Проект RubiRobot

управление роботами на базе контроллера Lego EV3

версия 0.2.6 от 2021-02-07

ОС: **Linux Debian проекта ev3dev**
Язык программирования: **FreePascal**.
Лицензия: **LGPL v3.0**
Сайт: **http://rubirobot.ru**
Группа VK: **https://vk.com/rubirobot**

Оглавление

Введение

Штатное программное обеспечение от компании Lego для контроллера Lego EV3 (в дальнейшем - EV3) поддерживает графический язык программирования, содержит обширную локализованную документацию и множество примеров программ, имеет низкий порог вхождения и может достаточно эффективно использоваться школьниками и студентами всех ступеней обучения. К сожалению, данное ПО не приспособлено для разработки сложных и объемных программ, не поддерживает некоторые алгоритмические конструкции и не функционирует ни на одной из отечественных операционных систем.

Для программирования EV3 существуют сторонние прошивки и/или такие языки программирования как C, C++, Java, Python и некоторые другие. В то-же время, в РФ для обучения программированию в большинстве школ используются разные диалекты языка программирования Pascal (Delphi, PascalABC, FreePascal и т. п.). Достаточно долгое время не существовало средств разработки на языке Pascal для EV3, что затрудняло в программировании робототехнических систем применение школьниками полученных на уроках информатики знаний и умений. Ситуацию частично исправляет проект ev3dev, который позволяет загружать на EV3 операционную систему Linux Debian, в репозитариях которой имеется компилятор FreePascal. Таким образом, появляется возможность создания эффективного инструментария для программирования роботов EV3 с использованием этого языка и первым таким решением является проект RubiRobot.

Проект **RubiRobot** предназначен для разработки программ управления роботами на базе контроллера Lego EV3, с использованием языка программирования FreePascal. Проект быстро развивается, включая в себя все новые аспекты разработки программ управления EV3. Идея создания программной библиотеки для Lego EV3 появилась во время подготовки школьников к робототехническим соревнованиям в конце 2016 - начале 2017 года. Разработка началась в апреле 2017, первые рабочие решения приходятся на конец мая - начало июня 2017 года. В начале декабря 2017 года на сайте http://rubirobot.ru/ была опубликована первая публичная версия библиотеки под лицензией LGPLv3. К июлю 2018 года проект перерос рамки программной библиотеки и объединил в себе четыре параллельно разрабатываемых компонента:

1. Программную библиотеку RubiRobotLib. Библиотека содержит набор классов, объектов и функций для доступа ко всем подсистемам EV3. В состав библиотеки входят более 15 модулей, более 30 демонстрационных примеров разной степени сложности, документация, консольная утилита-менеджер ev3 для автоматизации основных операций по компиляции, загрузки на EV3 и запуска программ. Текущая стабильная версия RubiRobotLib доступна по адресу http://rubirobot.ru/files/src/RubiRobotLib.tar.gz. Текущая разрабатываемая версия ("рабочая сборка") доступна по адресу http://rubirobot.ru/files/src/RubiRobotLib.daily.tar.gz.

2. Графический менеджер RubiRobotManager. Представляет собой графическое приложение для ОС Linux, дублирует большинство функций утилиты ev3, содержит анализатор портов EV3. Является полезным, но необязательным компонентом проекта. В текущей версии проекта доступен в составе компонента RubiRobotCross.

3. Установщики RubiRobotCross и RubiRobotCross_x86-64. RubiRobotCross - это самораспаковывающийся архив, предназначенный для установки на дистрибутив ALT Linux (7-9 платформы). Устанавливает и конфигурирует RubiRobotLib, RubiRobotManager, кроссбиблиотеки ev3dev и кросскомпилятор FreePascal. Является обязательным компонентом проекта при использовании кросскомпиляции программ для EV3 на хостовом компьютере. Текущая стабильная версия RubiRobotCross доступна по адресу http://rubirobot.ru/files/img/setupRubiRobot.tgz. Текущая разрабатываемая версия ("рабочая сборка") доступна по адресу http://rubirobot.ru/files/img/setupRubiRobot.daily.tgz. Для других дистрибутивов Linux, начиная с версии RubiRobot 0.2.6, разработан установщик RubiRobotCross_x86-64, доступный по адресу http://rubirobot.ru/files/img/daily/setupRubiRobot_x86-64.tgz. Его ограничением является возможность функционирования только на аппаратной платформе x86-64 (AMD64). Рабочая сборка доступна по адресу http://rubirobot.ru/files/img/daily/setupRubiRobot_x86-64.daily.tgz

4. Образ ev3dev-RubiRobot. Это модифицированный образ microSD на базе ev3dev, со включенной в состав библиотекой RubiRobotLib. Образ является необходимым и достаточным компонентом для программировании EV3 в ssh-сессии с использованием библиотеки RubiRobotLib. На сегодняшний день поддерживаются образа ev3dev-jessie и ev3dev-stretch, доступные по адресам http://rubirobot.ru/files/img/ev3dev-J-rubirobot.img.bz2 и http://rubirobot.ru/files/img/ev3dev-S-rubirobot.img.xz. Существуют также версии для робототехнических соревнований, характеризуемые сокращенным временем загрузки операционной системы и отсутствием автозагрузки некоторых сервисов, доступные по адресам http://rubirobot.ru/files/img/ev3dev-J-rubirobot-sport.img.xz и http://rubirobot.ru/files/img/ev3dev-S-rubirobot-mini.img.xz.

Адрес постоянного расположения RubiRobot в сети Интернет - сайт http://rubirobot.ru/, зеркала - http://firstbyte.rubirobot.ru/, http://shgpi.rubirobot.ru/. Автор проекта RubiRobot - Слинкин Д.А. (xdsl@list.ru).

Разработка проекта RubiRobot

Поставленные задачи идентифицируются датой. Решенные задачи помечаются знаком "*", датой решения задачи и версией библиотеки, в которой задача была решена.

Дорожная карта по развитию инфраструктуры проекта RubiRobot

2018-06. Подготовить документацию по подготовке SD-карты и подключению к EV3 со спецификой для седьмой и восьмой платформ AltLinux.
*2018-06. При использовании установщика предусмотреть размещение ссылок на библиотеку на рабочем столе и в главном меню (решено 2018-06. v0.2.3).
*2017-12. На основе документации ev3dev разработать русскоязычную документацию по подготовке SD-карты, инсталляции, подключению, обновлению ПО, доустановке требуемых пакетов, русификации. (решено 2018-01. v0.2)
*2017-12. Подготовить документацию по разработке программ на EV3 в SSH-сессии. (решено 2018-01. v0.2)
*2017-12. Подготовить документацию по кросскомпиляции. (решено 2018-06. v0.2.2)
*2017-12. Готовить и периодически публиковать образ SD-карты с необходимым ПО для использования библиотеки rubirobot и архив библиотек для кросскомпиляции. (решено 2018-06. v0.2.2)

Дорожная карта по развитию кода компонентов RubiRobot

2020-02. Выполнить рефакторинг кода в целях избавления от типа данных Variant.
2019-04. В связи с выходом релиза ev3dev-stretch, подготовить модифицированный образ ev3dev-stretch с необходимым программным обеспечением и актуальной версией библиотеки RubiRobotLib. Подготовить очередную версию библиотеки, в полном объеме учитывающую возможности ev3dev-stretch.
*2018-09. Обеспечить поддержку нескольких шрифтов. (решено 2018-10. v0.2.5)
*2018-09. Обеспечить возможность параллельного исполнения кода. (решено 2018-10. v0.2.5)
*2018-08. Обеспечить работоспособность библиотеки на дистрибутиве ev3dev-stretch. (решено 2018-08. v0.2.4)
*2018-08. Обеспечить поддержку блока питания. (решено 2018-09. v0.2.4)
*2018-07. Обеспечить возможность параллельного исполнения кода библиотеки, обеспечить потокобезопасность кода. (решено 2018-10. v0.2.5)
*2018-06. Разработать графическое приложение для хостового компьютера на базе ОС Linux, с функциями управления EV3 и анализатора портов. (решено 2018-06. v0.2.3)
*2018-06. Ввести возможность определения уже запущенного приложения на базе библиотеки RubiRobot и обеспечить различные реакции на данный факт по выбору программиста: игнорирование, завершение текущего приложения, завершение ранее запущенного приложения и т.д. (решено 2018-06. v0.2.3)
*2018-02. Разработать механизм программного рестарта устройств для исправления ситуации неправильного определения типа устройства или отсутствия инициализации устройства при его физическом наличии. Проблема часто возникает сразу после старта системы и критична для робототехнических соревнований. (решено 2018-04. v0.2.1)

Модуль rubiroSound:
*2019-01. Возможность подачи звуковых сигналов напрямую, без вызова внешних программ. (решено 2019-02. v0.2.5.1)
2017-12. Возможность генерации аудио-файлов
2017-12. Создание очереди звуковых запросов
2017-12. Поддержка мультиязычности

Модуль rubiroScreen:
2017-12. Уменьшение зависимости от fcl-image при сохранении неизменной интерфейсной части.
*2017-12. Поддержка нескольких шрифтов, возможность указывать их местоположение в файловой системе (решено 2018-10. v0.2.5)

Модули rubiroMotors и rubiroSensors:
2018-11. Переработка рулевого управления в целях синхронизации вращения моторов.
*2018-02. Определение и установка позиции мотора в градусах и в оборотах (решено 2018-06. v0.2.1)
2018-01. Создание простого механизма отключения и включения бесперебойного режима, с автоматической поддержкой всех способов доступа к устройствам.
*2017-12. Корректировка формулы поворота и способа остановки в TEv3Rule для максимального соответствия программному обеспечению Lego. (решено 2018-01. v0.2)
*2017-12. Поддержка рулевого управления для средних моторов. (решено 2018-01. v0.2)
*2017-12. Реализация режима ожидания (waitMode) для инфракрасного датчика. (решено 2018-01. v0.2)
*2017-12. Определение свойств-переменных для моторов и датчиков, их создание в момент первого обращения, автоматическое уничтожение при отключении и по окончании программы. Цель: упростить обращение к датчикам-моторам, без объявления переменных и явного создания объектов. Например - gyro и gyro1 - первый подключенный гироскоп, gyro2 - второй и т.д.; gyro[1] - гироскоп на первом порту, gyro[2] - гироскоп на втором порту и т. д.; rule и rule1 - рулевое управление на первых двух моторах, rule2 - на последующих двух моторах, rule[1,4] - рулевое управление на первом и четвертом моторах и т. д. (решено 2018-01. v0.2, функциональный способ доступа к устройствам)

Обнаруженные ошибки

Консольный менеджер-утилита ev3
#7, 2018-12. При наличии в беспроводной сети нескольких роботов, существует возможность подключения только к одному из них. Попытка подключения к другому роботу указанием IP-адреса (ev3 setip …) завершается сбросом адреса и возвратом к предыдущему подключению.
RubiRobotManager
*#10, 2019-05. Долгое нажатие на кнопку BACK робота завершало не только работу текущей программы, но и анализатора портов менеджера. Повторный запуска анализатора был невозможен без перезагрузки робота (решено 2019-05-29)
*#4, 2018-06. Ошибка в версии 0.1 от 2018-06-28 не позволяла повторно запускать анализатор портов, если его работа была аварийно прервана (решено 2018-07-02)
Модуль uev3
*#5, 2018-06. В версии 0.2.1 стало невозможным собирать проекты непосредственно на EV3 из-за ошибки при компиляции модуля uev3. Причина: для озвучки аварийного завершения

программы использовались типы данных, которые присутствуют во FreePascal 3.0.0, но отсутствуют во FreePascal 2.6.4 (решено 2018-06, v0.2.2)

Модуль uev3func

*#8, 2019-02. FreePascal 3.3.1 не может скомпилировать данный модуль по причине изменения механизмов обработки вариантных типов данных (решено 2019-03, v0.2.5.1)

Модули uev3Devices, uev3sysfs

*#2, 2017-12. Уничтожение отключившихся устройств в uev3Devices.TEv3DeviceFactory.Refresh входит в противоречие с режимом ожидания повторного подключения в uev3sysfs.TEv3Data. При отказе от уничтожения становится невозможным подключение устройства другого типа к использованному ранее порту. Цугцванг. (решено 2018-01, v0.2, бесперебойный режим)

Модуль rubiroScreen

#3, 2017-12. Стили кисти функционируют некорректно при переходе на freepascal версии 3.0.2

Модуль rubiroMotors

*#11, 2019-11. Запуск мотора на низкой скорости может в течении нескольких десятков миллисекунд генерировать для него состояние блокировки. Таким образом, последующий запуск метода wait мотора или рулевого управления в режиме ожидания искусственной остановки немедленно возвращает управление, еще до начала движения. (решено 2019-12, v0.2.6)

*#9, 2019-02. Запуск мотора, находящегося в состоянии блокировки, выводит его из данного состояния не мгновенно, а в течении 50-100 мс. Таким образом, последующий запуск метода wait мотора или рулевого управления в режиме ожидания искусственной остановки немедленно возвращает управление, еще до начала движения. (решено 2019-03, v0.2.5.1)

*#6, 2018-11. Счетчики оборотов могли работали некорректно при использовании реверса, который, в свою очередь, применялся при отрицательных скоростях для вращения на определенное количество оборотов или градусов. В версии 0.2.5 ошибка исправлена, так как реверс более не используется в организации вращения моторов. (решено 2018-12, v0.2.5)

*#1, 2017-12. Для всех видов метода RunTime моторов и рулевого управления не работает реверс. При задании отрицательной скорости направление вращения не изменяется. (решено 2017-12, v0.1)

Новое в версиях проекта

Версия 0.2.6

2021-02. Полная поддержка возможностей ev3dev-stretch. В RubiRobotManager для ev3dev-stretch дублируется содержимое дисплея EV3, уменьшена зависимость от доступности трея (для поддержки Debian и Ubuntu)

2021-02. Модифицирован консольный менеджер ev3 в целях поддержки различных дистрибутивов Linux.

2021-01. Поддержка XRGB формата экрана модулем rubiroScreen в ev3dev-stretch. Изображение формируется и хранится в формате RGB, отображается четырьмя оттенками серого на экране блока. Для ev3dev-jessie сохранена поддержка черно-белого изображения.

2020-12. В прошивке EV3 удален запуск ntp-клиента во время старта системы. Вместо этого при первом подключении к EV3 на нем устанавливаются текущие дата и время, полученные с хостового компьютера.

2020-04. Разработан модуль полезных алгоритмов rubiroAlgo.

2020-02. Реализован альтернативный способ подключения модулей библиотеки.

2019-12. В модуле rubiroMotors введена целочисленная переменная motorArtificalWait со значением по умолчанию 200. Означает время в миллисекундах и применяется во время ожидания искусственной остановки мотора. Если мотор был заблокирован на motorArtificalWait миллисекунд, то он считается остановленным (#11).

2019-09. Расширен набор команд консольного менеджера ev3.

