с прописной буквы и содержит в себе окно или параграф, имя этого файла можно использовать для создания пользовательских элементов Qt Quick в других файлах разметки, а сами файлы – как шаблоны для этих элементов. Именно поэтому в листинге кода выше дополнительное окно имеет тип COMInit. Его можно было бы полностью определить внутри главного окна, присвоив, например, только id, но хорошим тоном считается разделение длинных (а также повторно используемых) кодов разметки QML, поэтому код разметки графического интерфейса окна инициализации VirtualCOM вынесен в отдельный файл:

------------------------------------

## Параграфы и стили элементов в Qt Quick

Параграф (Item) – это элемент Qt Quick, который может представлять собой контейнер для других элементов – кнопок, текстовых полей и т.д. – для их группировки и обращения с ними как с единым целым либо являться составляющей какого-либо из окон, также функционируя в качестве контейнера. У любого Item также есть уникальный id и стандартный набор свойств.

Используя элемент Loader (загрузчик), можно загружать параграфы динамически, подставляя их в какой-либо другой компонент. Ниже приведен пример реализации совместной работы загрузчика и таблицы вкладок (TabBar):

------------------------------

В данном случае алгоритм реализован следующим образом: при переключении вкладок элемента TabBar меняется индекс текущей вкладки currentIndex. Выбор параграфа Item, который будет загружен как страница очередной вкладки, привязан к номеру вкладки с помощью оператора switch-case. После выбора варианта вкладки соответствующий параграф загружается из файла, на который в свойстве loader.source дана ссылка. Начало работы загрузчика привязано к событию onCurrentIndexChanged, т.е. оператор switch-case выполняется после того, как индекс текущей вкладки сменился.

Хранение параграфов в отдельных файлах является обыкновенной практикой, так как при этом их экземпляры легко загружать через Loader, как в примере выше, или использовать эти экземпляры для наполнения других элементов (чаще всего – окон). Данный подход коррелирует с объектно-ориентированным подходом программирования: классы – это файлы с определенными в них видами элементов Item, а объекты-экземпляры классов – созданные в других файлах объекты с типом-именем файла, в котором задан параграф.

В QML также присутствует несколько стилей элементов на выбор. Их можно задать как для каждого отдельного файла или элемента в частности, так и указать общий стиль для всего запускаемого графического приложения целиком. Стиль задается в самом начале работы программы в коде С++ функцией QQuickStyle::setStyle() с указанием имени стиля в качестве аргумента. [11] Стилизация элементов Qt Quick может использоваться программистом для персонализации спроектированного и реализованного им графического приложения, либо для возможности подчеркнуть какие-либо графические, отображаемые элементы цветом или формой, так как различные стили Qt Quick предоставляют различные возможности по стилизации.

## Интерактивные элементы в Qt Quick. Блок отрисовки пользовательского интерфейса конфигурирования отправляемых пакетов

Окна, параграфы (а также контейнеры разметки (Layout), контейнеры прокрутки (ScrollView) и некоторые другие) – это элементы, в большинстве случаев невидимые в интерфейсе и несущие роль контейнеров, нежели интерактивных элементов. Но подавляющее большинство элементов языка QML представляют собой интерактивные (отображаемые) элементы, с которыми пользователь может взаимодействовать.

Блок отрисовки пользовательского интерфейса конфигурирования и отправки пакетов полностью состоит из файлов QML, в которых определены те или иные части интерфейса. Эти части представляют собой параграфы, наполненные интерактивными элементами. Главной частью пользовательского интерфейса является элемент TabBar, в который динамически с помощью элемента Loader подгружаются вкладки (данный подход был разобран выше, и был приведен пример его реализации в рассматриваемом в данной работе приложении). Каждая вкладка – это параграф с кнопками для посыла сигналов отправки пакетов, а также надписями и полями или переключателями для выбора, настройки, задания опциональных полей отправляемых пакетов. Пример одного из параграфов (---название файла---):

----------------------------------

Элементы, используемые в этом и других параграфах, разбираются ниже.

Самый простой тип элемента – надпись (Label). Надписи часто используются в совокупности с другими элементами Qt Quick, так как различные элементы пользовательского интерфейса требуют уточнения касательно их работы или назначения в том или ином приложении. Главные свойства надписи – это непосредственно сам текст надписи и его стиль, т.е. шрифт и написание. Так надпись выглядит при разработке интерфейса, в виде кода QML-разметки и в готовом приложении:

------------------

Второй наиболее часто используемый вид элемента – кнопка (Button). Это элемент, с которым может быть осуществлено взаимодействие, может быть нажат. К нажатию кнопки на низком уровне привязано QML-событие (то есть, нажатие кнопки посылает сигнал в обработчик сигналов QML). О сигналах речь пойдет позже.

У кнопки также есть отображаемый текст, ее стиль, стиль самой кнопки и некоторые свойства, характерные только для этого типа элементов, такие, как возможность нажатия.

Примеры кнопки QML:

-------------------

В разработанном интерфейсе кнопки повсеместно используются для отправки сигналов о том, что пользователь желает отправить тот или иной пакет по каналу в GPS-модуль.

Если речь заходит об интерактивных элементах, помимо кнопок, в Qt Quick реализованы переключатели (RadioButton) и отметки (CheckBox):

---------------------------

И у переключателя, и у отметки есть главное свойство “отмечен” – checked – которое принимает булево значение true, когда элемент находится в отмеченном состоянии. Если имеется только один ненастроенный переключатель, то он, вообще говоря, будет иметь поведение такое же, как и отметка; но если отметки – это самостоятельные элементы, то переключатели могут при переключении взаимодействовать. Это взаимодействие заключается в том, что, если несколько переключателей объединить в колонку (Column) или в любой другой контейнер и присвоить эту колонку группе кнопок (ButtonGroup) свойству buttons, то при включении одного переключателя все остальные выключаются, т.е. может быть выбран только один из переключателей. Ниже приведен пример реализации такого поведения:

----------------------------

Также, как и кнопка, переключатели и отметки посылают сигналы при изменении свойства checked для отслеживания момента, когда пользователь осуществляет свой выбор.

В некоторых случаях в интерфейсе должна иметься возможность вводить некоторый текст, данные, которые необходимы для дальнейшего функционирования программного алгоритма или изменения его поведения. Поля для ввода текста в Qt Quick называется TextField:

-----------------------

TextField имеет свойство text, которое содержит введенный на данный момент текст в поле и позволяет получить к нему доступ, и свойство placeholder, в котором задано содержимое, отображающееся, когда текста в поле нет. Это свойство чаще всего используется в качестве подсказки для пользователя, что именно можно или нужно вводить в это поле. При изменении текста в поле также посылается особый сигнал. Очевидно, что стиль текста в текстовом поле также может быть настроен.

Среди элементов Qt Quick также предусмотрены компоненты для осуществления пользовательского выбора. Таким образом, цифровой счетчик (SpinBox) и список выбора (ComboBox) используются как элементы для выбора определенного числового и текстового значения соответственно.

Главное свойство элемента SpinBox – это свойство value, оно отображает текущее выбранное в счетчике значение. Это значение может находиться в пределах, заданных в свойствах from и to. Данный элемент, как и все остальные, генерирует ряд сигналов, но они не рассматриваются в рамках данной работы.

Элемент ComboBox имеет свойство model, в котором задается набор значений (строк, чисел и т.д.) как список. В готовом элементе ComboBox при нажатии на него выбор осуществляется из значений этого списка. Можно узнать, какое именно выбрано значение в данный момент, если обратиться к свойству списка выбора currentIndex.

Типичные цифровой счетчик и список выбора показаны ниже:

------------------------------

Замечательно, что у всех отображаемых элементов есть id, зная который, можно обращаться к свойствам элемента, а также значения ширины, высоты элемента и его расположение относительно контейнера, которому он принадлежит (позиция по х и у).

## Сигналы и события в Qt Quick

В системе графического интерфейса Qt Quick на низком уровне разработана система сигналов и автоматического вызова некоторых событий, когда эти сигналы возникают где-то извне. Это означает, что какой-либо элемент интерфейса может генерировать программный сигнал, означающий, что произошло какое-либо событие. Тривиальные примеры сигналов: нажатие кнопки, открытие и закрытие окна, установка отметки CheckBox в положение «отмечено» или «не отмечено», смена вкладки TabBar, движение окна или курсора мыши и тому подобное. Без сигналов нужно было бы постоянно отслеживать изменения этих параметров, которые должны были бы измениться, что неизбежно влияло бы отрицательно на производительность кода, отвечающего за графический интерфейс.

Программист может воспользоваться преимуществом вызова особых методов по соответствующим сигналам; эти методы называются *событиями*, потому что они исполняются самостоятельно, когда происходит какое-либо событие (а в ответ на это событие отправляется сигнал, как было замечено выше).

Пример обработки события кнопки onClicked, которое всегда вызывается, когда происходит нажатие этой кнопки:

----------------------------------

Этот фрагмент демонстрирует, что в ответ на закрытие дополнительного окна с настройкой порта VirtualCOM вызовется событие onCloseWindow, и поэтому произойдет закрытие дополнительного окна, показ основного окна и отправка сигнала в среду С++, указывающее, что нужно открывать новое соединение через последовательный порт (о системе сигналов и слотов между С++ и QML будет рассказано позже):

------------------------------------

# Способы взаимодействия между графическим интерфейсом QML и внутренними программными алгоритмами C++

Существует несколько способов взаимодействия между кодом графического интерфейса, реализованного на языке QML, и внутренними программными алгоритмами, скрытыми от глаз обыкновенного пользователя приложения, написанными на С++. Ниже будут описаны те из них, которые применялись в ходе проектирования и реализации программного обеспечения для взаимодействия с GPS-приемником посредством спецоборудования.

## Обращение к объектам и свойствам из С++ по их объектному имени

В файлах QML принята иерархическая система элементов, при которой каждый элемент является родителем некоторых других элементов, а они, в свою очередь, являются его преемниками и, в свою очередь, сами могут быть родительскими элементами. Для поиска вниз по иерархической структуре элементов QML (от родителя к потомкам) в С++ определена целая группа специальных классов, повторяющих названия элементов Qt Quick. Несколько примеров: класс, который является общим для всех вышеописанных классов – QObject; объект класса QQuickWindow создается при запуске графического приложения и, по сути, является программным отображением главного окна этого приложения, и так далее.

Имея ссылку на объект главного окна приложения, можно также ссылаться на его преемников – то есть, на любой элемент интерфейса – и на свойства окна и свойства преемников. Получить ссылку на главное окно возможно в самом начале работы приложения из объекта типа QQmlApplicationEngine, без которого приложение с интерфейсом на QML не может существовать.

Итак, получение ссылки на какой-либо объект-преемник происходит с помощью метода класса QObject findChild<T>(const char \*objectName). Поиск «ребенка» происходит по его объектному имени; но замечательно то, что для возможности получения ссылки это имя должно быть явно задано программистом интерфейса, т.к. по умолчанию ни один из элементов QML не имеет объектного имени. Поиск элемента-преемника в данном случае не рекурсивный и происходит только на глубине одного вложения. В качестве Т может быть указан тип искомого объекта (название соответствующего класса С++); впрочем, в большинстве случаев достаточно указать вместо Т QObject \*, так как необязательно знать тип объекта, чтобы работать с его свойствами. [7]

После того, как ссылка на объект-элемент интерфейса получена, имеется возможность чтения и изменения его свойств. В классе QObject для этих целей есть два метода. Метод property(const char \*propertyName) возвращает значение свойства с заданным именем, а метод setProperty(const char \*propertyName, QVariant newValue) задает свойству с указанным именем новое значение, передаваемое вторым параметром. QVariant – это вспомогательный класс установки взаимодействия между С++ и QML (и обратно), который позволяет избежать конфликта типов величин, когда идет взаимодействие между двумя достаточно разными языками программирования. QVariant во время передачи может представлять собой любой тип данных, а для получения того типа данных, который необходимо было передать изначально в объекте QVariant, будет получен с помощью функций преобразования типов на стороне получателя.

Важно заметить, что при поиске объектов из С++ в QML и обращения к свойствам найденных объектов необходимо всегда проверять, была ли получена ссылка на элемент QML, так как если она не была получена корректно (объект не был найден по указанному имени, окно уже не существует и прочие подобные ошибки), то ссылка на объект будет равна нулевому указателю, а обращение по нулевому указателю неизбежно приведет к критической ошибке и падению программы. Поэтому подобные случаи следует обрабатывать, описывая соответствующий код исправления ошибок.

## Система слотов и сигналов между C++ и QML

Система сигналов и слотов Qt – это универсальная, программно реализованная система для передачи сигналов между двумя частями приложения, реализованными на разных языках программирования – графическом интерфейсе, написанном на QML, и алгоритмах обработки данных и логики приложения, написанных на С++. Благодаря данной реализации передачи сигналов интерфейсная часть может реагировать на события, происходящие в алгоритмической системе, а алгоритмы – принимать данные и вызывать функции по сигналам графического интерфейса. При этом сигнал не знает, «слушает» ли кто-то из методов его вызов, а слоты не знают, откуда получают сигнал для исполнения своего тела; это позволяет более легко проектировать и изменять исходный код приложения Qt, не обновляя множество зависимостей при каждом изменении структуры кода. [2] Тривиальный пример: вычислять или обрабатывать какие-либо данные каждый раз, когда нажата QML-кнопка; затем вычисленные данные передать для отображения обратно в QML-интерфейс; а чуть позже присоединить еще одну кнопку QML, не изменяя текущей связки сигнала и слота.

Данная система полностью спроектирована инженерами Qt Creator, программисту приложений остается лишь пользоваться ей, создавая свои слоты и сигналы либо используя встроенные сигналы (см. главу про интерактивные элементы Qt Quick). И слоты, и сигналы представляют из себя особые методы, объявленные и реализованные на языке С++ или QML. Сигналом называется метод стороны-отправителя, слот – принимающей стороны. После задания сигнала С++ его можно вызвать с помощью ключевого слова emit – при этом система слотов и сигналов получит оповещение, что данный сигнал был выпущен, вызван, и запустит выполнение соответствующего слота QML (чаще всего он представляет собой обыкновенную функцию, возможно, с параметрами). При вызове сигнала из QML ключевое слово не требуется – система заранее знает, что данная функция является сигналом, а не просто методом, и вызовет привязанный к нему слот в С++ (который также представлен стандартной функцией). Связывание происходит на этапе компиляции приложения с помощью специального метода из класса QObject под названием connect(). Параметры метода connect() более чем объясняют суть его работы: сигнал, указанный во втором параметре, являющийся методом объекта в первом параметре, будет привязан к слоту – четвертому параметру – объекта – третьего параметра. [4] Исчерпывающий пример будет приведен позже.

## Использование объектных имен и ссылок, системы сигналов и слотов. Блок установки связей между подсистемой формирования пакетов и подсистемой информационного обмена с GPS-модулем

При разработке программного обеспечения для взаимодействия с GPS-модулем посредством спецоборудования были изучены и использованы оба метода передачи информации между подсистемой пользовательского ввода (которая представлена написанным на QML графическим пользовательским интерфейсом) и подсистемой информационного обмена со спецоборудованием (реализованной с использованием языка С++). Далее будет приведен программный код, выполняющий описанные функции, и приведено его разъяснение.

1. **Рассмотрим создание и связывание сигнала С++ и слота QML на примере связки, отправляющей данные, сгенерированные в коде С++, в текстовую область QML-интерфейса.**

Сигнал С++ задается в классе в обособленном разделе signals и представляет собой обычный метод, который имеет параметры:

--------------------------------------------

Обычно методы С++ в разделе объявлений не имеют имен формальных параметров, однако здесь это необходимо, чтобы обращаться к этим параметрам внутри соответствующего QML-слота.