2019-05. Исправлена ошибка аварийного завершения работы анализатора портов на EV3 при использовании RubiRobotManager (#10).

2019-04. Обеспечена поддержка датчика света NXT Light Sensor 9844.

2019-04. Обеспечена компенсация некоторых типов погрешностей гироскопического датчика.

Версия 0.2.5.1

2019-03. Запуск мотора предварительно выводит его из состояния блокировки, если это необходимо (#9).

2019-02. Метод TEv3Sound.beepDirect позволяет подавать тоновые сигналы напрямую на звуковое устройство без вызова внешней программы.

2019-02. Исправлена ошибка, не позволяющая компилировать библиотеку во FreePascal 3.3.1 (#8).

Версия 0.2.5

2018-12. Оптимизированы классы моторов и рулевого управления, скорость выполнения команд возросла на ~50%. Попутно исправлена ошибка #6.

2018-11. Проведена реструктуризация библиотеки, все базовые типы данных, переменные и константы перенесены из вспомогательных модулей в модуль uev3. Для конечного пользователя теперь рекомендуется подключать только модуль uev3 и необходимые rubiro-модули.

2018-10. Обеспечена возможность параллельного исполнения пользовательских процедур, реализована потокобезопасность всех объектов библиотеки RubiRobotLib.

2018-10. Введена поддержка мьютексов. Поддерживаются до 6 мьютексов на каждый объект библиотеки RubiRobotLib.

2018-10. Введена поддержка дополнительных шрифтов, поддерживаются англоязычные и русскоязычные шрифты 3-х типов (mono, sans, serif). Имеется возможность включать шрифты в код исполняемого файла для снижения зависимости от наличия шрифтов в целевой системе.

2018-10. Обеспечена возможность ожидания естественной остановки моторов, с игнорированием их блокировки (искусственной остановки).

Версия 0.2.4

2018-09. Обеспечен доступ к блоку питания: модуль rubiroBattery, класс TEv3Battery, объект ev3Battery.

2018-08. Обеспечена возможность работы библиотеки RubiRobotLib на дистрибутиве ev3dev-stretch. Сохранена обратная совместимость с ev3dev-jessie.

2018-08. Стало возможным отключать цветовую индикацию при старте программы (см. переменную LedsOnStartup модуля uev3).

2018-08. Обеспечена возможность запуска моторов прямой подачей напряжения с источника питания.

2018-09. Для установки скорости моторов теперь используется вещественный тип данных вместо целочисленного.

Версия 0.2.3

2018-06. При установке на рабочем столе и в главном меню размещаются ярлыки графического менеджера RubiRobotManager. Также на рабочем столе размещается ссылка на основной каталог библиотеки.

2018-06. Разработан менеджер RubiRobotManager, версия 0.1 - графическое приложение для хостового компьютера на базе ОС Linux, с функциями управления EV3 и анализатором портов.

2018-06. Введены три режима старта RR-программы, с возможностью завершать уже запущенные RR-программы или обмениваться с ними сигналами.

2018-06. Модуль поддержки дисплея rubiroScreen теперь не подключается автоматически. Это позволяет значительно (до 2-х раз) сократить размер исполняемого файла программ, которые не используют дисплей EV3.

Версия 0.2.2

2018-06. Разработаны: консольный менеджер для удаленного управления EV3, установщик библиотеки RubiRobot и кросскомпилятора, подготовлена документация по кросскомпиляции.

2018-06. Добавлены до 10 счетчиков вращений/оборотов/градусов на мотор, позволяющих обеспечивать ожидание достижения мотора определенной позиции.

2018-06. Добавлены определение и установка позиции мотора в градусах и в оборотах.

2018-06. Исправлена ошибка #5, не позволявшая собирать проекты непосредственно на EV3.

Версия 0.2.1

2018-05. Проведена озвучка старта и аварийного завершения программы.

2018-04. Решена задача программной реинициализации датчиков и моторов, которые не были подключены или неверно определены операционной системой при старте EV3.

2018-04. Обеспечено автоматическое завершение работы программы при длительном нажатии на кнопку BACK EV3.

2018-04. Для моторов добавлены методы движения до абсолютной позиции.

2018-03. Добавлена поддержка устройства NXT-кнопка.

Версия 0.2

2018-01. Управление реверсом в классах TEv3TachoMotror и TEv3Rule было перенесено из public в protected. Причина: реверс используется в отдельных методах, связанных с запуском двигателей на отрицательной скорости, поэтому пользовательское изменение реверса может привести к неадекватному поведению двигателя.

2018-01. Введен механизм управления таймерами в классе TEv3. Данный класс является базовым, что позволяет обеспечить поддержку до 10 таймеров каждому объекту системы. Произведен рефакторинг библиотеки с заменой всех действий, связанных в временными задержками, на таймеры TEv3.

2018-01. Решена проблема неоднозначности в работе uev3Devices.TEv3DeviceFactory.Refresh. Теперь в метод Refresh передается логический параметр freeLostDevices, по умолчанию равный false. При этом сохраняется возможность продолжения работы программы при переподключении физических устройств в случае их форсмажорного кратковременного отключения (механизм получил название "Бесперебойный режим работы"). Недостатком является невозможность во время работы программы корректного переподключения к одному и тому-же порту другого типа устройства.
При передаче freeLostDevices=true появляется возможность произвольным образом переподключать устройства, однако все объекты, связанные с отключенными устройствами, должны быть в обязательно порядке пересозданы. Применяется в ситуациях, когда

предполагаются многочисленные подключения-отключения различных физических устройств, например, в приложении portview из примеров библиотеки.

2018-01. Решена задача доступа к объектам датчиков, моторов и рулевому управлению без объявления соответствующих переменных, с автоматическим созданием требуемых объектов и уничтожением их в конце работы программы. Доступ реализован в виде вызова соответствующих функций. Например, для цветовых датчиков определены функции ev3Color, ev3Color1, ev3Color2, ev3Color3, ev3Color4. Механизм получил название "Функциональный способ доступа к устройствам"

2018-01. Рулевое управление функционирует как на больших, так и на средних моторах. Для указания типа мотора следует передать в конструктор соответствующий класс. По умолчанию, без передачи класса в конструктор, рулевое управление функционирует на больших моторах.

Подготовка к программированию EV3

Программирование контроллера EV3 с использованием библиотеки RubiRobotLib подразумевает разработку программ на языке FreePascal с подключением соответствующих модулей библиотеки, компиляцией и запуском полученных исполняемых непосредственно на EV3. Чтобы отличить такие программы от любых других, будем в дальнейшем обозначать их термином **"RR-программы"**.

Загрузка EV3 с Linux Debian проекта ev3dev, образ ev3dev-RubiRobot

Для создания на EV3 RR-программ потребуется microSD-карта с дистрибутивом Debian Linux проекта ev3dev (http://www.ev3dev.org/) и загрузка EV3 с подготовленной карты. Проект ev3dev предлагает два образа Debian Linux для загрузки: стабильный ev3dev-jessie (Debian 8) и более новый ev3dev-stretch (Debian 9). Начиная с версии 0.2.4 проект RubiRobot поддерживает не только ev3dev-jessie, но и ev3dev-stretch. Для ускорения подготовки к программированию, могут быть использованы модифицированные образа microSD на базе ev3dev-jessie и ev3dev-stretch, с обновленным программным обеспечением, русификацией, установленным FreePascal и дополнительным ПО, настроенной и подготовленной к эксплуатации библиотекой RubiRobotLib. Образа доступны для загрузки по адресам http://rubirobot.ru/files/img/ev3dev-J-rubirobot.img.bz2 и http://rubirobot.ru/files/img/ev3dev-S-rubirobot.img.xz. Процесс их загрузки на microSD ничем не отличается от описанного в документации для оригинального образа. Также, специально для робототехнических соревнований подготовлены оптимизированные образа, доступные по адресам http://rubirobot.ru/files/img/ev3dev-J-rubirobot-sport.img.xz и http://rubirobot.ru/files/img/ev3dev-S-rubirobot-mini.img.xz. В образах отключены некоторые сервисы, ускорена загрузка системы, в домашнем каталоге пользователя визуально доступны только программы portview (анализатор портов) и rrKill (чистильщик), все остальное, в том числе библиотека RubiRobotLib, находится в скрытых каталогах.

Существует подробная документация по работе с образом на английском языке на сайте проекта ev3dev по адресам http://www.ev3dev.org/docs/getting-started/ и http://www.ev3dev.org/docs/tutorials/. На русском языке по адресу http://www.proghouse.ru/article-box/107-ev3dev доступна пошаговая инструкция подготовки SD-карты и подключения к EV3 из операционной системы Windows 7. Если по каким-то причинам сайт www.proghouse.ru недоступен, копия статьи находится в наборе документации проекта в файле ev3dev.install.pdf.

Если используется модифицированный образ microSD проекта RubiRobot, то дальнейшие действия в текущем параграфе можно пропустить. Если используется оригинальный образ microSD проекта ev3dev, то после выполнения всех подготовительных работ рекомендуется провести обновление ПО, русификацию и доустановку некоторых пакетов. Для этого, подключившись в ssh-сессии к EV3, следует выполнить следующий набор команд:

1. Обновление списка репозитариев
```
sudo apt-get update
```

2. Обновление установленного ПО. Процесс может оказаться достаточно длительным (несколько часов), поэтому следует обеспечить на этот период достаточную зарядку аккумулятора, либо подключить EV3 к электросети.

```
sudo apt-get upgrade
```

Не рекомендуется выполнять полное обновление дистрибутива с помощью sudo apt-get dist-upgrade. При таком обновлении существует вероятность получить систему с недостаточно хорошо протестированной функциональностью. Например, лично автор попал в ситуацию (https://github.com/ev3dev/ev3dev/issues/1249), когда после полного обновления системы перестал поддерживаться определенный тип датчиков.

3. Установка поддержки национальных языков, файлового менеджера Midnight Commander и набора шрифтов dejavu, один из которых используется библиотекой Rubirobot для вывода текста на дисплей EV3.

```
sudo apt-get install locales-all mc
```

4. Создание в домашнем каталоге пользователя подкаталога bin. Размещение в этом каталоге исполняемых файлов позволит запускать их в дальнейшем без указания пути, как обычные команды консоли.

```
mkdir ~/bin
```

5. Обеспечение поддержки русского языка. После запуска команды откроется псеводографическое окно, где следует выбрать ru_RU.UTF-8 локалью по умолчанию. После этого рекомендуется перезагрузить EV3, чтобы все изменения вступили в силу.

```
sudo dpkg-reconfigure locales
sudo reboot
```

6. Если компиляция RR-программ предполагается на блоке, то следует также установить пакеты fpc и gcc. Установка длительная (несколько часов), т. к. FreePascal требует большого количества дополнительного ПО. Поэтому следует обеспечить на этот период достаточную зарядку аккумулятора, либо подключить EV3 к электросети.

```
sudo apt-get install fpc gcc
```

Существует возможность запускать RR-программы на оригинальном образе microSD проекта ev3dev без установки какого-либо дополнительного программного обеспечения, с учетом некоторых ограничений:
1. Не использовать в RR-программах вывод текста на графический монитор, по причине отсутствия необходимых шрифтов.
2. Не использовать в RR-программах русскоязычный ввод-вывод ни в каком виде, по причине отсутствия русской локализации.

Режимы программирования EV3

Разработка RR-программ возможна в двух режимах:

1) Разработка и компиляция программы **непосредственно на блоке EV3 в SSH-сессии**. Данный вариант достаточно просто реализуется, но обладает рядом недостатков, одним из которых является невысокая скорость компиляции программы (от 5 секунд и выше) и невозможность использования на блоке графических средств разработки (Geany, Lazarus и т.п). Немного ускорить процесс разработки поможет подготовка текста программы на

хостовом компьютере (в дальнейшем будем так называть персональный компьютер, к которому подключен контроллер EV3), с последующим переносом и компиляцией ее на EV3.

2) Разработка и компиляция программы на хостовом компьютере **средствами кросскомпиляции** с последующим копированием полученного исполняемого файла на EV3. Данный вариант реализуется гораздо сложнее первого, однако после первичной настройки позволяет во много раз увеличить скорость компиляции программы.

Следует отметить, что скорость разработки **НЕ ВЛИЯЕТ** на скорость выполнения программы, поэтому исполняемый файл будет одинаково хорошо работать на блоке вне зависимости от способа его получения.

Простейшая программа для EV3

Текст программы для EV3 начинается с указания необходимых директив компилятора и подключения набора используемых модулей. Тело программы начинается с вызова функции инициализации библиотеки ev3Init.

В простейшем случае программа может выглядеть так:

```
{$mode objfpc}    // переход в режим поддержки синтаксиса Object Pascal
{$H+}             // поддержка «длинных» строк
uses  cthreads,   // поддержка многозадачности
      uev3;       // базовый модуль библиотеки RubiRobotLib
begin
 ev3init();
 …
end.
```

С подключением всего набора пользовательских модулей библиотеки RubiRobotLib программа будет выглядеть следующим образом:

```
{$mode objfpc}          // переход в режим поддержки синтаксиса Object Pascal
{$H+}                   // поддержка «длинных» строк
uses  cthreads,         // поддержка многозадачности
      uev3;             // базовый модуль библиотеки RubiRobotLib
     ,rubiroSound       // поддержка аудио
     ,rubiroButtons     // поддержка кнопок блока
     ,rubiroLeds        // поддержка цветоиндикации блока
     ,rubiroScreen      // поддержка дисплея блока
     ,rubiroMotors      // поддержка моторов и рулевого управления
     ,rubiroSensors     // поддержка датчиков
     ,rubiroBattery     // поддержка блока питания
     ,rubiroParallel;   // поддержка параллельного исполнения
begin
 ev3init();
 …
end.
```

Для уменьшения громоздкости начального кода программы, начиная с версии 0.2.6 библиотеки необходимые директивы компилятора хранятся в файле **rubiro.inc**, а для списка модулей определено два макроса — **rubiroBase**, содержащий минимально необходимый

набор модулей, и **rubiroAll**, содержащий весь набор пользовательских модулей. Таким образом, минимально необходимый начальный код сокращается:

```
{$i rubiro.inc}
uses rubiroBase;
begin
 ev3init();
 …
end.
```

Программа с поддержкой всех возможностей библиотеки будет выглядеть так:

```
{$i rubiro.inc}
uses rubiroAll;
begin
 ev3init();
 …
end.
```

Любой промежуточный вариант подразумевает подключение rubiroBase и части пользовательских модулей. Ниже приведена программа, где в дополнение к базовому набору модулей подключается модуль поддержки цветоиндикации:

```
// "переливание" цветов (led0.pp)
{$i rubiro.inc}
uses rubiroBase, rubiroLeds;
 var i:byte;
begin
 ev3Init();
 for i in byte do ev3Leds.all(255-i,i,i,255-i);
 for i:=0 to 255 do ev3Leds.all(i,255-i,255-i,i);
end.
```

Подготовка к программированию на EV3 в SSH-сессии

Программировать робота с помощью библиотеки можно непосредственно на EV3, в ssh-сессии, с использованием любого текстового редактора (например - редактора файлового менеджера Midnight Commander, в дальнейшем - mc) и компилятора freepascal. Таким образом, на хостовом компьютере из дополнительного программного обеспечения потребуется установить только ssh-клиент. В любом дистрибутиве ОС Linux ssh-клиент установлен по умолчанию, для ОС Windows рекомендуется использовать putty (http://www.putty.org/). Недостатком такого подхода является **невысокая скорость компиляции**, от 5 секунд для простейших программ, до 1 минуты для RR-программ с использованием всех возможностей библиотеки.