Далее создается слот в коде QML, представляющий собой обыкновенную QML-функцию, он размещен иерархически в составе главного окна Window данного приложения:

---------------------------------------------

Он должен иметь столько же параметров, сколько и соответствующий С++-сигнал.

Осталось только привязать сигнал к слоту, чтобы второй реагировал на вызов первого и исполнялся в это время. Данный сигнал генерируется, когда необходимо отослать какое-либо сообщение о принятых данных или об инициализации последовательного порта в область логгирования (текстовую область интерфейса), чтобы данное сообщение было отображено на дисплее. Используется функция connect():

-----------------------------------------

Первым параметром указывается ссылка на объект receiverThread, так как это объект класса COMHandler, в котором и был определен сигнал, указанный вторым параметром. Таким образом, слот, указанный четвертым, будет реагировать только на указанный сигнал из этого объекта. Для сигнала указывается его сигнатура, заключенная в специальный макрос SIGNAL(). Как и макрос SLOT() для макроса, он служит опознавательным знаком для системы сигналов и слотов, занося эти ссылки на эти методы в специальный список методов, вызов которых необходимо отслеживать. Третьим параметром указывается объект (элемент интерфейса QML), который содержит определение функции-слота, сигнатура которой указывается четвертым параметром. В данном случае это главное окно приложения mainWindow, ссылка на которое была получена раньше и содержится в классе-помощнике QMLDataHelper. Важно, чтобы количество и тип параметров сигнала и слота совпадал, а также, чтобы и сигнал, и слот были доступны в соответствующих объектах, иначе связки не произойдет, и слот не будет слышать вызова сигнала.

1. **Создание и связывание сигнала QML и слота C++** производится следующим образом:

Данная часть механизма рассматривается на примере передачи сигналов из интерфейса в логическую часть (из QML в С++), которые оповещают логическую часть приложения о том, что пользователь запросил формирование и/или отправку того или иного пакета. Связка передает также два параметра – это код и подкод отправляемого пакета. Это часть протокола связи с GPS-модулем. Этот протокол будет рассмотрен позже.

Сигнал явно объявляется в коде QML как часть некоего элемента (объекта) в иерархии элементов, в данном случае – вложен в главное окно приложения. Названию сигнала предшествует ключевое слово signal, указывается полный список аргументов с типами и названиями:

--------------------------------------

Слот необходимо задать в специальном разделе класса С++ slots, впрочем, его сигнатура и определение выглядят также, как у обычного метода класса:

----------------------------------------

Далее, так же, как и в случае обратного направления передачи данных, рассмотренном выше, сигнал и слот необходимо связать с помощью метода connect(), сигнатуры сигнала и слота должны совпадать, и сигнал и слот должны быть частью указанных объектов QML и С++ соответственно:

--------------------------------------

Здесь сигнал, как было отмечено выше, принадлежит главному окну mainWindow, ссылка на которое есть во вспомогательном классе, а слот – объекту handler класса COMHandler. После соединения каждый раз, когда сигнал sig\_send\_command будет вызываться внутри QML как функция, слот send\_command() в С++-классе COMHandler будет выполняться с переданными параметрами:

-------------------------------------

1. **Для формирования пакетов, требующих определенного предварительного пользовательского ввода, необходимо получить значения свойств настроенных пользователем элементов QML с помощью обращения по объектному имени.**

Яркий пример – конфигурирование текущих настроек ввода-вывода GPS-модуля; отправке пакета с новыми настройками предшествует выбор опций ввода-вывода в графическом интерфейсе, установка переключателей и отметок:

---------------------------------------------

При нажатии кнопки «Отправить новые настройки» логическая часть приложения получает сигнал sig\_send\_command и исполняет слот send\_command() с кодом, означающим, что требуется считать выставленные пользователем настройки и сформировать и отправить соответствующий пакет GPS-модулю.

Для облегченного использования методов findChild(), property() и обработки ошибок, связанных с их выполнением, был создан вспомогательный класс QMLDataHelper. Он содержит постоянную ссылку на главное окно – корневой объект для поиска всех остальных элементов – и методы для поиска свойства элемента по заданному имени элемента и имени свойства:

----------------------------------------------

Так можно получить свойство checked элементов-переключателей и отметок и затем сформировать пакет для отправки:

--------------------------------------

1. **Блок интерфейса вывода состояния GPS-модуля – яркий пример использования объектных имен элементов QML для изменения значения их свойств из кода С++.**

Когда GPS-модуль присылает полезную нагрузку – данные о состоянии, температуре, местоположении, отслеживаемых спутниках и т.п. – логическая часть приложения С++ обрабатывает данную информацию, расшифровывает ее и выводит некоторые из принятых данных на дисплей. Это означает, что новые данные передаются из С++ в графический интерфейс. Для этого достаточно получить ссылку на соответствующий элемент-надпись в интерфейсе QML и, используя метод setProperty(), установить обновленное значение свойства.

Ниже показано выполнение вышеописанной операции для вывода списка отслеживаемых GPS-модулем спутников:

-------------------------------

Вывод значений температуры и позиции, вычисленных GPS-модулем:

-------------------------------

# Блок отправки автоматически формируемых пакетов

Этот блок является одним из алгоритмических блоков описываемого программного обеспечения, реализованных на языке С++. Блок автоматического формирования простых пакетов и их отправки представлен непосредственно частью метода send\_command(), который принадлежит классу COMHandler, так как этот класс занимается всеми операциями, которые связаны с процессом информационного обмена на низком уровне, в том числе сюда относится формирование и отправка всех видов пакетов. Более подробно класс COMHandler рассматривается в следующей разделе данной работы.

Метод send\_command() является слотом класса COMHandler. Это означает, что он связан с сигналом QML. Связанный сигнал называется sig\_send\_command и отправляется каждый раз, когда оператор нажимает на любую из кнопок отправки пакета. Вместе с самим сигналом в код С++ передаются два параметра: код и подкод пакета, который требуется сформировать и отправить.

Некоторые пакеты не требуют добавления специальных данных, а содержат лишь начало и конец пакета (согласно разделу документации GPS-модуля о структуре пакета [6]) и один или два байта между ними – код пакета или код и подкод пакета. Такие короткие пакеты, не содержащие байтов данных, называются запросами. Ниже приведен пример запроса настроек ввода-вывода GPS-модуля:

-------------------

Блок отправки автоматически формируемых пакетов занимается именно подобными пакетами-запросами. Он добавляет к основе пакета байты кода и подкода (если последний указан для отправляемого в данном пакете запроса) и, минуя дальнейшую настройку пакета, завершает его байтовой последовательностью, означающей конец пакета, и отправляет в канал передачи данных. Листинг данного блока можно изучить ниже.

-----------------------

-------------------(тут схема алгоритма)

# Блок отправки настраиваемых пакетов. Конфигурирование пакетов

Помимо формирования и отправки пакетов-запросов, автоматически собираемых из кода пакета (и подкода, если он требуется), рассматриваемое в рамках данной работы программное обеспечение также способно отправлять пакеты конфигурирования и задания первоначальных настроек GPS-модуля. Для выполнения этих действий требуются пакеты, информационная часть которых может быть изменена оператором посредством выбора различных переключателей, полей в выпадающем списке, значений счетчиков, ввода пользовательских данных в текстовые поля и так далее. Графический интерфейс, который используется для визуализации процесса конфигурирования, и его устройство и разработка были рассмотрены ранее, теперь же стоит описать процесс, который происходит после того, как все необходимые значения в интерфейсе выбраны, и оператор нажимает кнопку отправки настраиваемого пакета.