Если используется модифицированный образ microSD проекта RubiRobot и актуальная версия библиотеки, то последующие операции по установке и настройке библиотеки можно пропустить. Предполагается, что дальнейшие действия будут производится в ssh-сессии на EV3, которая была открыта с помощью putty в ОС Windows или штатного ssh-клиента в ОС Linux:

1. Для выполнения файловых операций и подготовки текста программы рекомендуется использовать файловый менеджер Midnight Commander (запуск - команда mc). Для компиляции программы и выполнения некоторых других операций следует применять

штатную консоль Linux. При запущенном mc временный выход в консоль реализуется комбинацией клавиш "Ctrl-o". Повторное нажатие той-же комбинации вернет в Midnight Commander.
Файловый менеджер Mindnight Commander поддерживает встроенный и один из внешних текстовых редакторов. Рекомендуется выбрать встроенный текстовый редактор (если он не выбран по умолчанию). Для этого можно использовать меню mc: "Настройки/Конфигурация/Прочие настройки/Встроенный редактор". Если по каким-то причинам встроенный редактор не устраивает, то следует отключить указанный элемент конфигурации и выбрать внешний редактор командой select-editor.

2. Для начала работы с библиотекой следует перейти в домашний каталог (например, командой cd) и загрузить ее последнюю версию с сайта http://rubirobot.ru/, например - следующей командой:

```
wget http://rubirobot.ru/files/src/RubiRobotLib.tar.gz
```

Разархивировать полученный файл можно командой:

```
tar -xf RubiRobotLib.tar.gz
```

В результате в домашнем каталоге пользователя будет создан подкаталог **RubiRobotLib** со всеми компонентами библиотеки. В текущей версии библиотеки это каталог **units** с исходными файлами библиотеки, каталог **examples** с примерами, каталог **docs** с документацией, каталог **linux/ev3** с полезными файлами конфигурации и скриптами, которые рекомендуется скопировать в домашний каталог робота и каталог **linux/host** аналогичной структуры, который предназначен для хостового компьютера на базе ОС Linux.

3. Далее следует создать пользовательский файл конфигурации FreePascal в домашнем каталоге пользователя с указанием подключения основного конфигурационного файла и модулей библиотеки RubiRobot, использованием оптимизационной компоновки для уменьшения размера исполняемого файла. Указанный файл называется **.fpc.cfg** и присутствует в каталоге linux/ev3/home/robot/ библиотеки. Рекомендуется его скопировать в домашний каталог пользователя.

Система подготовлена к разработке ПО. Для проверки ее работоспособности рекомендуется перейти в каталог примеров и скомпилировать один из них, например, анализатор портов EV3 portview:

```
cd ~/RubiRobotLib/examples
fpc portview.pp
```

Первая компиляция будет достаточно длительной (примерно 90 секунд), так как требует перекомпиляции всей библиотеки. Продолжительность последующих компиляций будет в разы меньше. Полученный исполняемый файл portview следует запускать с блока EV3 через элемент меню "File Browser" программы управления блоком brickman (Brick Manager).

Для собственных проектов рекомендуется создать отдельный подкаталог в домашнем каталоге пользователя. Полученные исполняемые файлы можно запускать как непосредственно с блока EV3 через brickman (элемент меню "File Browser"), так и в ssh-сессии.

При запуске программ с блока через элемент меню "File Browser" следует помнить, что используемые текстовые шрифты для дисплея блока не поддерживают русский язык, поэтому для вывода с помощью процедур write и writeln рекомендуется использовать латиницу. Поддержка русского языка на дисплее EV3 обеспечивается объектом ev3screen библиотеки.
При необходимости досрочно завершить выполнение программы следует несколько секунд жать на кнопку BACK.

При запуске программ в ssh-сессии русский язык консоли полностью поддерживается, однако следует помнить, что нажатие кнопок на блоке не только передается в программу, но и задействует соответствующие функции в программе управления блоком. Например, если запустить в ssh-сессии RR-программу, которая потребует от пользователя последовательного нажатия на EV3 кнопок BACK и CENTER, то это скорее всего приведет к выключению блока. Дело в том, что нажатие кнопки BACK, при нахождении в главном меню brickman, приводит к выводу меню перезагрузки, а последующее нажатие кнопки CENTER задействует элемент меню "Power Off". Поэтому следует очень аккуратно взаимодействовать в ssh-сессии с программами, которые обрабатывают кнопки блока, не допуская подобных коллизий.
При необходимости досрочно завершить выполнение программы следует использовать комбинацию клавиш Ctrl-C.

В целях упрощения запуска программ с блока, рекомендуется копировать скомпилированные исполняемые файлы программ непосредственно в домашний каталог. Для этого служит скрипт fpccp, который находится в каталоге linux/ev3/home/robot/bin/ библиотеки. Рекомендуется скопировать его в каталог bin домашнего каталога робота. Скрипт удаляет целевой исполняемый файл, затем компилирует переданный файл программы и при отсутствии ошибок - копирует полученный исполняемый файл в домашний каталог. Таким образом, при возникновении ошибок компиляции, исполняемый файл будет гарантированно отсутствовать в домашнем каталоге пользователя. Соответственно, при отсутствии ошибок компиляции, исполняемый файл будет гарантированно присутствовать в домашнем каталоге пользователя.

Подготовка к программированию с использованием средств кросскомпиляции, установщик RubiRobotCross.

Если на хостовом компьютере используется ОС Linux, рекомендуется применять кросскомпиляцию при программировании EV3. Это резко повысит скорость разработки и снизит нагрузку на EV3. При таком подходе наличие библиотеки RubiRobotLib на EV3 является необязательным, достаточно иметь данную библиотеку и кросскомпилятор на хостовом компьютере, а на EV3 будут храниться и запускаться только исполняемые файлы.

Начиная с версии 0.2.2 библиотеку RubiRobotLib совместно с кросскомпилятором и сопутствующими утилитами можно установить на хостовый компьютер в автоматизированном режиме. Данный пакет программ называется RubiRobotCross и доступен в виде самораспаковывающегося архива. Установщик RubiRobotCross предназначен для выполнения на операционной системе ALT Linux (платформы 7-9). Конкретный дистрибутив платформы значения не имеет, установщик предложит загрузить все необходимые пакеты. Основная причина выбора ALT Linux: автор использует его в своей повседневной работе, а также - дистрибутив содержит полный набор исходных кодов FreePascal, что позволяет создать кросскомпилятор во время установки RubiRobotCross. Для установки на других

дистрибутивах Linux (Debian, Ubuntu и т. п.) можно воспользоваться бинарной сборкой RubiRobotCross для платформы x86-64 (AMD64, x64 и т.п.), которая включает в себя подготовленный кросскомпилятор FreePascal, ассемблер и компоновщик для соответствующей платформы. Версия RubiRobotCross_x86-64 может быть установлена и на ALT Linux, при условии выбора платформы x86_64.

В дальнейшем планируется обеспечить поддержку других дистрибутивов ОС Linux.

Актуальная версия RubiRobotCross находится по адресу http://rubirobot.ru/files/img/setupRubiRobot.tgz. Данный архив достаточно загрузить и распаковать любым архиватором. Единственный исполняемый файл в архиве носит имя setupRubiRobot.версия-датасборки.run. Например, для версии 0.2.2, собранной 11 июня 2018 года, исполняемый файл установщика будет называться **setupRubiRobot.0.2.2-20180611.run**. Установка производится в терминале, однако запуск установщика возможен как консольными, так и графическими средствами. В последнем случае установщик самостоятельно запустит графический терминал и продолжит в нем работу.

Для полной установки потребуется не менее 700 мегабайт во временном каталоге и порядка 350 мегабайт в целевом местоположении. Практически весь объем складывается из кроссбиблиотек EV3 и кросскомпилятора FreePascal. Кроссбиблиотеки включают в себя более 2000 системных файлов, скопированных с EV3. Такой набор кроссбиблиотек избыточен для создания подавляющего большинства проектов и на текущий день может быть уменьшен до нескольких десятков файлов, что и было сделано, начиная с версии 0.2.6 проекта.

Установка производится **под правами обычного пользователя** и не изменяет системные файлы. Исключение составляет установка несколько пакетов из штатного репозитария. Настройка конфигурационных файлов производится для текущего пользователя, другие пользователи доступа к библиотеке, утилитам и кросскомпилятору иметь не будут.

Установка состоит из нескольких опциональных этапов. Это позволяет в дальнейшем повторить установку, пропустив один или несколько из них.

Этапы установки:
1. Установка пакетов xterm, xdg-utils, sshpass, binutils-arm-linux-gnu и lazarus
2. Ввод имени каталога для размещения результата установки.
3. Установка консольного и графического менеджеров проекта RubiRobot.
4. Настройка доступа к EV3 по ssh.
5. Организация беспарольного доступа к EV3.
6. Создание кросскомпилятора.
7. Копирование библиотеки RubiRobotLib в целевое местоположение.
8. Компиляция библиотеки RubiRobotLib и всех примеров.

Результат установки:

Пакет RubiRobotCross при установке размещает свое содержимое в отдельном подкаталоге домашнего каталога пользователя (исключения - см. ниже). По умолчанию подкаталог носит имя пакета (RubiRobotCross) и обозначается как "основной каталог" пакета.

Пакет RubiRobotCross содержит следующий набор файлов в основном каталоге:
1. Краткая справка: README
2. Версии программных продуктов пакета: VERSION
3. Текст лицензионного соглашения на русском языке: COPYING.LESSER
4. Текст лицензионного соглашения на английском языке: COPYING.LESSER.EN
5. Документация: rubiro.manual.archive.pdf

Пакет RubiRobotCross содержит следующий набор программных продуктов в основном каталоге:
1. Программная библиотека RubiRobot, каталог RubiRobotLib/
2. Графический менеджер библиотеки RubiRobot, каталог RubiRobotManager/
3. Кроссбиблиотеки Debian Linux проекта ev3dev, каталог crosslibs/
4. Кросскомпилятор FreePascal, каталог fpc/. Кросскомпилятор отсутствует в архиве установщика. Он присутствует только в результирующем каталоге на хостовом компьютере и создается в процессе установки из исходного кода компилятора FreePascal, который доступен в репозитарии ОС Linux хостового компьютера. Данный этап установки - самый длительный (до 10 минут) и требует как минимум 1 гигабайт оперативной памяти хостового компьютера, иначе результат не может быть гарантирован. Длительность этапа резко уменьшается (до нескольких десятков секунд) при использовании мощного хостового компьютера и SSD.

Во время установки RubiRobotCross размещает некоторые файлы в домашнем каталоге текущего пользователя, но за пределами основного каталога, а именно:
1. Консольный менеджер библиотеки RubiRobot, файл ~/bin/ev3
2. Отдельные утилиты консольного менеджера, файлы ~/bin/ev3fpc, ~/bin/ev3cp, ~/bin/ev3run и т.д.
3. Конфигурационные файлы консольного менеджера ~/.ssh/ev3ip, ~/.ssh/ev3pass и т.д.
4. Ссылка на кросскомпилятор FreePascal, файл ~/bin/ppcarm
5. Ярлыки графического менеджера на рабочем столе и в главном меню
6. Ссылка на основной каталог RubiRobotCross на рабочем столе

Доступ к исходному коду установщика RubiRobotCross:
Установщик создан при помощи утилиты makeself, что позволяет штатными средствами выполнить его разархивацию без запуска скрипта установки и без последующего удаления временных файлов.

Например:

```
./setupRubiRobot.0.2.2-20180611.run --keep --noexec --target ~/RubiRobot
```

Такой запуск распакует архив установщика в каталог RubiRobot домашнего каталога пользователя, что позволит получить прямой доступ к файлам архива и скрипту установки.

Менеджеры библиотеки RubiRobotLib

Менеджеры библиотеки RubiRobotLib - это вспомогательные программные продукты, предназначенные для функционирования на хостовом компьютере с операционной системой

Linux. Менеджеры позволяют упростить и ускорить процесс разработки программ для EV3, освобождают от многих рутинных операций.

Консольный менеджер ev3 из состава RubiRobotLib

В состав библиотеки RubiRobotLib входит консольный менеджер-утилита **ev3**. Это bash-скрипт, который во время установки библиотеки совместно несколькими вспомогательными файлами копируется в каталог ~/bin и тем самым становится доступным для запуска. Менеджер, с помощью ssh и кросскомпилятора FreePascal, обеспечивает терминальный доступ к EV3, удаленный запуск любых команд, управляет скоростным каналом связи, паролем доступа и IP-адресом EV3, позволяет компилировать, копировать и запускать программы на EV3.

Краткая помощь по утилите доступна запуском ev3 без параметров, развернутая помощь - запуском ev3 с параметром help.

Примеры:
Примеры приведены для ситуации, когда текущим каталогом хостового компьютера является каталог examples библиотеки RubiRobot.