---------------------- (пример вкладки конфигурирования пакета)

Для передачи информации о том, что оператор требует отправить тот или иной пакет, в коде QML используется один и тот же сигнал sig\_send\_command, и значения, передающиеся вместе с ним, те же – это код и подкод пакета. Сигнал принимается тем же слотом в коде С++, что был рассмотрен ранее – send\_command(), однако после добавления в пакет кода пакета и подкода (если он был указан для данного пакета) отправляемый пакет дополнительно настраивается. Данный процесс начинается с выбора метода, который бы подошел для настройки данного пакета. Выбор осуществляется по коду пакета, а в некоторых случаях пакеты различаются еще и подкодом: например, протокол GPS-модуля Trimble TSIP дает определение нескольким пакетам с одинаковым кодом пакета 0х8Е, которые называются суперпакетами. При отправке такого пакета стоит учесть, что они различаются с помощью именно подкода. Типичные исходящие суперпакеты протокола TSIP:

-----------------------------

Все методы конфигурирования настраиваемых пакетов реализованы в C++-классе CommandBuilder. Каждый метод, как упоминалось ранее, служит для настройки определенного пакета. Основная задача этих методов – получение данных о произведенной настройке пакета из интерфейса QML (из элементов, которыми наполнена вкладка, на которой может настроить этот пакет оператор). Также в некоторых методах происходит преобразование нескольких значений булевого типа (переключатели и отметки имеют значение этого типа, которое означает «отмечен»/«не отмечен») в битовое поле. Битовым полем называется байт данных, в котором биты по отдельности означают включение или выключение, присутствие или отсутствие каких-либо параметров.

Пример метода, конфигурирующего пакет изменения настроек ввода-вывода GPS-модуля:

-------------------------------

Еще один пример метода, который настраивает пакет:

-------------------------------

Данные методы специально запрограммированы таким образом, чтобы в них можно было передать настраиваемый пакет по ссылке. Метод вносит все необходимые изменения в пакет и возвращает его обратно по той же ссылке. После выполнения метода из класса CommandBuilder пакет завершается байтами конца пакета и отправляется в канал передачи данных.

Также данным классом и указанным выше сигналом sig\_send\_command пользуются несколько кнопок интерфейса, расположенных рядом с окном логгирования приходящей информации: кнопка для полной остановки рассылки GPS-модулем пакетов на компьютер оператора, кнопка для активации (возобновления рассылки) и кнопка для единоразового обновления всех величин, которая вызывает метод отправки сразу нескольких пакетов-запросов с целью получения обновленных значений важной информации. Эти элементы интерфейса отвечают за настройку авторассылки пакетов одним нажатием кнопки и за обновление всей важной информации, отображаемой в интерфейсе описываемого программного обеспечения, опять же, одной кнопкой.

Важным нюансом добавления пользовательской информации в пакет является тот факт, что все байты в пакете, которые оказались равны байту DLE (значение 0х10), должны быть экранированы. Аспекты и реализация экранирования байтов DLE при формировании отправляемых пакетов в рассматриваемом программном продукте будут рассмотрены позже.

Ниже приведена схема общего алгоритма работы блока отправки настраиваемых пакетов.

--------------------------------

# Подсистема информационного обмена с GPS-модулем. Интерфейс VirtualCOM

При использовании разработанного ПО для взаимодействия с GPS-модулем посредством спецоборудования программное обеспечение запускается на ПК под управлением оператора, а GPS-модуль подключен к ПК с помощью кабеля USB-MicroUSB посредством спецоборудования. Задачи спецоборудования являются коммерческой тайной; известно лишь то, что оно также служит преобразователем между интерфейсами MicroUSB (т.е. протоколом информационного обмена USB со стороны ПК) и COM (протокол последовательного информационного обмена со стороны GPS-модуля). Таким образом, GPS-модуль использует последовательный порт для приема и передачи данных. Чтобы разрешить различия между двумя интерфейсами, на ПК для передачи по каналу USB пакеты на самом деле передаются как информация для канала последовательного интерфейса. Такая возможность обеспечивается с помощью виртуального интерфейса VirtualCOM – он является адаптером (преобразователем) между интерфейсами USB и СОМ.

Ниже приведена формализованная схема аппаратных средств ПО для взаимодействия с GPS-модулем посредством спецоборудования:

-----------------------------------------------

Интерфейс порта VirtualCOM работает следующим образом: имеется физический интерфейс USB, к которому подключен кабель для передачи и приема данных от внешнего устройства (в рамках данной работы это – GPS-модуль). На ПК создается виртуальный последовательный порт. Когда информация передается через порт USB на компьютер, она имеет вид последовательных сигналов. Из-за этого драйвер USB-порта не может работать с этой информацией (она не имеет вид USB-пакета), однако, он «видит», что какая-то информация всё-таки имеется, и с ней что-то нужно сделать. Виртуальный последовательный порт выступает в роли того, кто может разрешить данную ситуацию: информация передается ему, и уже этот драйвер обрабатывает ее, после чего остальные программы, считывающие информацию с этого порта (в данном случае речь идет о ПО для взаимодействия с GPS-приемником), могут работать с полученными данными. [4]

Драйвер виртуального СОМ-порта получает входящую информацию, потому что постоянно опрашивает USB-драйвер на наличие входящих данных. Для обозначения подобной связи существует термин «привязка» (англ. mapping); в данном случае соответствующий USB-порт привязывается к порту COMxx, где хх – номер доступного порта. Привязка происходит на программном уровне, виртуально, так как на самом деле компьютер не имеет физических COM-интерфейсов – лишь интерфейс VirtualCOM. При этом приложение, описываемое в данной работе, после привязки может использовать этот связанный USB-порт как последовательный интерфейс. Когда же нужно что-то отправить в интерфейс VirtualCOM через USB-порт, приложение побайтно посылает данные в виртуальный интерфейс, который передает эти байты в интерфейс USB. Там байты оборачиваются в пакеты USB и передаются через интерфейс с параметрами виртуального COM-порта (скорость и прочие параметры). Это необходимо, так как известно, что на стороне внешнего устройства пакеты будут обратно преобразованы интерфейсом MicroUSB в Serial-информацию. [5]

Среда разработки приложений Qt Creator предоставляет специальный класс для создания виртуального последовательного порта и использования его для записи и чтения информации под названием QSerialPort. [8] В этом классе определены методы для получения информации о данных, с которыми в данный момент работает VirtualCOM-порт (входящие и исходящие байты), информации о состоянии VirtualCOM-порта; также имеются методы и константы для конфигурирования и открытия/закрытия VirtualCOM-порта. Далее в ходе проектирования и разработки ПО для взаимодействия с GPS-модулем посредством спецоборудования был разработан класс COMHandler, подключающий функционал класса QSerialPort (это означает, что в классе COMHandler доступны все методы и открытые переменные класса QSerialPort), и в нем реализованы методы открытия, закрытия порта VirtualCOM, чтения и записи потока байтов, передаваемых через данный интерфейс. Также он хранит информацию об открытом порте (ссылку на него) и прочтенные из порта, но еще не обработанные данные.

# Блок инициализации порта ввода/вывода VirtualCOM

Перед использованием порта VirtualCOM для связи с GPS-модулем его необходимо создать, т.е. проинициализировать. В инициализацию входят создание объекта класса порта VirtualCOM, конфигурирование порта, то есть установка настроек передачи данных по последовательному каналу, и его очистка и открытие в режиме чтения и записи [3].

В пользовательском графическом интерфейсе инициализация порта VirtualCOM представлена отдельным окном, которое открывается при запуске приложения взаимодействия с GPS-модулем, а также при нажатии кнопки «Настройка VirtualCOM» в главном окне. Так выглядит окно инициализации порта VirtualCOM:

-----------------------------------

Очевидно, что на программном уровне это – элемент QML Window, сохраненный отдельно в файле COMInit.qml, чтобы его можно было использовать как объект при определении основного окна. В это вспомогательное окно добавлены различные интерактивные компоненты, такие, как надписи, текстовое поле имени порта и списки выбора различных настроек. По умолчанию заданы те настройки, которые наиболее подходят для установки связи с GPS-модулем, согласно документации. [6] Ниже приведен фрагмент кода этого окна на QML:

--------------------------------------

При нажатии кнопки «Подключиться» вызывается сигнал closeWindow, объявленный в начале файла. Он передает основному окну все выбранные настройки:

-------------------------------

Важно заметить, что при обычном закрытии окна без нажатия этой кнопки инициализации порта VirtualCOM не произойдет.

Главное окно получает отправленный окном настройки порта сигнал closeWindow и выполняет его определение onCloseWindow, предоставленное внутри объекта, представляющего собой это окно настройки (именно этот объект-окно и отображается на дисплее. Окно в файле – лишь определение того, чем является пользовательский элемент COMInit). При этом происходит автоматическое закрытие дополнительного окна, показ основного окна и отправка сигнала sig\_open\_port в область кода С++:

---------------------------------

В отличие от связывания слотов и сигналов между С++ и QML, когда имена имеют небольшое значение, при создании пользовательских сигналов QML внутри, опять же, QML-кода имеется особая договоренность. Сигнал и его вызов может называться singalName, но его определение – событие, которое произойдет при получении сигнала – должно иметь имя *onS*ignalName, т.е. «при сигнале signalName». Поэтому сигнал нажатия на кнопку инициализации порта назван closeWindow, а его событие имеет имя onCloseWindow. [12]

Сигнал sig\_open\_port отправляется с помощью уже рассмотренной системы слотов и сигналов в логическую часть приложения на С++. Он связан с методом класса COMHandler configureCOM(). Параметры, передаваемые из среды QML в С++-код, всё те же – это настройки, которые нужно установить порту VirtualCOM перед его открытием. После создания объекта COMHandler::com типа QSerialPort, который на программном уровне и представляет собой порт ввода-вывода VirtualCOM, и установки настроек [3], соответствующих полученным из пришедшего сигнала, порт открывается для записи и чтения, и в текстовую область, куда записываются все события, отслеживаемые приложением (область логгирования), отправляется сообщение об успешном открытии порта:

-------------------------

# Блок интерфейса для обмена со спецоборудованием посредством порта VirtualCOM

Как только порт готов к работе, можно использовать его для чтения и записи информации. Для реализации этих возможностей было спроектировано несколько алгоритмов, речь о которых пойдет ниже. Эти алгоритмы составляют блок интерфейса для обмена со спецоборудованием посредством порта VirtualCOM, который состоит из блока чтения и логгирования данных, поступающих на порт VirtualCOM, и блока записи данных в порт VirtualCOM.

## Блок чтения и логгирования данных, поступающих на порт VirtualCOM

Для получения данных, поступивших на порт VirtualCOM, необходимо либо проверять состояние порта – есть ли какие-нибудь непрочитанные данные в его регистрах – с помощью метода waitForReadyRead()[10], либо отлавливать приход сигнала readyRead из класса QSerialPort, который оповещает о том, что VirtualCOM-порт принял новый байт данных и готов передать его остальной части приложения. Был выбран второй способ [9], и сигнал readyRead был связан с помощью системы слотов С++ и сигналов С++ с пользовательским слотом readFromCOM(). Это связывание происходит каждый раз после открытия порта, так как нельзя читать данные из закрытого или несуществующего порта:

-------------------------------

Cлот readFromCOM() представляет собой метод класса COMHandler, который занимается побайтным считыванием данных. Обработка входящей информации специально осуществляется побайтно, так как сигнал readyRead всегда приходит столько раз, сколько байтов поступило на порт (т.е. по одному сигналу на каждый пришедший байт данных). Рассмотрим код данного метода:

------------------------------

Все считанные байты хранятся в массиве readedData. Байт, прочитанный при прошлом вызове readFromCOM(), содержится в переменной previouslyReadedChar.

На самом деле в readedData байт за байтом собирается пакет, пришедший от GPS-модуля. Так как информационный обмен с GPS-модулем осуществляется порционно, пакетами, то и обрабатывать принятые данные имеет смысл только после того, как принят целый пакет. Как только будет обнаружен конец пакета, будет вызван метод receiveReport(), который занимается обработкой входящих пакетов и публикует содержащуюся в пакете информацию в текстовую область графического пользовательского интерфейса, выпуская сигнал appendReceivedText, и назначает надписям в интерфейсе полученные значения, если была принята какая-либо важная информация. Этот метод будет рассмотрен позже.

Переменная previouslyReadedChar позволяет обнаружить байты экранирования DLE и конец пакета, состоящий из последовательности DLE-ETX, так как в обоих случаях достаточно знать, какой байт пришел в прошлый раз. Если это был DLE, и сейчас принят DLE, значит, это экранирование байта DLE, и второй раз добавлять его в массив readedData не нужно. Если это был DLE, а сейчас это – ETX, то обнаружен конец пакета. Информация о структуре пакета и байтах экранирования приведена в документации к TSIP – протоколу информационного обмена с устройствами Trimble, в число которых входит используемый в данной работе GPS-модуль Trimble Mini-T Thunderbolt. Согласно документации, пакет начинается с байта DLE (0x10) и заканчивается последовательностью DLE-ETX (0x10 0x03). Чтобы не перепутать конец пакета с частью пакета, являющейся полезной нагрузкой и случайно содержащей последовательность DLE-ETX, все байты DLE в части полезной нагрузки экранируются, т.е. перед каждым байтом DLE ставится еще один байт DLE. Таким образом, только конец пакета будет выглядеть как DLE-ETX и содержать нечетное количество байтов DLE перед байтом ETX, а все случайные совпадения будут иметь четное количество байтов DLE перед байтом ETX: [6]

-------------------------------------

Экранирование байтов осуществляется и на стороне оператора (когда пакет отправляется от оператора к GPS-приемнику, о чём упоминалось ранее), и на стороне GPS-приемника (когда GPS-приемник шлет пакет оператору). Разумеется, перед обработкой пакета необходимо удалить все лишние (то есть экранирующие) байты DLE из последовательности, составляющей пришедший пакет. Поэтому в коде метода readFromCOM(), приведенного выше, реализован следующий алгоритм:

-------------------------------------------

## Блок записи данных в порт VirtualCOM

Вместе с блоками инициализации порта VirtualCOM и чтения данных, приходящих на этот порт, в классе COMHandler реализован блок, который занимается записью данных в порт VirtualCOM для передачи их по каналу связи GPS-модулю. Данный блок представлен одним методом-слотом send\_command().

Когда оператор путем нажатия кнопки в графическом интерфейсе приложения подает команду отправить какой-либо пакет GPS-модулю, код интерфейса QML отправляет с помощью системы слотов и сигналов сигнал sig\_send\_command коду С++, а именно – связанному с ним слоту send\_command(). Этот метод почти всегда вызывается вышеупомянутым сигналом (исключение составляет метод класса COMHandler requestEssentialValues(), служащий для запроса важных значений для отображения их в интерфейсе), и поэтому запись в порт VirtualCOM можно отождествлять с командами оператора об отправке тех или иных пакетов.

Метод send\_command() собирает пакет; о сборке как тривиальных пакетов-запросов, так и комплексных настраиваемых пакетов было рассказано ранее в соответствующем разделе. После успешной сборки пакета он отправляется в канал связи, представленный программно виртуальным портом VirtualCOM, методом QSerialPort::write(). При отправке указывается полное содержимое пакета, его длина, а также максимальная отсрочка (реальная может быть и меньше) перед следующей возможной записью, чтобы текущая запись информации в порт успела пройти без осложнений. Ниже приведен отрывок кода метода send\_command() на С++, отвечающий непосредственно за отправку пакетов и запись в порт VirtualCOM:

-------------------------------------

Стоит заметить, что, так как эта операция осуществляется довольно просто, нет необходимости строить какие-либо сложные алгоритмические схемы для реализации их в блоке записи данных в порт VirtualCOM.