Компиляция примера btn1.pp, копирование и запуск на EV3 полученного исполняемого файла:
```
ev3 fpccprun btn1.pp
```

Компиляция примера btn1.pp и копирование на EV3 полученного исполняемого файла:
```
ev3 fpccp btn1.pp
```

Копирование и запуск на EV3 ранее скомпилированного исполняемого файла btn1:
```
ev3 cprun btn1
```

Запуск исполняемого файла btn1, физически расположенного в каталоге /home/robot на EV3, без каких-либо диагностических сообщений утилиты ev3:
```
ev3 silentrun btn1
```

Запуск графического терминала с подключением к EV3:
```
ev3x shell
```

Вывод содержимого домашнего каталога пользователя robot на EV3:
```
ev3 shell ls
```

Сброс всех соединений с EV3. Полезно в разных нештатных ситуациях.
```
ev3 drop
```

Попытка обнаружить "потерявшийся" EV3 в локальной сети. Требует пароль системного администратора.
```
ev3 fix
```

Принудительная установка IP-адреса EV3 (можно увидеть на дисплее робота). Полезно, когда робот перестает откликаться по штатному имени ev3dev.local и попытки его обнаружить с помощью команд ev3 fix или ev3 ufix не привели к успеху.

```
ev3 setip 10.42.0.33
```

Графический менеджер RubiRobotManager

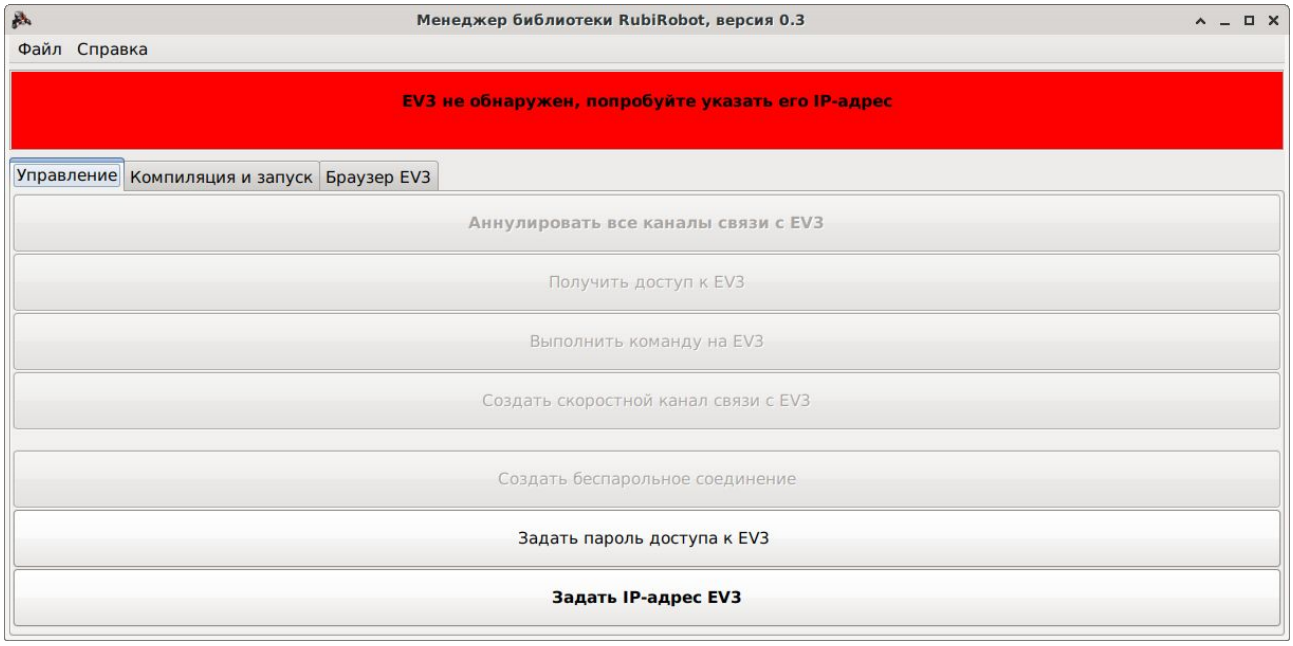
Графический менеджер **RubiRobotManager** представляет собой исполняемый файл для Linux x86-64. RubiRobotManager, в отличие от консольного менеджера, не входит в состав библиотеки RubiRobotLib и является самостоятельным программным продуктом. При установке пакета программ RubiRobotCross графический менеджер и его исходный код копируются в целевой каталог библиотеки. По умолчанию, это каталог ~/RubiRobotCross/RubiRobotManager/ для самого менеджера и ~/RubiRobotCross/RubiRobotManager/src/ для исходного кода. В случае использования на хостовом компьютере платформы, отличной от x86-64, потребуется перекомпиляция менеджера с помощью скрипта makeall.sh из каталога исходного кода менеджера, с последующей заменой исполняемого файла менеджера на его актуальную версию. Начиная с версии 0.2.5 RubiRobotCross позволяет выполнить перекомпиляцию графического менеджера во время установки.

Графический менеджер имеет собственную нумерацию версий и находится в зависимости от определенной версии библиотеки RubiRobotLib. Например, RubiRobotManager версии 0.1 требует наличия на хостовом компьютере библиотеки RubiRobot версии 0.2.3 или выше. Данный факт надо учитывать только при раздельной установке, так как в штатном установщике RubiRobotCross соответствие версий гарантированно соблюдается.

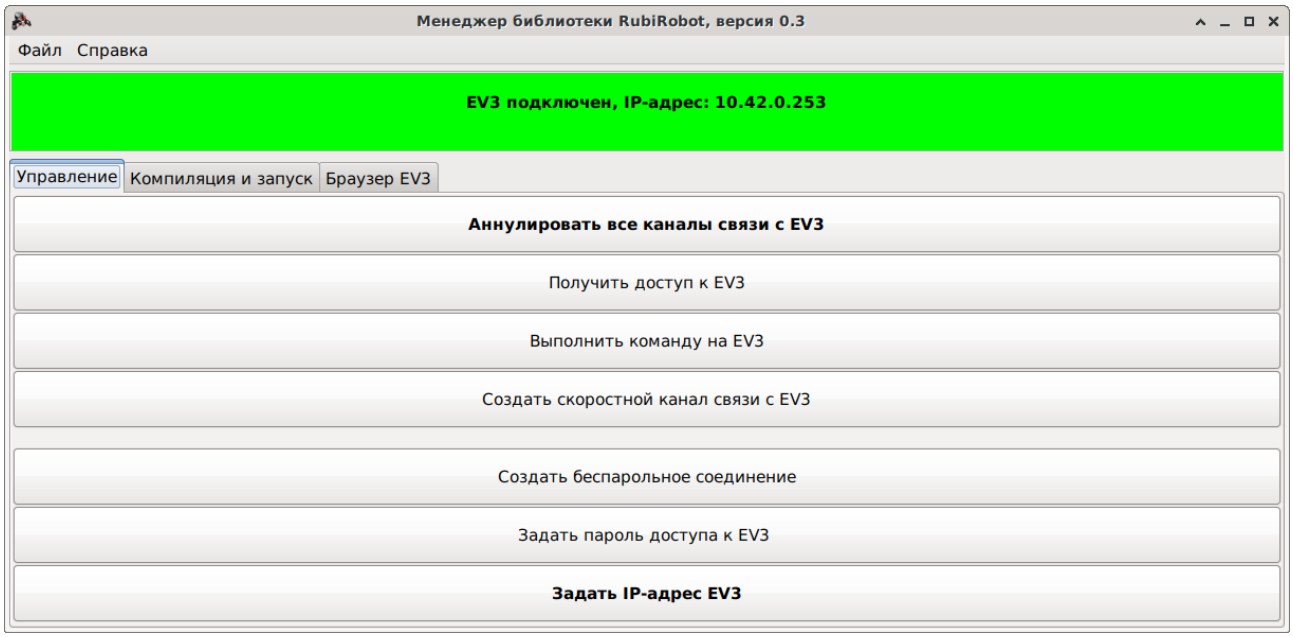
Большая часть возможностей графического менеджера дублирует его консольную версию. Фактически, RubiRobotManager является графической надстройкой над утилитой ev3, с набором дополнительных возможностей. Одной из них является браузер EV3, функциями анализа портов, управления режимами и состоянием некоторых устройств, возможностью просматривать и делать скриншоты дисплея EV3.

При запуске менеджер интегрируется иконкой () в область уведомлений панели задач (системный трей) и периодически (раз в полсекунды) проверяет наличие подключения EV3 к компьютеру. При обнаружении EV3 менеджер сигнализирует об этом, меняя цвет иконки () и показывая во всплывающем сообщении IP-адрес EV3. Аналогичным образом менеджер сигнализирует и о потере связи с EV3.

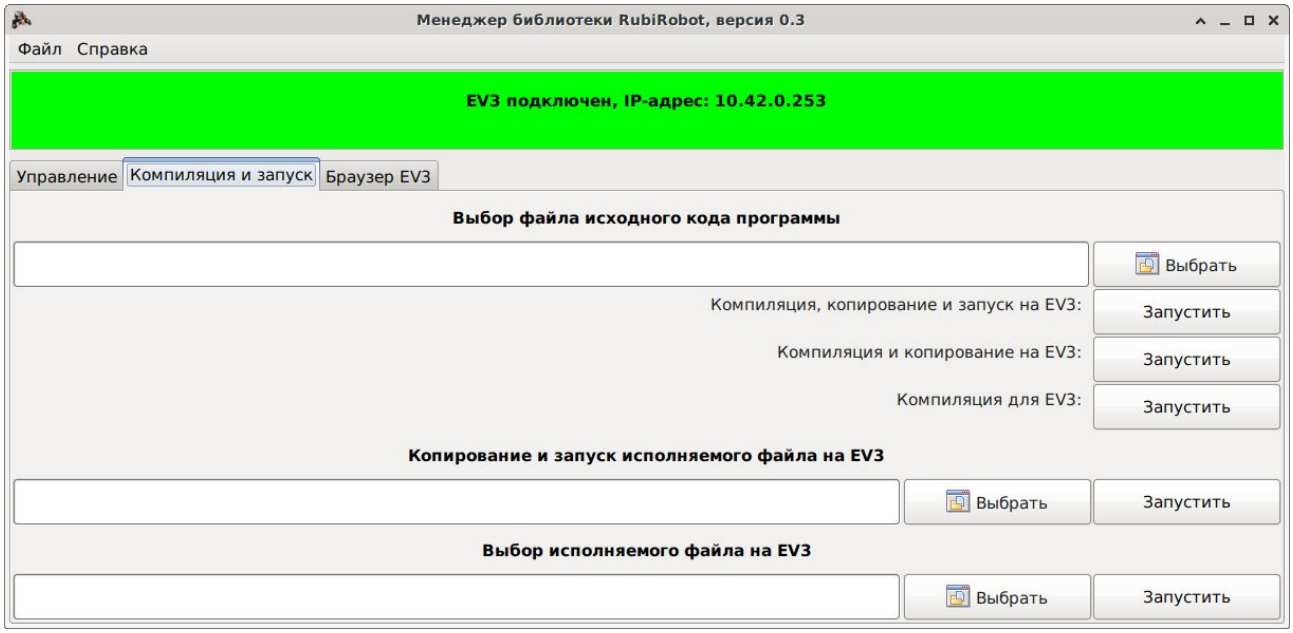
Щелчок по иконке менеджера приводит к открытию его основного окна (см. изображение ниже). Содержимое вкладок менеджера полностью доступно только при подключении к EV3. В ситуации, когда подключенный и получивший корректный IP-адрес EV3 остается недоступным в менеджере, следует указать корректный IP-адрес с помощью кнопки **"Задать IP-адрес EV3"** вкладки "Управление". Также, если доступно несколько роботов одновременно, такой подход позволяет выбрать одного из них.



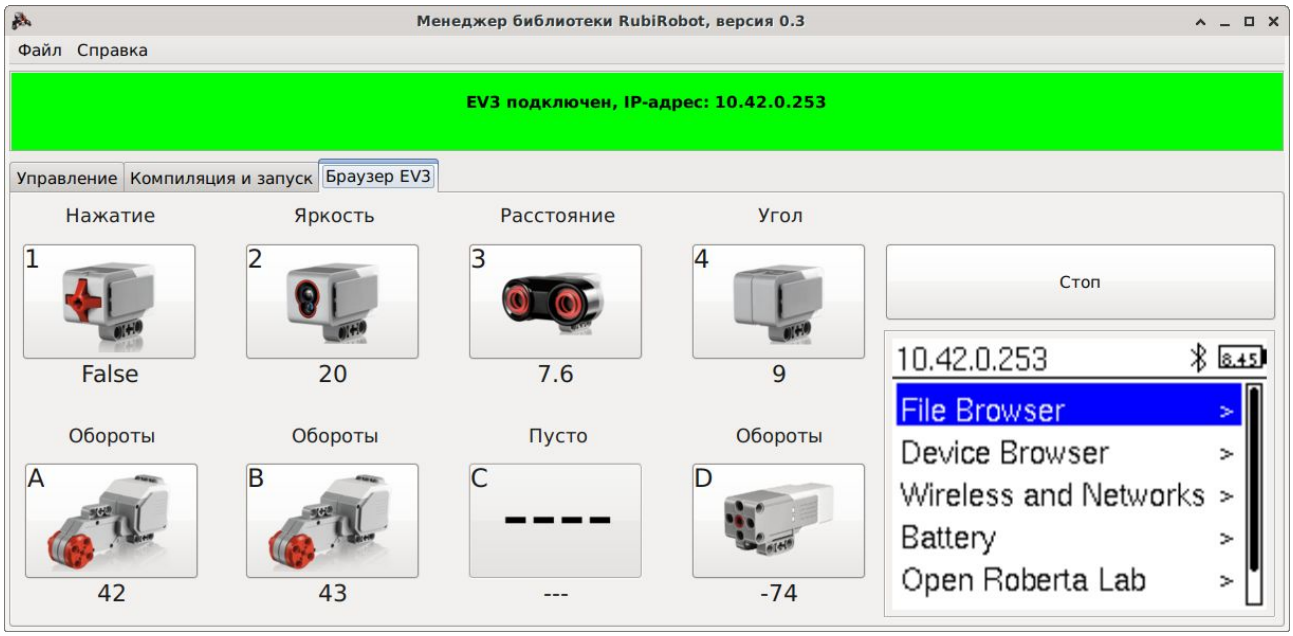
Внешний вид менеджера с подключенным к компьютеру EV3 можно увидеть на изображении ниже. Элемент **"Аннулировать все каналы связи с EV3"** вкладки "Управление" может быть крайне полезен при зависании скоростного канала связи с роботом. К сожалению, данный факт невозможно определить в автоматическом режиме, поэтому при резком замедлении коммуникаций с EV3 рекомендуется нажать на эту кнопку.



Большинство команд во вкладках "Управление" и "Компиляция и запуск" (изображение ниже) так или иначе запускают в графическом терминале консольный менеджер ev3, поэтому результат из действий в целом не отличается от результатов работы утилиты ev3. Выделение двух вкладок связано с наличием двух групп функциональных возможностей утилиты ev3. Первая связана с администрированием подключения к EV3, вторая - с кросскомпиляцией, копированием и запуском приложений на базе библиотеки RubiRobotLib.



Анализатор портов доступен на вкладке "Браузер EV3" (изображение ниже). Информация в анализаторе обновляется 3 раза в секунду, фактически демонстрируя реальное состояние подключенных к роботу внешних устройств. Щелчок по кнопке датчика циклически переключает его режим, если это возможно (например, датчик цвета переключается между режимами "Яркость", "Освещенность" и "Цвет", а датчик-кнопка и датчик расстояния не поддерживают переключение режимов). Щелчок по двигателю обнуляет количество оборотов. Кнопка «Старт/Стоп» запускает и завершает работу браузера. Под кнопкой расположено поле для демонстрации дисплея EV3, содержимое которого обновляется совместно с информацией о портах EV3.



Важной особенностью анализатора портов является блокировка управления устройствами, если на EV3 запущена другая программа на базе библиотеки RubiRobotLib. В таком случае анализатор переходит в режим "только для чтения" и блокирует возможность изменения состояния устройств, что позволяет минимизировать вмешательство в работу других программ. Если этого окажется недостаточно, работу анализатора всегда можно прекратить кнопкой "Стоп". Анализатор завершит соединение с EV3 и изменит надпись на кнопке со "Стоп" на "Старт". Повторное нажатие на кнопку снова запустит анализатор.