# Блоки подготовки информации к передаче в виде пакета

Протокол TSIP использует в качестве последовательности, указывающей на конец пакета, последовательность байтов DLE-ETX (значение 0х10 0х03). При этом существует важный нюанс: такая последовательность может случайным образом встретиться и в информационных байтах пакета, в таком случае она будет ошибочно распознана как конец пакета, и остальная часть пакета будет отброшена. По этой причине протокол TSIP предусматривает принудительное экранирование всех байтов DLE в информационной части, чтобы последовательность DLE-ETX всегда определялась корректно: после применения экранирования только настоящий конец пакета будет иметь перед байтом ETX нечетное количество байтов DLE, это очевидно: [6]

-------------------

Операции экранирования и снятия экранирования байтов DLE в информационной части пакета выполняются при передаче пакетов по каналу в обе стороны, это показывает приведенная ниже схема:

-----------------------

Снятие экранирования обсуждалось ранее, однако, необходимо дополнительно разобрать алгоритм добавления экранирующих байтов DLE, реализованный в ПО для взаимодействия с GPS-модулем посредством спецоборудования.

Рассматриваемое в данной работе программное обеспечение имеет специальные классы, написанные на С++, для подготовки данных к передаче через канал связи: QByteArrayHelper и TypesConverter. Класс присоединения данных к пакету с произведением автоматической экранировки QByteArrayHelper имеет несколько методов, каждый – для одного из типов данных, которые могут быть присоединены к передаваемому пакету. В основном, это пользовательские данные, считанные из элементов графического интерфейса при получении сигнала об отправке пакета. Пример метода для присоединения и экранирования значения с плавающей точкой двойной точности (тип double):

----------------------------------

Как видно из листинга выше, сначала значение double преобразуется в массив байтов (конечно, содержимое массива эквивалентно байтовому представлению первоначального значения). Это необходимо, так как для выполнения экранирования байты должны быть проверены независимо друг от друга. После выполнения преобразования вызывается перегруженный метод appendAndStuff() для каждого из однобайтовых значений в массиве. Этот метод выглядит следующим образом:

--------------------------

То есть, если байт, который присоединяется в данный момент – это DLE, то необходимо добавить еще один байт DLE, в этом и состоит экранирование.

За преобразование многобайтовых значений в массив байтов и, наоборот, массивов байтов в значения отвечает **блок конвертации байт в числовые типы**, который представлен классом С++ TypesConverter. Его методом toByteArray() как раз и пользуется приведенный выше метод класса QByteArrayHelper. Класс TypesConverter содержит два типа методов: первые создают массив байтов из полученного на вход многобайтового значения (все вариации методов toByteArray()), вторые создают многобайтовое значение и возвращают его, используя полученный на вход массив байтов и начальную позицию в нем (методы bytesTo<имя\_типа\_значения>()) – так сделано для того, чтобы можно было передать целый пакет по ссылке и считать из него для преобразования только те байты, которые составляют требуемое значение. Примеры и тех, и других методов приведены ниже:

------------------------------

В этом классе для выполнения преобразований используются созданные программистом объединения, начинающиеся с нижнего подчеркивания и содержащие имя типа. Секрет объединения (union) в С++ в том, что все его поля ссылаются на одну и ту же область памяти, а в целом объединение имеет размер, равный размеру самого большого их полей [1]. В классе TypesConverter типы значений в объединениях и соответствующие им массивы байт специально подобраны так, чтобы при занесении значения в поле «многобайтовое значение» (.value) его байтовое представление появлялось в поле массива байтов (.bytes). В обратную сторону это работает точно таким же образом. Следующая иллюстрация наглядно показывает работу объединения для типа double в классе TypesConverter:

-------------------------

Методы блока конвертации потока байтов в числовые значения часто используются в подсистеме вывода приходящей информации для преобразования многобайтовых величин, пришедших в пакете от GPS-модуля, в привычные программисту типы данных, так как это необходимо перед выводом полученных величин в логе и/или графическом интерфейсе.

# Подсистема вывода приходящей информации на экран

В разработанном программном обеспечении для взаимодействия с GPS-модулем посредством спецоборудования наряду с подсистемами пользовательского ввода и отправки пакетов и подсистемой информационного обмена с GPS-модулем функционирует подсистема вывода информации, поступающей из канала связи с GPS-модулем. Она тесно связана с блоком чтения данных из порта VirtualCOM. Прочитанные данные подготавливаются для вывода, преобразуются, анализируются и выводятся либо в текстовую область логгирования, либо в соответствующие текстовые поля графического интерфейса главного окна программного обеспечения. Преобразованием, анализом и выводом данных и занимается подсистема вывода.

Данная подсистема практически полностью представлена классом С++ под названием PacketParser, который содержит реализации всех методов разбора приходящих от GPS-модуля пакетов – по одному методу для каждого типа пакета из списка тех, которые описаны в документации TSIP. Также существенную роль здесь играет метод С++-класса COMHandler receiveReport(), задача которого состоит в анализе первых байтов пакета с помощью PacketParser’a после того, как пакет передан объекту вышеназванного класса, и выбора подходящего метода разбора пришедшего пакета; после разбора пакета информацию о нем необходимо вывести в область логгирования, что метод receiveReport() и делает.

Помимо алгоритмической части программного обеспечения на языке С++, к подсистеме вывода относится непосредственно та часть интерфейсного QML-кода, которая предоставляет область логгирования для доступа к ней из кода С++, и все текстовые поля QML, хранящие и отображающие принятую информацию, а именно полезную нагрузку пакетов GPS-модуля и его текущее состояние. Все данные текстовые поля имеют объектное имя, так как проще всего обратиться к ним и изменять их отображаемое содержимое из кода С++, используя объектное имя. Область логгирования заполняется текстовыми сообщениями с помощью системы слотов и сигналов, описанной ранее.

Подсистема вывода приходящей информации имеет большое практическое значение в данном программном обеспечении, поскольку одной из главных задач этого ПО является задача отслеживания состояния GPS-модуля, подключенного на другом конце информационного канала, а также прием полезной информации от этого GPS-модуля.

Небольшой пример, как выглядит подсистема вывода приходящей информации для оператора, управляющего описываемым ПО и для программиста, который ее разрабатывал:

-------------------------------------

# Блок текстового интерфейса вывода

Одна из составляющих подсистемы вывода приходящей информации – это блок текстового интерфейса вывода.

Отображаемая часть данного блока, видимая оператору, использующему ПО для взаимодействия с GPS-модулем, представлена кодом разметки графического интерфейса QML и содержит несколько надписей, текстовых полей, в которых размещаются все данные, необходимые для отслеживания состояния GPS-модуля и прогресса тактической задачи, выполняемой оборудованием, на котором этот модуль установлен (перемещение, скорость оборудования и так далее). Также имеется область логгирования, в которой публикуются сообщения справочного и отладочного характера, такие, как уведомление о приеме пакета, предупреждения о неразобранных пакетах, либо информация от GPS-модуля, недостаточно важная, чтобы постоянно отображать ее в интерфейсе, но, тем не менее, оператор может выразить желание ознакомиться с ней. Интерфейс, спроектированный таким образом, имеет хорошую читаемость, так как элементы, содержащие информацию, не пропадают с дисплея на всём протяжении работы; также они сгруппированы по категориям данных и имеют начальные значения по умолчанию, если какая-то информация не была до сих пор получена.

На программном уровне описанный выше графический интерфейс организован в основном файле QML main.qml, содержащем описание всех элементов главного окна, вложенных в это окно Window. Загрузка видимой части приложения начинается с запуска среды Qt Quick с параметром начального QML-файла “main.qml”. Ниже показан фрагмент данного файла:

--------------------------------

Данная часть графического интерфейса имеет название **блока интерфейса вывода состояния GPS-модуля** и была разобрана ранее.

Графический интерфейс должен каким-то образом получать данные для отображения от GPS-модуля, однако, он не может делать это напрямую по очевидным причинам: интерфейс должен обладать функцией предоставлять оператору графическое представление приложения, а не обрабатывать какие-либо потоки информации. За него приемом, обработкой и передачей занимается вторая часть подсистемы вывода приходящей информации, о чём речь пойдет позже.