С точки зрения реализации, анализатор портов представляет собой клиент-серверное приложение. Сервером является программа pview, исходный код которой доступен в каталоге pview менеджера. Клиентом - отдельный программный поток менеджера. Программа pview встраивается в менеджер при его компиляции, и при необходимости изымается, копируется и запускается на роботе с помощью утилиты ev3. После чего начинается передача данных между сервером и клиентом по двунаправленному каналу связи средствами стандартного ввода-вывода. Таким образом, никаких дополнительных механизмов сетевого взаимодействия между EV3 и менеджером не задействуется, что позволяет экономить вычислительные ресурсы робота.

Возможности библиотеки RubiRobotLib

Подготовка кода и компиляция программы.

Как было сказано выше, существует различный инструментарий для подготовки программы. Самым универсальным, независимым от хостовой операционной системы, является набор консольных текстовых редакторов на самом блоке EV3: mcedit, nano, vim и т. д. В то-же время это самый неудобный набор, с точки зрения скорости подготовки программы, наличия справки, автодополнения кода, браузера кода и т.д. Частично, ситуацию может улучшить использование редактора fp из состава FreePascal, однако известными проблемами в нем являются отсутствие поддержки русского языка и сложность в первоначальном конфигурировании.

При использовании кросскомпилятора на хостовом компьютере появляется дополнительная возможность применять как универсальные, так и специализированные графические среды. К первым можно отнести легковесную среду разработчика Geany, ко вторым - среду Lazarus.

Выбор Geany потребует его начальной настройки на использование утилиты ev3, например - установку в значение "ev3fpccprun %f" второй команды языка программирования Pascal для сборки и запуска программы (элемент меню "Сборка/Установить команды сборки"). Это позволит нажатием клавиши F9 компилировать и запускать на EV3 текущую программу.

Самый широкий набор функциональных возможностей для подготовки кода предоставляет специализированная среда Lazarus. В наборе примеров библиотеки RubiRobotLib имеется подготовленный шаблон-проект для данной среды (examples/lazarus/template.lpi). Ограничением библиотеки RubiRobotLib в Lazarus является невозможность использования отладчика. Способ его отключения подробно описан в начальном комментарии шаблона. В последних версиях Lazarus существует отдельная команда компиляции и запуска программы без использования отладчика, ассоциированная обычно с комбинацией клавиш Ctrl+Shift+F9. Если Lazarus в основном используется для программирования EV3, рекомендуется назначить на указанное действие более привычную клавишу F9.

Библиотека RubiRobotLib широко использует тип Variant для параметров/результатов вызова функций и методов классов. При компиляции программы с использованием последних версий FreePascal преобразование из типа Variant в другие типы и обратно может породить многочисленные предупреждения, которые фактически не несут полезной информации программисту, но засоряют вывод. Для блокирования таких сообщений рекомендуется использовать директиву {$WARN 6058 off} в начале программы/модуля, либо использовать подключаемый файл rubiro.inc из состава библиотеки с заранее подготовленным набором директив компилятора и макросов.

Конфигурация кросскомпилятора FreePascal по умолчанию обеспечивает минимизацию объема исполняемого файла. Это сделано для увеличения скорости копирования на EV3 и запуска программы. Однако в процессе разработки может потребоваться получать подробную информацию о возникающих ошибках. Для этого можно передавать компилятору или утилите ev3 опцию -gl, позволяющую при возникновении ошибки выводить имя файла, функции/процедуры и номер строки, где она произошла. Например, программа snsNXTT.pp

для своей работы требует наличие подключенного к роботу датчика-кнопки NXT. В его отсутствии программа аварийно завершит работу с выводом совершенно неинформативного сообщения, сопровождаемого звуковым сигналом:

```
ev3fpccprun snsNXTT.pp
...
EAccessViolation: Access violation
  $0004EB2C
  $0004CE14
  $00022670
  $00008D90
```

Использование опции -gl даст более информативный результат, который позволит определить строку, где произошла ошибка:

```
ev3fpccprun -gl snsNXTT.pp
...
EAccessViolation: Access violation
  $000505D4
  $0004E8BC
  $00024118
  $00008D90  main,  line 16 of snsNXTT.pp
```

Также, можно полностью перекомпилировать все используемые программой модули библиотеки, добавив опцию -B, что позволит исследовать весь набор вызовов процедур и функций, который привел к ошибке. Однако это важно, в основном, для разработчиков библиотеки.

Указанные опции можно разместить на постоянной основе в конфигурационном файле FreePascal, либо указать в настройках проекта Lazarus.

Инициализация библиотеки

Любая RR-программа должна подключить модуль **cthreads** и **uev3** напрямую или с помощью макроса rubiroBase/rubiroAll и первым действием вызвать процедуру инициализации библиотеки **ev3init()**. Исключением является ситуация, когда программист изменяет значения некоторых базовых переменных для тонкой настройки возможностей библиотеки, после чего вызывает ev3init(). Без предварительного вызова ev3init() большинство обращений к классам, объектам и функциям библиотеки будет завершаться ошибкой с аварийным остановом программы. Штатный набор манипуляций при инициализации подразумевает создание всех необходимых объектов библиотеки, подключение к аппаратным устройствам, специфичная настройка обработки исключительных ситуаций и сигналов ОС Linux. В зависимости от способа вызова, ev3init может провести ряд дополнительных манипуляций.

Процедура ev3init описана в uev3.pp следующим образом:

```
  procedure ev3Init();
  procedure ev3Init(signal:string);
  procedure ev3Init(links:array of const);
  procedure ev3Init(signal:string; links:array of const);
```

Первый вариант процедуры инициализирует библиотеку без каких-либо дополнительных манипуляций.

Второй вариант позволяет при старте программы проиграть звуковой сигнал, закодированный в строке signal. Модуль uev3 содержит 5 константных строк с именами

signal1, …, signal5, с закодированными короткими звуковыми композициями. Любая из этих строк которых может быть передана в ev3init. Например:

```
…
begin
 ev3init(signal4);
 …
end.
```

Третий вариант дает возможность закрепить за отдельными портами конкретные устройства и выполнить их реинициализацию, если они по каким-либо причинам не обнаружены или неверно инициализированы. Данная возможность особенно актуальна на робототехнических соревнованиях, правила которых могут запрещать любые манипуляции с роботом после его включения по окончании карантина (например - правила WRO и RRO). Однако при включении робота существует вероятность некорректного определения отдельных датчиков и двигателей. При невозможности программной или аппаратной реинициализации (например - отключением и повторным подключением устройств), робот практически гарантировано не сможет решить поставленные перед ним задачи, что приведет к поражению команды в текущей попытке.
В массив links процедур ev3init должно быть передано произвольное количество пар идентификаторов порт-устройство. Например:

```
…
begin
 ev3init([1,'ev3color','3','ultra','outD','ev3MediumMotor','outA','большой']);
 …
end.
```

В приведенном примере за портом 1 фиксируется цветовой датчик, за портом 3 - ультразвуковой, за портом D - средний мотор, за портом A - большой мотор.
Если при инициализации библиотеки обнаруживается, что переданный набор не соответствует текущему набору подключений, процедура ev3init выводит на дисплей EV3 информацию об ошибке и пытается реинициализировать некорректные устройства, сопровождая процесс звуковыми сигналами. Попытки реинициализации будут продолжаться до тех пор, пока все устройства не будут корректно определены, либо до принудительной остановки программы.

В таблице ниже приводится набор корректных идентификаторов для всех портов и устройств библиотеки версии 0.2.1 и выше (регистр символов неважен, символы русского алфавита - в кодировке utf8):

Порт или устройство	Корректные идентификаторы
Порт 1	1 '1' 'in1' 'port1'
Порт 2	2 '2' 'in2' 'port2'
Порт 3	3 '3' 'in3' 'port3'
Порт 4	4 '4' 'in4' 'port4'
Порт A	5 'A' 'outA' 'portA'
Порт B	6 'B' 'outB' 'portB'
Порт C	7 'C' 'outC' 'portC'

Порт D	8 'D' 'outD' 'portD'
Ультразвуковой датчик расстояния EV3	'ev3ultra' 'ultra' 'ультра'
Датчик гироскопа EV3	'ev3gyro' 'gyro' 'гиро'
Датчик цвета/света EV3	'ev3color' 'color' 'цвет'
Датчик-кнопка EV3	'ev3touch' 'touch' 'кнопка'
Инфракрасный датчик EV3	'ev3infra' 'infra' 'инфра'
Датчик-кнопка NXT	'nxttouch' 'ntouch' 'нкнопка'
Световой датчик NXT	'nxtlight' 'nlight' 'нсвет'
Большой мотор EV3	'ev3motor' 'lmotor' 'largemotor' 'большой'
Средний мотор EV3	'ev3mmotor' 'mmotor' 'mediummotor' 'средний'

Четвертый вариант инициализации является комбинацией двух предыдущих.

Режимы старта программы.

Существует **3 режима запуска** RR-программы, которые контролируются набором переменных, констант и типов данных в модуле uev3. Кодовые имена режимов: "Герой" (константа rrHero), "Гражданин" (константа rrPatrial) и "Злодей" (константа rrHostile).

Режим "Герой" используется по умолчанию. В этом режиме стартующая RR-программа принудительно завершает работу ранее запущенных RR-программ в режимах "Герой" и "Злодей", а также информирует RR-программы в режиме "Гражданин" о своем запуске. Это позволяет заблокировать возможность одновременного выполнения отдельных RR-программ. Для корректного исполнения большинства программ достаточно использовать данный режим.

В режиме "Гражданин" стартующая RR-программа принудительно завершает работу ранее запущенных RR-программ в режиме "Злодей", а также информирует RR-программы в режимах "Герой" и "Гражданин" о своем запуске. Этот режим позволяет обеспечить одновременное выполнение большинства RR-программ. Данный режим используется например, при работе RR-программы анализатора портов, и позволяет запускать анализатор портов одновременно с RR-программой, решающей основную задачу.

В режиме "Злодей" стартующая RR-программа принудительно завершает работу всех ранее запущенных RR-программ. Этот режим позволяет гарантировать наличие только одной выполняющейся RR-программы в каждый момент времени. Может быть полезен для гарантированного завершения всех выполняющихся RR-программ.

Текущий режим хранится в переменной defaultSelfStatus модуля uev3 и может быть изменен до вызова ev3init(), например:

```
…
begin
 defaultSelfStatus:=rrPatrial;
 ev3init();
```

```
  …
end.
```

Старт программы.

Визуально, переход в рабочий режим RR-программы определяется по желтому цвету цветоиндикаторов блока.

Старт программы в ssh-сессии занимает менее секунды, но та-же самая программа стартует через BrickMan в течении 3-5 секунд. Чтобы избежать этих потерь на робототехнических соревнованиях, рекомендуется в начале программы устанавливать ожидание нажатия кнопки блока или внешней кнопки и запускать робота на выполнение конкурсного задания не запуском программы, а нажатием кнопки.

Следует учитывать, что в прошивке ev3dev-stretch BrickMan запускает программы с повышенным приоритетом исполнения, в то время как в ssh-сессии используется нормальный приоритет. Потенциально это может привести к различию в поведении одной и той-же программы, запущенной различными способами.

Переменная LedsOnStartup модуля uev3 позволяет отключить цветовую индикацию при старте RR-программы. Для этого LedsOnStartup следует присвоить значение false до вызова ev3init(). Это может быть полезным при одновременной работе нескольких RR-программ на EV3.

Начиная с версии 0.2.1 библиотеки, программа в состоянии самостоятельно рестартовать датчики и моторы, которые не были корректно инициализированы при старте робота. В штатной ситуации потерь времени на указанные действия нет, однако в случае необходимости рестарта внешних устройств, робот может потратить до 5 дополнительных секунд на указанные действия. В любом случае, такие потери более предпочтительны, чем невыполнение поставленной задачи.

Штатное и принудительное завершение работы программы.

При необходимости, любая программа на базе библиотеки RubiRobotLib может быть принудительно остановлена длительным нажатием на кнопку BACK. Это верно как при запуске на блоке через менеджер блока - программу BrickMan, так и, начиная с версии 0.2.1 библиотеки, при запуске программы в ssh-сессии. В последнем случае принудительная остановка происходит также при закрытии окна терминала с ssh-сессией, нажатии комбинаций "Ctrl-C" или "Ctrl-\". В новой версии прошивки ev3dev-stretch остановка программы при запуске ее через BrickMan происходит немедленно по нажатию на кнопку BACK, что затрудняет обработку данной кнопки в программе. В октябре 2018 года автор программы BrickMan исправил данную проблему, введя в конфигурационный файл BrickMan дополнительный параметр delay (https://github.com/ev3dev/ev3dev/issues/1162).

При штатном завершении работы программы датчики и моторы, а также все подсистемы EV3 приводятся в состояние по умолчанию. В подавляющем большинстве случаев принудительной остановки или возникновения программных исключений библиотека также

приводит EV3 в состояние по умолчанию. Однако, если прерывание программы по каким-либо причинам не было обработано библиотекой (например - получение программой неперехватываемого сигнала SIGKILL, возникновение аппаратных или непредусмотренных библиотекой программных ошибок), возможны ситуации, когда робот, в отсутствии программы, продолжает выполнять запущенные программой в асинхронном режиме действия и сохраняет нештатное состояние датчиков, цветоиндикации блока и т. д. Например, моторы могут продолжать вращение, звуковой файл - проигрываться, цветоиндикаторы гореть желтым цветом и т. п.

Простейшее решение данной проблемы - запустить любой пример из состава библиотеки RubiRobotLib и немедленно его прервать штатными средствами.

Более эффективное решение - использовать программу rrKill.pp (чистильщик) из набора примеров библиотеки RubiRobotLib:

```
{
 Гарантированное завершение всех запущенных
 ранее rr-программ (rrKill.pp)
}
{$i rubiro.inc}

program rrKill;
uses rubiroBase;
begin
 defaultSelfStatus:=rrHostile;
 ev3init();
end.
```

Существуют и более кардинальные способы: физическое отключение внешних устройств, выключение блока, изъятие аккумулятора.

Модули библиотеки

Модули библиотеки можно условно разделить на 3 части:

1. Базовый модуль uev3.
Содержит функцию инициализации библиотеки ev3Init, основной класс TEv3, объект ev3, средства управления режимами старта RR-программы, несколько макроопределений и, начиная с версии 0.2.5 библиотеки - подавляющее число базовых типов, констант и переменных, функций и процедур, используемых в библиотеке.

2. Модули поддержки основных возможностей EV3.

Модуль	Описание
rubiroScreen	Поддержка дисплея EV3
rubiroSound	Поддержка аудио-системы EV3
rubiroLeds	Поддержка цветовой индикации EV3
rubiroButtons	Поддержка кнопок EV3
rubiroMotors	Поддержка двигателей EV3
rubiroSensors	Поддержка датчиков EV3

Модуль	Описание
rubiroBattery	Поддержка источника питания EV3
rubiroParallel	Поддержка параллелизации кода

3. Вспомогательные модули

Модуль	Описание
uev3devices	Низкоуровневая поддержка подключаемых устройств.
uev3sysfs	Низкоуровневое чтение-запись данных для sysfs-файлов
uev3fb, uev3freetype, uev3ftfont, uev3image	Низкоуровневая поддержка дисплея
uev3func	Низкоуровневые функции библиотеки
fontEnMono.pp, fontEnSans.pp, fontEnSerif.pp, fontRuMono.pp, fontRuSans.pp, fontRuSerif.pp, fontstore.pp, pseudolresources	Поддержка встраиваемых шрифтов.

Начиная с версии 0.2.5 библиотеки, программисту не требуется подключать вспомогательные модули, т. к. основная часть базовых типов данных, констант, переменных, функций и процедур перенесена в модуль uev3. Исключения из этого правила редки и каждый такой случай описывается в документации отдельно.

Перекомпиляция модулей библиотеки

Перекомпиляция модулей библиотеки может быть полезной при изменении исходного кода библиотеки, при смене версии компилятора и т. п. Перекомпиляция производится запуском скрипта makeall.sh, расположенного в каталоге модулей библиотеки.

Подключаемый файл uev3defines.inc содержит несколько полезных макроопределений, изменение состояния которых вступит в силу после перекомпиляции библиотеки. Рассмотрим некоторые из них:

Макроопределение	Описание
EV3DisplayUse (отключено по умолчанию)	Определяет, будет-ли модуль поддержки дисплея rubiroScreen подключаться автоматически. Модуль достаточно громоздкий, и его отключение позволяет значительно (до 2-х раз) сократить размер исполняемого файла программ, которые не используют дисплей EV3. Если модуль отключен, то возможные ошибки инициализации устройств будут выводиться в стандартный поток ошибок. Если модуль подключен, то те-же ошибки будут выводиться на графический дисплей.
EV3SoundUse (включено по умолчанию)	Определяет, будет-ли аудио-модуль rubiroSound подключаться автоматически. Если модуль отключен, то сигнализация о возможных ошибках или сигналы старта программы будут реализованы прямым вызовом внешней аудио-программы. В противном случае, для тех-же целей будет использован объект ev3sound модуля rubiroSound.

EV3LedsUse (включено по умолчанию)	Определяет, будет-ли модуль цветоиндикации rubiroLeds подключаться автоматически. Если модуль отключен, то функции ожидания не будут сигнализировать о факте своей работе миганием цветоиндикаторов. Если при этом модуль не подключить и в основной программе, цветоиндикация запущенной RR-программы также будет полностью отсутствовать.
EV3ButtonsUse (включено по умолчанию)	Определяет, будет-ли модуль поддержки кнопок rubiroButtons подключаться автоматически. Если модуль отключен, то программа не будет завершать свою работу при длительном нажатии клавиши BACK.

Отсутствие автоматического подключения не запрещает принудительного подключения указанных модулей в основной программе.

Особенности классов библиотеки

Особенностью классов библиотеки является то, что большинство методов в качестве результата возвращает ссылку на текущий объект. Это позволяет осуществлять последовательный вызов методов объекта без повторного указания объекта. Например:

```
{$i rubiro.inc}
uses rubiroBase, rubiroMotors;
 var LM: TEv3LargeMotor;
begin
 ev3init();
 LM:=TEv3LargeMotor.Create;
 ...
 LM.setHowStop('стой').run(100).wait();
 ...
end.
```

Другой особенностью является использование типа variant для большинства параметров и повсеместная перегрузка методов классов. Это позволяет использовать одно и то-же имя метода, но с разными типами и количеством параметров, что фактически приближает возможности библиотеки к возможностям интерпретируемых языков с динамической типизацией.

Предопределенные объекты

Предопределенные объекты создаются вызовом процедуры ev3init() из модуля uev3. Наиболее важные из них перечислены в таблице ниже:

Объект	Класс	Модуль	Краткое описание
ev3	TEv3	uev3	Может использоваться для ожидания возникновения событий на других объектах. Поддерживает до 10 таймеров. Прикладным программистом применяется в основном для управления таймерами. Необходимость в создании других экземпляров отсутствует.
ev3buttons	TEv3Buttons	rubiroButtons	Обеспечивает обработку кнопок EV3. Используется повсеместно. Необходимость в создании других экземпляров отсутствует.

Объект	Класс	Модуль	Краткое описание
ev3leds	TEv3Leds	rubiroLeds	Обеспечивает обработку цветоиндикации EV3. Используется повсеместно. Необходимость в создании других экземпляров отсутствует.
ev3screen	TEv3Screen	rubiroScreen	Обеспечивает обработку дисплея EV3 в графическом режиме. Используется повсеместно. Необходимость в создании других экземпляров отсутствует.
ev3sound	TEv3Sound	rubiroSound	Обеспечивает поддержку аудио EV3. Используется повсеместно. Необходимость в создании других экземпляров отсутствует.
ev3battery	TEv3Battery	rubiroBattery	Обеспечивает поддержку источника питания EV3. Используется редко. Необходимость в создании других экземпляров отсутствует.
ev3DeviceFactory	TEv3DeviceFactory	uev3devices	Фабрика устройств. Обеспечивает инициализацию устройств и хранит набор подключенных устройств. Автоматически обновляется: при попытке инициализировать новое устройство, при попытке обращения к отключенному устройству. В последнем случае осуществляет несколько попыток обновления со звуковой и цветовой индикацией. Активно используется для внутренних целей, необходимость доступа к объекту извне возникает крайне редко.

Классы подключаемых устройств

Класс	Модуль	Краткое описание
TEv3InfraSensor	rubiroSensors	Инфракрасный датчик (код 45509). Позволяет определять расстояние до препятствия (до примерно 70 см), направление и расстояние до маяка (код 45508), набор нажатых кнопок на маяке. Операции на маяке проводятся с учетом выбранного канала связи.
TEv3GyroSensor	rubiroSensors	Гироскопический датчик (код 45505). Позволяет определить скорость и направление при движении в горизонтальной плоскости, скорость и угол наклона.
TEv3UltraSensor	rubiroSensors	Ультразвуковой датчик (код 45504). Позволяет определять расстояние до препятствия в миллиметрах, сантиметрах, метрах; оценивать наличие шумов от других датчиков; получать данные сразу, либо ожидать изменившегося значения. Штатно поддерживает режим постоянного непрерывного определения расстояния. Экспериментально поддерживает режим одноразового определения расстояния (не рекомендуется к использованию)
TEv3ColorSensor	rubiroSensors	Датчик цвета/света (код 45506). Поддерживает режимы определения цвета, определения составляющих цвета (RGB), определения уровня отраженного света и определения освещенности. Позволяет получать данные сразу, либо ожидать изменившегося значения
TEv3TouchSensor	rubiroSensors	Датчик касания (код 45507). Позволяет определить текущее состояние кнопки, позволяет ожидать нажатия, отжатия и щелчка.
TNXTTouchSensor	rubiroSensors	Датчик касания (код 9843). Позволяет определить текущее состояние кнопки, позволяет ожидать нажатия, отжатия и щелчка.
TNXTLightSensor	rubiroSensors	Датчик света NXT (код 9844). Поддерживает режимы определения уровня

Класс	Модуль	Краткое описание
		отраженного света и определения освещенности.
TEv3MediumMotor	rubiroMotors	Средний мотор (код 45503). Позволяет запускать мотор с остановкой по истечении времени, по достижению количества оборотов или градусов, а также без ограничений. Дает возможность реверса. Перед запуском и при запуске позволяет указать скорость и один из трех способов остановки, а также в любой момент времени получить эти данные. Позволяет определить текущую скорость и позицию, ждать остановки мотора.
TEv3LargeMotor	rubiroMotors	Большой мотор (код 45502). Возможности эквивалентны среднему мотору.
TEv3Rule	rubiroMotors	Рулевое управление. Позволяет использовать одновременно пару больших моторов. Позволяет запускать моторы с ожиданием остановки по достижению количества оборотов или градусов. Позволяет запускать моторы без ожидания с остановкой по истечении времени. Дает возможность реверса. Перед запуском и при запуске позволяет указать скорость и один из трех способов остановки, а также в любой момент времени получить эти данные. Позволяет определить текущую скорость и позицию, ждать остановки моторов.

Общие свойства классов устройств

Среди общих для всех классов устройств свойств следует выделить логическое свойство **Connected** и свойство перечислимого типа **Port**.
Свойство Connected является универсальным способом определения валидности объекта. Оно возвращает true только в случае, если объект корректно подключен к активному в настоящий момент физическому устройству. Во всех остальных случаях, в том числе и при неудачной инициализации объекта, метод Connected возвращает false.
Свойство Port возвращает порт, к которому подключено физическое устройство, связанное с текущим объектом. Приведение полученного значения к целочисленному типу (например, с помощью функции ord) дает число от 1 до 4 для датчиков, от 5 до 8 - для моторов, 0 - при невалидности объекта.

Средства доступа к объектам, методы инициализации устройств.

Доступ к объектам подключаемых устройств (моторам и датчикам, рулевому управлению) реализуется двумя способами: **классическим** и **функциональным**. Каждый способ поддерживает по два метода инициализации устройств - **фиксированный** и **динамический**.

Объекты подключаемых устройств могут создаваться, обрабатываться и уничтожаться программистом штатными **(классическими)** средствами объектно-ориентированного программирования. Каждый класс устройств содержит два конструктора для поддержки фиксированного и динамического методов инициализации.
Фиксированный метод предполагает создание объекта устройства конструктором Create с одним параметром (номером порта). При этом производится попытка инициализации устройства по указанному порту (в случае неудачи конструктор вернет nil).
Динамический метод использует конструктор Create без параметра. При этом производится попытка инициализации устройства заявленного типа на любом порту (в случае неудачи конструктор вернет nil). Поиск устройства производится от меньшего номера порта к большему. При наличии нескольких устройств приоритет отдается ранее не

инициализированным. Динамический метод позволяет инициализировать устройства только последовательно, без возможности доступа к устройствам на старших портах ранее, чем к устройствам на младших портах.

В приведенном ниже примере предполагается, что к контроллеру подключены два датчика-кнопки, к 1 и 4 портам.

Фиксированный метод инициализации устройств.	Динамический метод инициализации устройств

{$i rubiro.inc}
uses rubiroBase, rubiroSensors;
var t1, t4: TEv3TouchSensor;
begin
 ev3Init();
 // инициализация датчика-кнопки
 // на первом порту
 t1:=TEv3TouchSensor.Create(1);
 // инициализация датчика-кнопки
 // на четвертом порту
 t4:=TEv3TouchSensor.Create(4);
 …
 t1.free;
 t4.free;
end.
	{$i rubiro.inc}
	uses rubiroBase, rubiroSensors;
	var tOne, tTwo: TEv3TouchSensor;
	begin
	 ev3Init();
	 // инициализация первого датчика-кнопки,
	 // по факту - на первом порту
	 tOne:=TEv3TouchSensor.Create;
	 // инициализация второго датчика-кнопки,
	 // по факту - на четвертом порту
	 tTwo:=TEv3TouchSensor.Create;
	 …
	 tOne.free;
	 tTwo.free;
	end.

Существует альтернативный **(функциональный)** способ доступа к объектам подключаемых устройств, с помощью специальных функций, которые объявлены для каждого класса устройств. Функции автоматически создают необходимые объекты при первом обращении и возвращают их в качестве результата. Созданные объекты сохраняются в оперативной памяти и при последующих обращениях возвращаются без попытки повторного создания. Как и в классическом способе, поддерживаются фиксированный и динамический методы создания объектов. При фиксированном методе в соответствующую функцию передается номер порта. Динамический метод предполагает использование "номерных" функций, которых объявлено четыре для каждого устройства и две - для рулевого управлении. Кроме этого, функция фиксированного метода при вызове без параметра является псевдонимом первой функции динамического метода. Динамический метод функционального способа доступа позволяет обращаться к устройствам в любом порядке.

Начиная с версии 0.2.5 библиотеки функциональный способ также может быть использован для приведения типа переданного объекта и выделения объекта из элемента константного массива. Для этого объект или элемент константного массива должен быть передан в качестве параметра в соответствующую функцию (объект должен быть корректного типа, в противном случае генерируется исключительная ситуация.). Такой подход рекомендуется использовать в функциях с параметрами-константными массивами, например - в функциях параллельного исполнения кода.

Классический и функциональный способы доступа к одним и тем-же типам устройств не рекомендуется смешивать в одной программе. Кроме этого, не рекомендуется при штатном способе доступа создавать и использовать различные объекты, ссылающиеся на одно и то-же физическое устройство. На программном уровне подобные действия не запрещены, так как в

некоторых ситуациях они могут быть оправданы (например, при отладке библиотеки разработчиком).

Выбор классического или функционального способа доступа к объектам устройств находится в компетенции программиста. Классический доступ позволяет создавать и уничтожать объекты по мере необходимости, обладает высокой скоростью, позволяет использовать некоторые низкоуровневые возможности библиотеки. Функциональный способ доступа не требует контроля за объектами устройств, не требует объявлять переменные-объекты, автоматически уничтожает объекты по окончании работы. В тоже время, функциональный способ доступа не позволяет, например, отказаться от бесперебойного режима работы библиотеки (см. ниже).
Рекомендуется выбирать классический способ доступа, если критичным фактором является **скорость исполнения** программы. Функциональный способ доступа увеличивает **скорость разработки** программы и ограждает от многих ошибок.

В таблице ниже приведен набор средств функционального доступа к объектам устройств для всех классов библиотеки.