Блок текстового интерфейса вывода можно разделить на **блок приема полезной нагрузки с GPS-модуля** и **блок приема информации о текущем состоянии GPS-модуля**.

Блок приема полезной нагрузки публикует в графическом интерфейсе переданные этому блоку (из кода С++, который получил эти значения из порта VirtualCOM от GPS-модуля) различные полезные данные, которые подключенный к ПК GPS-модуль сообщает регулярно: текущая позиция GPS-модуляв пространстве в различных системах измерения, текущая скорость перемещения GPS-модуля в метрах в секунду или градусах широты/долготы; также выводится список SV с их номерами согласно международным обозначениям. Для каждого SV для наглядности публикуется уровень его сигнала относительно GPS-модуля, используется ли этот SV для отслеживания позиции и скорости GPS-модуля и берется ли состояние SV во внимание при расчете вышеуказанной полезной нагрузки GPS-модулем. Иллюстрация ниже показывает, как вышеописанный вывод может выглядеть во время выполнения описываемым программным обеспечением своей работы:

---------------------------

Блок приема информации о текущем состоянии GPS-модуля предоставляет оператору остальную информацию, не менее важную, но не относящуюся к ежесекундно вычисляемой GPS-модулем полезной нагрузке. GPS-модуль регулярно либо единоразово, по команде оператора, присылает эту информацию в порт VirtualCOM в пакетах соответствующего типа, а программный код на С++, разобрав ее, отправляет с помощью обращения по объектному имени к элементам интерфейса для выведения на дисплей оператору. К таким данным можно отнести несколько критически необходимых для отслеживания параметров – дата и время, установленные GPS-модулем после их получения и подтверждения от нескольких SV, температура приемо-передающего устройства, расположенного на плате GPS-модуля, и напряжение на ЦАП, которое должно быть в строго определенных рамках для корректного функционирования модуля. Также в главном окне выводятся текущие настройки форматов для различных параметров ввода-вывода и текущая конфигурация автоматической рассылки пакетов. Дополнительно сделано отображение полной информации о модели и серийных кодах платы GPS-модуля и ее прошивки, а также об установленном на ней программном обеспечении. Некоторые значимые сведения о состоянии GPS-модуля и о характере совершаемых им в данных момент операций также распечатываются на дисплей в графическом интерфейсе:

----------------------------

# Блок разбора приходящих пакетов

Очевидно, что с GPS-модуля по каналу связи между ним и компьютером, на котором запущено ПО для взаимодействия с GPS-модулем посредством спецоборудования, в порт VirtualCOM поступают не понятные для человека значения тех или иных параметров, а поток сигналов-байтов. После получения из этого потока блоком чтения данных из порта VirtualCOM целого пакета информация по-прежнему зашифрована с точки зрения простого обывателя. Необходимо иметь некий модуль, который способен преобразовать множество байтовых значений в читаемые и воспринимаемые человеком числа, слова и иные по типу значения. Именно такую роль в рассматриваемом ПО и выполняет блок разбора приходящих пакетов.

В данном программном обеспечении блок разбора приходящих пакетов полностью представлен классом PacketParser. Обращение к методам данного класса происходит каждый раз при получении нового пакета от GPS-модуля, конкретный метод выбирается по типу полученного пакета (т.е. по его коду и, когда он указан, подкоду). Для этого в методе класса COMHandler receiveReport(), после работы которого пакет преобразуется в запись в области логгирования и полезные величины (и первое, и второе отображаются в графическом интерфейсе, это обязанность блока текстового интерфейса вывода), имеется создание объекта PacketParser и вызов метода начала разбора, который и является инициатором работы с методами класса непосредственного разбора пакетов PackerParser:

---------------------------

Итак, данный класс имеет конструктор объектов, которому на вход поступает массив байтов – то есть, считанный ранее пакет – который необходимо отформатировать перед копированием во внутреннюю переменную data объекта класса PacketParser. Каждый такой объект представляет собой по сути парсер, готовый разобрать переданный ему ранее в конструкторе пакет. Для каждого пакета объект создается отдельно. После создания объекта-парсера достаточно лишь вызвать у него метод analyseAndParse(), а объект сам проанализирует код и подкод и вызовет соответствующий внутренний (private) метод разбора пакета, либо сформирует сообщение о том, что пакет не был разобран по причине того, что он не был распознан как валидный:

-----------------------------------

Остальные методы класса PacketParser – это либо акцессоры для получения кода и подкода пакета (так как получение данных из объекта напрямую противоречит концепции инкапсуляции в объектно-ориентированном программировании), либо методы разбора пакетов, начинающиеся с “parse\_”, после которого идет символическое название пришедшего пакета. Данные названия для удобства программирования и отладки полностью совпадают с названиями макросов #define кодов и подкодов пакетов, определенных в файле kinda\_trimble\_shared.h, и они едины во всём исходном коде данного программного обеспечения:

---------------------------

Данные коды и подкоды пакетов для работы с GPS-модулем Trimble Mini-T Thunderbolt находятся в приведенной документации к протоколу TSIP. [6]

# Блоки конвертации пришедшей информации перед ее выводом. Строение пакетов TSIP

Если рассматривать методы разбора поверхностно, они устроены согласно единому принципу преобразования и распространения данных. Строение каждого пакета известно из документации, и сначала производится его разбор согласно этому строению. Приводить полный перечень строения пакетов, которые могут быть приняты от GPS-модуля Trimble Mini-T Thunderbolt или переданы ему, не имеет смысла, так как он весьма велик. Ниже будет приведен отрывок для ясности того, о чём идет речь; строение остальных пакетов можно уточнить в документации TSIP вышеупомянутого GPS-модуля [6]:

----------------------------- (здесь прилично всего должно быть)

Разобранная информация записывается в строку, которая позже будет передана для логгирования обратно в метод receiveReport(), который создавал объект парсера пакетов. При данном разборе активно используется класс TypesConverter и его статические методы. Они позволяют легко выделять из массива байтов многобайтовые величины и преобразовывать их в представление, в котором они могут храниться и использоваться в коде С++. О сущности этих преобразований было рассказано ранее. Исходный код типичного разбора выглядит так:

---------------------------------------

Далее некоторая информация, которая была выделена заказчиком как важная для демонстрации ее оператору в графическом интерфейсе, передается в определенные поля графического интерфейса. Это означает, что эти текстовые поля обновляются с использованием обращения к ним по объектному имени:

----------------------------------

Иногда перед модификацией текстовых полей промежуточные значения отсылаемых величин сохраняются в переменные, как в листинге выше.

При проектировании методов разбора важным аспектом выступал тот факт, что в пакете хранится информация разного по строению рода, требующая при разборе надлежащего к ней подхода. В пакетах TSIP, приходящих от GPS-модуля, могут встречаться:

- однобайтовые величины – преобразуются обычным приведением типа к числу типа unsigned char (в Qt определенного как quint8):

---------------------------------

- многобайтовые величины – преобразуются в соответствующий тип с помощью методов класса TypesConverter, как было указано выше:

-------------------------------

- битовые поля – это однобайтовые числа, каждый бит которых представляет собой некое значение, несет смысловую нагрузку (например, значение бита «0» - «выключен», «не отслеживается», «настройка 0» и так далее; значение бита «1» - «включен», «отслеживается», «настройка 1» и так далее). Битовые поля разбираются побитно; обращение к биту происходит с использованием маски определенного бита BITx, где х – номер бита в байте:

------------------------

Битовые маски определены в файле kinda\_trimble\_shared.h:

------------------------

- однобайтовые значения, которые являются не просто числом, а каждое их значение из списка определенных несет в себе смысловую нагрузку. Такие байты можно разобрать с помощью оператора switch-case:

--------------------------

После того, как все байты пакета были разобраны, а важные величины отправлены в графический интерфейс, выполняющийся метод разбора возвращает строку с логом пакета обратно в метод receiveReport(), а тот отправляет данные о логгировании графическому интерфейсу.