Класс	Модуль	Функции доступа к устройствам

TEv3InfraSensor	rubiroSensors	Фиксированный доступ по номеру порта
		// номера портов от 1 до 4
		function ev3Infra(index:integer):TEv3InfraSensor;
		Динамический доступ по порядку подключения.
		// псевдоним для ev3Infra1
		function ev3Infra:TEv3InfraSensor;
		// доступ к первому подключенному устройству
		function ev3Infra1:TEv3InfraSensor;
		// доступ к второму подключенному устройству
		function ev3Infra2:TEv3InfraSensor;
		// доступ к третьему подключенному устройству
		function ev3Infra3:TEv3InfraSensor;
		// доступ к четвертому подключенному устройству
		function ev3Infra4:TEv3InfraSensor;

TEv3GyroSensor	rubiroSensors	Фиксированный доступ по номеру порта
		// номера портов от 1 до 4
		function ev3Gyro(index:integer):TEv3GyroSensor;
		Динамический доступ по порядку подключения.
		// псевдоним для ev3Gyro1
		function ev3Gyro:TEv3GyroSensor;
		// доступ к первому подключенному устройству
		function ev3Gyro1:TEv3GyroSensor;
		// доступ к второму подключенному устройству
		function ev3Gyro2:TEv3GyroSensor;
		// доступ к третьему подключенному устройству
		function ev3Gyro3:TEv3GyroSensor;
		// доступ к четвертому подключенному устройству
		function ev3Gyro4:TEv3GyroSensor;

TEv3UltraSensor	rubiroSensors	Фиксированный доступ по номеру порта
		// номера портов от 1 до 4
		function ev3Ultra(index:integer):TEv3UltraSensor;
		Динамический доступ по порядку подключения.
		// псевдоним для ev3Ultra1

Класс	Модуль	Функции доступа к устройствам

		function ev3Ultra:TEv3UltraSensor;
		// доступ к первому подключенному устройству
		function ev3Ultra1:TEv3UltraSensor;
		// доступ к второму подключенному устройству
		function ev3Ultra2:TEv3UltraSensor;
		// доступ к третьему подключенному устройству
		function ev3Ultra3:TEv3UltraSensor;
		// доступ к четвертому подключенному устройству
		function ev3Ultra4:TEv3UltraSensor;

TEv3ColorSensor	rubiroSensors	Фиксированный доступ по номеру порта
		// номера портов от 1 до 4
		function ev3Color(index:integer):TEv3ColorSensor;
		Динамический доступ по порядку подключения.
		// псевдоним для ev3Color1
		function ev3Color:TEv3ColorSensor;
		// доступ к первому подключенному устройству
		function ev3Color1:TEv3ColorSensor;
		// доступ к второму подключенному устройству
		function ev3Color2:TEv3ColorSensor;
		// доступ к третьему подключенному устройству
		function ev3Color3:TEv3ColorSensor;
		// доступ к четвертому подключенному устройству
		function ev3Color4:TEv3ColorSensor;

TEv3TouchSensor	rubiroSensors	Фиксированный доступ по номеру порта
		// номера портов от 1 до 4
		function ev3Touch(index:integer):TEv3TouchSensor;
		Динамический доступ по порядку подключения.
		// псевдоним для ev3Touch1
		function ev3Touch:TEv3TouchSensor;
		// доступ к первому подключенному устройству
		function ev3Touch1:TEv3TouchSensor;
		// доступ к второму подключенному устройству
		function ev3Touch2:TEv3TouchSensor;
		// доступ к третьему подключенному устройству
		function ev3Touch3:TEv3TouchSensor;
		// доступ к четвертому подключенному устройству
		function ev3Touch4:TEv3TouchSensor;

TNXTTouchSensor	rubiroSensors	Фиксированный доступ по номеру порта
		// номера портов от 1 до 4
		function nxtTouch(index:integer):TNXTTouchSensor;
		Динамический доступ по порядку подключения.
		// псевдоним для nxtTouch1
		function nxtTouch:TNXTTouchSensor;
		// доступ к первому подключенному устройству
		function nxtTouch1:TNXTTouchSensor;
		// доступ к второму подключенному устройству
		function nxtTouch2:TNXTTouchSensor;
		// доступ к третьему подключенному устройству
		function nxtTouch3:TNXTTouchSensor;
		// доступ к четвертому подключенному устройству
		function nxtTouch4:TNXTTouchSensor;

TNXTLightSensor	rubiroSensors	Фиксированный доступ по номеру порта // номера портов от 1 до 4

Класс	Модуль	Функции доступа к устройствам

		function nxtLight(index:integer):TNXTLightSensor;
		Динамический доступ по порядку подключения.
		// псевдоним для nxtLight1
		function nxtLight:TNXTLightSensor;
		// доступ к первому подключенному устройству
		function nxtLight1:TNXTLightSensor;
		// доступ к второму подключенному устройству
		function nxtLight2:TNXTLightSensor;
		// доступ к третьему подключенному устройству
		function nxtLight3:TNXTLightSensor;
		// доступ к четвертому подключенному устройству
		function nxtLight4:TNXTLightSensor;

TEv3MediumMotor	rubiroMotors	Фиксированный доступ по номеру порта
		// номера портов от 1(5) до 4(8)
		function ev3MMotor(index:integer):TEv3MediumMotor;
		// номера портов от 'A' до 'D' или от 'outA' до 'outD'
		function ev3MMotor(index:string):TEv3LargeMotor;
		Динамический доступ по порядку подключения.
		// псевдоним для ev3MediumMotor1
		function ev3MMotor:TEv3MediumMotor;
		// доступ к первому подключенному устройству
		function ev3MMotor1:TEv3MediumMotor;
		// доступ к второму подключенному устройству
		function ev3MMotor2:TEv3MediumMotor;
		// доступ к третьему подключенному устройству
		function ev3MMotor3:TEv3MediumMotor;
		// доступ к четвертому подключенному устройству
		function ev3MMotor4:TEv3MediumMotor;

TEv3LargeMotor	rubiroMotors	Фиксированный доступ по номеру порта
		// номера портов от 1(5) до 4(8)
		function ev3Motor(index:integer):TEv3LargeMotor;
		// номера портов от 'A' до 'D' или от 'outA' до 'outD'
		function ev3Motor(index:string):TEv3LargeMotor;
		Динамический доступ по порядку подключения.
		// псевдоним для ev3Motor1
		function ev3Motor:TEv3LargeMotor;
		// доступ к первому подключенному устройству
		function ev3Motor1:TEv3LargeMotor;
		// доступ к второму подключенному устройству
		function ev3Motor2:TEv3LargeMotor;
		// доступ к третьему подключенному устройству
		function ev3Motor3:TEv3LargeMotor;
		// доступ к четвертому подключенному устройству
		function ev3Motor4:TEv3LargeMotor;

TEv3Rule	rubiroMotors	Фиксированный доступ для рулевого управления на больших моторах по номеру порта
		// номера портов от 1(5) до 4(8)
		function ev3Rule(index1,index2:integer):TEv3Rule;
		// номера портов от 'A' до 'D' или от 'outA' до 'outD'
		function ev3Rule(index1,index2:string):TEv3Rule;
		Динамический доступ рулевого управления на больших моторах по порядку подключения.
		function ev3Rule:TEv3Rule;

Класс	Модуль	Функции доступа к устройствам

		Фиксированный доступ для рулевого управления на средних моторах по номеру порта
		// номера портов от 1(5) до 4(8)
		function ev3MRule(index1,index2:integer):TEv3Rule;
		// номера портов от 'A' до 'D' или от 'outA' до 'outD'
		function ev3MRule(index1,index2:string):TEv3Rule;
		Динамический доступ рулевого управления на средних моторах по порядку подключения.
		function ev3MRule:TEv3Rule;

Пример ниже демонстрирует использование функционального способа доступа к объектам для определения портов, к которым подключены два датчика-кнопки.

```
{
 Определение портов для двух датчиков-кнопок (sns8.pp)
}
{$i rubiro.inc}
uses rubiroBase, rubiroSensors;
begin
 ev3init();
 if not ev3touch.connected then begin
  writeln('First touchSensor NOT connected'); exit;
 end;
 writeln('First touchSensor connected, port: ', ev3touch.port);
 if not ev3touch2.connected then begin
  writeln('Second touchSensor NOT connected'); exit;
 end;
 writeln('Second touchSensor connected, port: ', ev3touch2.port);
end.
```

Некоторые примеры в библиотеке предлагаются в двух вариантах - с классическим и функциональным способами доступа к устройствам. В последнем случае имя файла программы содержит суффикс "_1".

Бесперебойный режим работы библиотеки

Отключение устройства может быть протестировано методом Connected любого объекта-датчика или объекта-мотора, однако программист фактически не может обработать ситуацию, когда физическое отключение возникло ПОСЛЕ проверки состояния объекта, но ДО обращения к устройству.

Для решения указанной проблемы библиотека RubiRobotLib по умолчанию работает в **бесперебойном режиме**, то есть поддерживает возможность бесперебойной работы объектов датчиков и моторов при **кратковременном** отключении физических устройств с их последующим переподключением к тому-же порту. Такая ситуация часто возникает при слабом контакте соединительных проводов, эксплуатации робота в сложных условиях (резкие повороты, переворачивания, тряска) и т. п.

При попытке обращения к методам отключенных объектов, которые получают доступ к устройству, система блокирует обращение и пытается countMaxRefreshFactory раз (по умолчанию - 10) переподключить отключенное устройство с интервалом в errorAttrInterval

миллисекунд (по умолчанию - 2000) между попытками и звуковой сигнализацией, регулируемой beepError (по умолчанию - true). При успешном переподключении программа продолжает свою работу с места блокировки. В случае неудачи - возникает Exception. Звуковая сигнализация позволяет по тону определить тип отключенного устройства (двигатели - высокий тон, датчики - низкий), и по количеству сигналов - номер порта (от 1 до 4 сигналов).

Недостатком такого подхода является **фиксирование типа устройства на порту, к которому оно однажды было подключено**. Другими словами, переподключение к одному и тому-же порту возможно только для устройств одного типа. В некоторых случаях это мешает корректному функционированию программ. Например, в примере portview.pp библиотеки представлена реализация анализатора портов контроллера. Анализатор должен корректно работать при динамических отключениях и подключениях любых устройств к любым портам. В бесперебойном режиме это невозможно реализовать.

В примере portview.pp представлен алгоритм отключения бесперебойного режима:
1. Блокирование переподключений устройств установкой countMaxRefreshFactory в значение 0.
2. Заключение кода программы в обработчик исключительной ситуации для блокирования реакции на обращение к отключенным устройствами.
3. Периодическое обновление списка подключенных устройств с помощью фабрики устройств ev3deviceFactory.Refresh(true).
4. Периодическая проверка объектов устройств на валидность с помощью Connected и уничтожение невалидных.
5. Периодическая проверка подключенных устройств и создание соответствующих им объектов.

Стоит обратить внимание, что при отключении бесперебойного режима невозможно корректное использование функционального способа доступа к объектам устройств. В следующих версиях библиотеки планируется создание более простого механизма отключения и включения бесперебойного режима, с автоматической поддержкой всех способов доступа к объектам.

Таймауты библиотеки

Запись и чтение значений датчиков, моторов, кнопок и цветоиндикаторов производится (за редким исключением) с использованием виртуальных файлов в файловой системе sysfs, которые, в свою очередь, обслуживаются драйверами соответствующих устройств. Слишком частое чтение или запись в эти файлы может вызвать некорректную работу драйверов и, как следствие, замедление работы, отказ в доступе к устройствам или даже зависание контроллера. Поэтому в библиотеке предусмотрены отдельные таймауты на чтение (переменная getInterval) и запись данных (переменная setInterval). По умолчанию они равны соответственно 5 и 10 миллисекунд, но могут быть увеличены как глобально, так и индивидуально для каждого класса или объекта.

Первым признаком недостаточности используемых таймаутов может служить процесс systemd_udevd, который начинает использовать более 10% процессорного времени и не уменьшает свою загруженность по окончании работы программы (проверяется утилитой top в ssh-сессии). В этом случае программисту рекомендуется начать решать возникшую

проблему путем перезагрузки робота и небольшого увеличения (на 5-10 миллисекунд) переменных setInterval и getInterval в самом начале программы, до вызова функции ev3init. Если проблема сохранится, следует повторить указанные действия.

Для ускорения выполнения операций на роботе, можно обнулить таймауты перед вызовом ev3init(). Рассмотрим пример. Программа tout.pp в бесконечном цикле считывает значение отраженного света с датчика цвета и периодически выводит количество итераций цикла в секунду.

Программа	**Пример вывода программы при нулевых таймаутах**	**Пример вывода программы при таймауте на чтение в 5мс и таймауте на запись в 10мс.**

`{`
` Демонстрация действия`
` таймаутов (tout.pp)`
`}`
`{$i rubiro.inc}`
`uses rubiroBase, rubiroSensors;`
` var cnt:integer=0;`
`  c:integer;`
`begin`
` getInterval:=0; // 5;`
` setInterval:=0; // 10;`
` ev3init();`
` ev3.timerStart;`
` while true do begin`
`  c:=ev3color.reflect;`
`  inc(cnt);`
`  if ev3.timer>1000 then begin`
`  ev3.timerStart;`
`  writeln(cnt);`
`  cnt:=0;`
`  end;`
` end;`
`end.`
	1128
	1313
	1285
	1125
	1293
	1319
	1277
	1304
	1217
	1311
	1277
	1302
	1278
	1298
	1274
	1307
	1267
	1295
	1169
	1273
	1236
	1314
	1286
	1302
		156
		170
		169
		164
		172
		169
		170
		169
		168
		166
		174
		169
		169
		169
		172
		171
		169
		170
		172
		171
		170
		169
		168
		169

Из таблицы можно увидеть, что при нулевых таймаутах скорость работы программы примерно в 7 раз выше, чем при таймаутах по умолчанию.

Средства преобразования типов.

Начиная с версии 0.2.5 библиотека позволяет манипулировать набором псевдотипов (**rr-типов**). Определены правила их преобразования, разработаны функции преобразования и функции определения соответствия типов, причем в качестве аргументов функций могут фигурировать большинство базовых типов, вариантный тип (Variant) и тип элемента константного массива (TVarRec). Средства преобразования типов сосредоточены в обязательном для подключения модуле uev3 и поэтому всегда доступны в rr-программе.

Область применения rr-типов достаточно широка. Это обработка константных массивов в процедурах и функциях, преобразования из строк в числа и обратно, уточнения типов

результатов, возвращаемых функциями и методами различных классов библиотеки, обеспечение корректности вычисления арифметических выражений и т. д.

Именование	Название rr-типа	Тип данных FreePascal, соответствующий rr-типу
Целочисленный	TInt	Int64
Вещественный	TReal	Double
Строковый	TStr	AnsiString
Логический	TBool	Boolean
Объект	TObject	TObject
Класс	TClass	TClass
Указатель	TPointer	Pointer

RR-типы определяются как самостоятельные типы данных, их название также присутствует во всех функциях определения и преобразования псевдотипов. Например, функция преобразования в rr-тип TReal называется **toReal**, а функция, определяющая, является ли ее аргумент значением rr-типа TReal, называется **isReal**.

Рассмотрим правила преобразования псевдотипов друг в друга:

1. Любой псевдотип может быть преобразован в любой псевдотип.
Например, строка может быть преобразована в вещественное значение вне зависимости от ее содержимого, любой объект - в логическое значение, указатель - в класс и т. д.

2. Преобразование в TInt
2.1. Из целочисленного - без изменений
2.2. Из вещественного - с округлением
2.3. Из строки, в виде любого числа - в это число, в случае вещественного - с округлением (Round)
2.4. Из строки, все остальное - в нуль
2.5. Из логического, true - в единицу, false - в нуль
2.6. Из объекта, существующий - в единицу, nil - в нуль
2.7. Из класса - существующий - в единицу, nil - в нуль
2.8. Из указателя - числовым знаковым значением.
2.9. Во всех остальных случаях - в нуль

3. Преобразование в TReal
3.1. Из целочисленного - с добавлением дробной части
3.2. Из вещественного - без изменений
3.3. Из строки, в виде любого числа - в это число
3.4. Из строки, все остальное - в 0.0
3.5. Из логического, true - в 1.0, false - в 0.0
3.6. Из объекта, существующий - в 1.0, nil - в 0.0
3.7. Из класса - существующий - в 1.0, nil - в 0.0
3.8. Из указателя - числовым знаковым значением.