# Тестирование готового программного продукта

После окончания процесса проектирования и разработки всех подсистем программного обеспечения для взаимодействия с модулем GPS Trimble необходимо провести его тестирование. Учитываются различные аспекты условий тестирования: состояние GPS-модуля, нагрузка на канал информационного обмена, подключение антенны. Также следует проверить ПО без подсоединенного модуля GPS, чтобы проверить работоспособность каждого разработанного блока и их успешное взаимодействие между собой.

Средства отладки Qt Creator предоставляют широчайшие возможности для вывода служебных сообщений и просмотра состояния программы, ее переменных в тот или иной момент выполнения. Самыми популярными и легкими для использования средствами отладки являются класс отладочной печати QDebug и точки останова в совокупности с функцией запуска приложения в режиме отладки.

Класс QDebug предоставляет удобную реализацию вывода пользовательских сообщений, или отладочной печати, в консоль Qt Creator, таким образом, имеется возможность добавить в исходный код отлаживаемого участка программы несколько операторов класса QDebug для вывода значений переменных, побочных значений, просмотра и анализа нити выполнения программного кода и так далее.

Если приложение Qt запущено в режиме отладки, то допускается использование точек останова, заданных в редакторе исходного кода. Приложение при выполнении вынуждено останавливать исполнение инструкций, если встречает точку останова. При этом процесс исполнения приложения в режиме отладки на низком уровне связан с запущенным Qt Creator (то есть Qt Creator в таком случае является программой-отладчиком тестируемого приложения). Именно эта особенность при встрече точки останова позволяет не только приостановить выполнение кода, но и просматривать значения всех переменных исполняемого приложения, просто наведя на них курсор. Точки останова – довольно мощный инструмент отладки, и не стоит им пренебрегать.

Использование вышеописанного инструментария отладки будет показано ниже на конкретных примерах.

# Тестирование графического интерфейса и его связи с программным кодом информационного обмена

Графический интерфейс и программный код информационного обмена постоянно взаимодействуют. Выше были описаны различные способы взаимодействия, такие, как обращение по объектному имени и система слотов и сигналов. Взаимодействие этих двух важных подсистем должно быть безукоризненно налажено.

Тестирование состоит в том, чтобы отправить все возможные пакеты, т.е. проверить работоспособность сигналов всех кнопок интерфейса, а если речь идет о конфигурируемых пакетах – проверить все допустимые значения настраиваемых полей, выпадающих списков и переключателей. При этом допускается использование не настоящего GPS-модуля, подключенного через спецоборудование, а миниатюрного конвертера MicroUSB-COM; подключение к его порту равнозначно подключению к GPS-модулю с точки зрения разработанного ПО. Если замкнуть контакты приема и передачи данных последовательного порта, расположенного на плате конвертера (контакты RX и TX), получим имитацию подключенного устройства, которое, однако, будет отсылать в ответ ту же информацию, которая была передана ей. Но для тестирования взаимодействия подсистемы пользовательского ввода и алгоритмов конфигурирования и отправки пакетов этого более чем достаточно. Также тестируется считывание байтов пакетов из канала связи – порта VirtualCOM – и обнаружение кода пакета и конца пакета, добавление байтов экранирования при отправке и их отбрасывание при приеме.

Схематическое представление стенда для тестирования вышеописанного взаимодействия:

-------------------

Реальное изображение стенда:

-------------------

При использовании точек останова можно следить за побайтным считыванием пакета:

--------------------------

Отладочная печать позволяет узнать, как выглядел сконфигурированный, сформированный и отправленный пакет, а также момент его отправки:

-----------------------

# Тестирование взаимодействия с модулем GPS Trimble на стенде

После того, как тестирование показало, что все алгоритмы отправки, формирования, приема пакетов функционируют в штатном режиме, плату-конвертер MicroUSB-COM можно заменить реальным оборудованием, а именно спецоборудованием, которое подключается по USB к персональному компьютеру оператора; а к спецоборудованию подключен GPS-модуль Trimble Mini-T Thunderbolt с антенной. Тестирование программного продукта с реальным устройством крайне важно, необходимо убедиться, что GPS-модуль понимает приходящие ему пакеты и отвечает именно той информацией, которая была запрошена. Также проверяется работоспособность и правильность функционирования класса разбора пакетов PacketParser и класса преобразования значений TypesConverter. Полученная информация будет отображена в графическом интерфейсе на дисплее оператора, поэтому не составляет труда убедиться в корректности работы алгоритмов.

Схема оборудования для проведения тестирования взаимодействия с модулем GPS:

---------------------------------

Реальный стенд для тестирования выглядит следующим образом:

---------------------------

Оператор под наблюдением программиста-специалиста по отладке моделирует различные ситуации работы, взаимодействия с GPS-модулем, например, конфигурирование настроек ввода-вывода, смена списка присылаемых автоматически пакетов, отслеживание уровней сигнала спутников и так далее. Как можно видеть, GPS-модуль правильно понимает запросы тестируемого программного продукта и отвечает корректной информацией, которая разбирается без проблем:

----------------------------

Тестирование взаимодействия с GPS-модулем прошло успешно.

# Исходные материалы и пособия

1. Qt 4.8. Профессиональное программирование на C++ / М. Шлее. — СПб.: БХВ-Петербург, 2012. — 912 с.
2. QML2 to C++ and back again, with signals and slots – andrew-jones.com [электронный ресурс]. URL: https://andrew-jones.com/blog/qml2-to-c-and-back-again-with-signals-and-slots/, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 08.11.2017 г.
3. Getting Started Qserialport in QT 5.1 (Gui App) – YouTube [электронный ресурс]. URL: https://www.youtube.com/watch?v=UD78xyKbrfk, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 10.11.2017 г. – Системные требования: ПК с установленным интернет-браузером с поддержкой HTML5, стереоколонки или наушники.
4. How Does a USB to Serial Adapter Work? | It Still Works | Giving Old Tech a New Life [электронный ресурс]. URL: https://itstillworks.com/usb-serial-adapter-work-4969162.html, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 20.11.2017 г.
5. rs232 – How does a RS-232 to USB converter work? – Super User [электронный ресурс]. URL: https://superuser.com/questions/1193105/how-does-a-rs-232-to-usb-converter-work, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 20.11.2017 г.
6. Trimble® Mini-T™ GPS Disciplined Clock Module. User Guide. Version 1.00, revision B [электронный ресурс]. – Trimble Navigation Limited, 2007. – 75 c. URL: http://galaxy.agh.edu.pl/~jena/Tele/GPS/MiniT\_UG\_1B.pdf, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 24.11.2017 г.
7. Interacting with QML Objects from C++ | Qt QML 5.10 [электронный ресурс]. URL: http://doc.qt.io/qt-5/qtqml-cppintegration-interactqmlfromcpp.html, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 02.12.2017 г.
8. QSerialPort Class | Qt Serial Port 5.10 [электронный ресурс]. URL: <http://doc.qt.io/qt-5/qserialport.html>, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 19.01.2018.
9. Terminal Example | Qt Serial Port 5.10 [электронный ресурс]. URL: http://doc.qt.io/qt-5/qtserialport-terminal-example.html, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 25.01.2018 г.
10. Blocking Slave Example | Qt Serial Port 5.10 [электронный ресурс]. URL: http://doc.qt.io/qt-5/qtserialport-blockingslave-example.html, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 25.01.2018 г.
11. Styling Qt Quick Controls 2 | Qt Quick Controls 2 5.10 [электронный ресурс]. URL: <http://doc.qt.io/qt-5/qtquickcontrols2-styles.html>, свободный. – Загл. с экрана. – Яз. англ. Дата обращения: 22.02.2018 г.
12. QML – Урок 021. Переключение между окнами в QML [электронный ресурс]. URL: https://evileg.com/ru/post/194/, свободный. – Загл. с экрана. – Яз. рус. Дата обращения: 24.02.2018 г.