3.9. Во всех остальных случаях - в 0.0

4. Преобразование в TStr

4.1. Из целочисленного - из числа в строку, десятичный формат
4.2. Из вещественного - из числа в строку, до 6 знаков после десятичной точки
4.3. Из строки без изменений
4.4. Из логического - в 'TRUE' или 'FALSE'
4.5. Из объекта, существующий - в имя класса, nil - в пустую строку
4.6. Из класса, существующий - в имя класса, nil - в пустую строку
4.7. Из указателя, числовым знаковым значением в строку.
4.8. Во всех остальных случаях - в пустую строку

5. Преобразование в TBool

5.1. Из целочисленного, 0 - в true, остальное - в false
5.2. Из вещественного, 0.0 - в true, остальное - в false
5.3. Из строки, в виде нулевого числового значения (0, 0.0, 0.000 и т.п.) - в false
5.4. Из строки, пустая строка, 'false','no','ложь','нет' в любом регистре - в false
5.5. Из строки, все остальное - в true
5.6. Из логического, без изменений
5.7. Из объекта, существующий - в true, nil - в false
5.8. Из класса, существующий - в true, nil - в false
5.9. Из указателя, nil в false, остальное - в true
5.10. Во всех остальных случаях - в false

6. Преобразование в TObject

6.1. Из объекта, без изменений
6.2. Из указателя, прямым преобразованием, если оно валидно, иначе - nil
6.2. Во всех остальных случаях → TPointer → TObject

7. Преобразование в TClass

7.1. Из объекта, в класс объекта
7.2. Из класса, без изменений
7.3. Во всех остальных случаях - нулевое значение в nil, ненулевое - в TObject

8. Преобразование в TPointer

8.1. Из целочисленного, прямым преобразованием
8.2. Из объекта, прямым преобразованием
8.3. Из класса, прямым преобразованием
8.3. Из указателя, без изменений.
8.4. Из остальных типов → TInt → TPointer
8.5. Из указателя - без изменений.
8.6. Во всех остальных случаях - в nil

Функции преобразования rr-типов	
function toInt(arg):TInt;	Преобразует аргумент arg в rr-тип TInt
function toReal(arg):TReal;	Преобразует аргумент arg в rr-тип TReal

function toStr(arg):TStr;	Преобразует аргумент arg в rr-тип TStr
function toBool(arg):TBool;	Преобразует аргумент arg в rr-тип TBool
function toObject(arg):TObject;	Преобразует аргумент arg в rr-тип TObject
function toClass(arg):TClass;	Преобразует аргумент arg в rr-тип TClass
function toPointer(arg):TPointer;	Преобразует аргумент arg в rr-тип TPointer

Функции определения rr-типов	
function isInt(arg):boolean;	Возвращает true, если аргумент arg имеет rr-тип TInt
function isReal(arg):boolean;	Возвращает true, если аргумент arg имеет rr-тип TReal
function isStr(arg):boolean;	Возвращает true, если аргумент arg имеет rr-тип TStr
function isBool(arg):boolean;	Возвращает true, если аргумент arg имеет rr-тип TBool
function isObject(arg):boolean;	Возвращает true, если аргумент arg имеет rr-тип TObject
function isClass(arg):boolean;	Возвращает true, если аргумент arg имеет rr-тип TClass
function isPointer(arg):boolean;	Возвращает true, если аргумент arg имеет rr-тип TPointer

Рассмотрим пример:

```
{
 Функция арифметического суммирования всех переданных параметров
 и возврат полученного результата в виде строки с комментариями (rrType1.pp)
}
{$i rubiro.inc}
uses rubiroBase;

function sum(data:array of const):string;
  var s:double; i:integer;
 begin
  s:=0.0;
  for i:=low(data) to high(data) do s:=s+toReal(data[i]);
  result:='Сумма '+toStr(length(data))+' слагаемых равна '+toStr(s);
 end;

begin
 ev3init();
 writeln(sum([1,2.3,true,false,'hello','4.6',nil]));
end.
```

Результат выполнения программы - фраза "Сумма 7 слагаемых равна 8.9". При выполнении функции sum, в соответствии с правилами преобразования rr-типов, значение 1 преобразуется в 1.0, 2.3 - остается без изменений, true - преобразуется в 1.0, false - в 0.0, 'hello' - в 0.0, '4.6' - в 4.6, nil - в 0.0. Полученная сумма 1.0+2.3+1.0+0.0+0.0+4.6+0.0 равна 8.9.

Средства параллельного исполнения кода.

В библиотеке изначально имеются элементы параллелизма, в виде асинхронного управления двигателями, проигрывания аудиофайлов и синтеза речи. Указанные возможности встроены в

объекты библиотеки и не контролируются прикладным программистом. Однако, начиная с версии 0.2.5, библиотека поддерживает возможность параллельного исполнения отдельных процедур и методов классов, передавая прикладному программисту все управление параллельным кодом. Дополнительно обеспечена потокобезопасность всех объектов библиотеки, разработаны простые средства синхронизации. Средства параллельного исполнения кода сосредоточены в модуле **rubiroParallel**.

Библиотека содержит достаточно простые средства параллелизации, однако для их эффективного применения программист должен владеть основными концепциями параллельного программирования. Новичкам рекомендую начать изучение данного вопроса с понятий многопоточности[1] и мьютекса[2].

Параллельное исполнение может быть востребовано при необходимости одновременного выполнения нескольких задач, например: манипуляций с моторами, считывания значений датчиков, обеспечения вывода на экран, выполнения сложных математических расчетов и т. д. Следует понимать, что параллельное исполнение не ускорит работу программы, так как EV3 содержит всего один микропроцессор. Преимуществом параллельного исполнения является возможность отделить друг от друга независимые участки кода, в отдельных случаях синхронизируя их выполнение.

Подготовка к параллельному исполнению начинается с создания параллельной процедуры. В модуле **rubiroParallel** определены 8 доступных типов параллельных процедур.

1. Процедура без параметров:

```
TParallelProcV0=procedure();
```

2-6. Процедуры с количеством вариантных параметров от 1 до 5. Тип variant применяется для обеспечения возможности передачи в процедуру значений большинства простых типов данных:

```
TParallelProcV1=procedure(v1:variant);
TParallelProcV2=procedure(v1,v2:variant);
TParallelProcV3=procedure(v1,v2,v3:variant);
TParallelProcV4=procedure(v1,v2,v3,v4:variant);
TParallelProcV5=procedure(v1,v2,v3,v4,v5:variant);
```

7. Процедура с константным массивом в качестве параметра. В отличие от типа variant, элементами константного массива могут быть указатели, объекты и даже классы. Передача указателей и объектов обеспечивает возможность двусторонней коммуникации между параллельно исполняемыми участками кода.

```
TParallelProcVA=procedure(VA: array of const);
```

8. Метод объекта с константным массивом в качестве параметра. По возможностям полностью идентичен предыдущему варианту.

```
TParallelMethod=procedure(VA: array of const) of object;
```

После определения параллельной процедуры, ее следует запустить с помощью процедуры **parallel**, которая также определена в 8 вариантах:

```
function parallel(proc:TParallelProcV0):dword;
function parallel(proc:TParallelProcV1; V1:variant):dword;
function parallel(proc:TParallelProcV2; V1,V2:variant):dword;
function parallel(proc:TParallelProcV3; V1,V2,V3:variant):dword;
function parallel(proc:TParallelProcV4; V1,V2,V3,V4:variant):dword;
```

1 https://ru.wikipedia.org/wiki/Многопоточность
2 https://ru.wikipedia.org/wiki/Мьютекс

```
function parallel(proc:TParallelProcV5; V1,V2,V3,V4,V5:variant):dword;
function parallel(proc:TParallelProcVA; VA:array of const):dword;
function parallel(proc:TParallelMethod; VA:array of const):dword;
```

В качестве первого параметра в **parallel** передается указатель на параллельную процедуру. Последующие параметры зависят от типа параллельной процедуры. Parallel запускает параллельную процедуру и возвращает идентификатор параллельной задачи.

С полученным идентификатором можно производить два действия:
1. Ожидать завершения параллельной процедуры с помощью **parallelWait**:
   `procedure parallelWait(id:dword);`
2. Аварийно завершить параллельную процедуру с помощью **parallelStop**:
   `procedure parallelStop(id:dword);`

Следует понимать, что аварийное завершение предполагает остановку параллельной процедуры вне зависимости от производимых ею в текущий момент действий и поэтому может привести к разнообразным проблемам, от утечек памяти, до зависания программы. Поэтому **не следует использовать parallelStop без крайней необходимости**.

По завершению работы программы, все параллельные процедуры также завершаются.

Синхронизация параллельно исполняющихся процедур друг с другом и с основной программой может выполняться с помощью двух видов мьютексов (одноместных семафоров):

1. Встроенные мьютексы.
Каждый объект библиотеки содержит 6 заранее инициализированных мьютексов, с номерами от 0 до 5.
Метод **lock** пытается захватить мьютекс с номером ID. Если мьютекс уже захвачен другим потоком, то текущий исполняемый поток блокируется вплоть до освобождения мьютекса.
```
procedure TEv3.lock(ID:integer=0);
```
Метод **unlock** освобождает ранее захваченный текущим потоком мьютекс с номером ID. Не следует освобождать мьютекс без предварительного захвата, это приведет к ошибке.
```
procedure TEv3.unlock(ID:integer=0);
```
Метод **tryLock** пытается захватить мьютекс с номером ID. Если мьютекс уже захвачен другим потоком, то tryLock возвращает false. Если захват прошел успешно, tryLock возвращает true.
```
function TEv3.tryLock(ID:integer=0):boolean;
```

2. Мьютексы, самостоятельно определяемые программистом.
Метод **initMutex** производит инициализацию переданного мьютекса. Эта операция должна быть выполнена **ДО** любых других операций с мьютексом.
```
procedure TEv3.initMutex(var Mutex:TEv3Mutex);
```
Метод **doneMutex** производит уничтожение переданного мьютекса. Эта операция должна быть выполнена над освобожденным или незахваченным мьютексом. После нее никакие операции над мьютексом невозможны, кроме его повторной инициализации.
```
procedure doneMutex(var Mutex:TEv3Mutex);
```
Остальные методы аналогичны методам работы со встроенными мьютексами.
```
procedure TEv3.lock(var Mutex:TEv3Mutex);
procedure TEv3.unlock(var Mutex:TEv3Mutex);
function TEv3.tryLock(var Mutex:TEv3Mutex):boolean;
```

Ниже приведены примеры использования параллельных процедур. Для понимания содержимого примеров, рекомендуется предварительно изучить параграф, описывающий работу с дисплеем EV3.

Пример 1:

```
{
 Пример параллельного исполнения процедур (par1.pp).
 Программа запускает 4 параллельных процедуры,
 заполняющих экран EV3 вертикальной и горизонтальной
 штриховкой, затем в основном потоке отыгрывает
 стартовую мелодию, ожидает завершения
 параллельных процедур и отыгрывает завершающую
 мелодию.
}
{$i rubiro.inc}
uses rubiroBase,rubiroSound,rubiroScreen,rubiroParallel;

procedure lr_draw(l2r:variant);
  var i:integer;
 begin
   for i:=1 to 45 do
    if l2r then ev3Screen.line(i*3,0,i*3,128).show
    else ev3Screen.line(178-i*3,0,178-i*3,128).show;
 end;

procedure ud_draw(u2d:variant);
  var i:integer;
 begin
   for i:=1 to 30 do
   if u2d then ev3Screen.line(0,i*3,178,i*3).show
   else ev3Screen.line(0,128-i*3,178,128-i*3).show;
 end;

 var pl,pr,pu,pd:dword;
begin
 ev3init();
 pl:=parallel(@lr_draw,true);
 pr:=parallel(@lr_draw,false);
 pu:=parallel(@ud_draw,true);
 pd:=parallel(@ud_draw,false);
 ev3sound.beep(signal2);
 parallelWait(pl);
 parallelWait(pr);
 parallelWait(pu);
 parallelWait(pd);
 ev3sound.beep(signal1);
end.
```

Пример 2:

```
{
 Пример параллельного исполнения процедур (par2.pp).
 При отладке алгоритмов часто требуется контролировать
 в реальном времени значения различных датчиков.
```

```
 Параллельная процедура p_show выводит значение
 уровня отраженного света дачика цвета в указанную позицию
 на экране EV3, периодически его обновляя. Программа
 использует одновременно две таких процедуры, для первого и второго
 датчиков цвета.
}
{$i rubiro.inc}
uses rubiroBase,rubiroScreen,rubiroSensors,rubiroParallel;
{
 В параллельную процедуру передаются 4 параметра: координаты вывода x,y,
 объект датчика цвета и указатель на стоп-переменную.
}
procedure p_show(args:array of const);
 var x,y:integer; c:TEv3ColorSensor;
     stop:pboolean; s:string=''; ref:integer;
 begin
  // Заполняем локальные переменные из содержимого константного массива
  x:=toInt(args[0]); y:=toInt(args[1]);
  c:=ev3Color(args[2]); stop:=toPointer(args[3]);
  {
   Устанавливаем в белый цвет карандаш и кисть для будущей
   очистки области вывода значения датчика
  }
  ev3Screen.brushSolid.brushWhite.penWhite;
  {
   Выполняем цикл, пока основная программа не установит
   стоп-переменную в истину
  }
  while not stop^ do begin
   ref:=c.reflect;
   {
    Стираем старое значение датчика и выводим новое, предварительно
    захватив мьютекс. Если не использовать мьютекс, то другая
    параллельная процедура может вывести содержимое холста
    во время его обновления текущей процедурой, что приведет
    к мерцанию экрана.
   }
   ev3.lock;
   // Очищаем экран в области вывода старого значения датчика
   with ev3screen.boundText(x,y,s) do
    ev3screen.rectangle(left,top,right,bottom);
   s:=toStr(ref);
   // Выводим новое зачение датчика
   ev3screen.print(x,y,s).show;
   ev3.unlock;
  end;
  {
   По завершению работы очищаем экран в области вывода
   параллельной процедуры
  }
  with ev3screen.boundText(x,y,s) do
   ev3screen.rectangle(left,top,right,bottom).show;
 end;

 var stop:boolean=false;
begin
 ev3init();
 parallel(@p_show,[10,20,ev3color1,@stop]);
 parallel(@p_show,[10,40,ev3color2,@stop]);
```

```
 readln;
 stop:=true;
 readln;
end.
```

Модуль алгоритмов

Начиная с версии 0.2.6 в библиотеку RubiRobotLib включен модуль полезных алгоритмов rubiroAlgo. В текущей версии rubiroAlgo содержит процедуры, связанные с использованием гироскопа в организации поворотов и прямолинейном движении. Каждая процедура предваряется подробным описанием ее возможностей непосредственно в исходном тексте модуля, поэтому здесь будут приведены только имена и краткое описание процедур.

TurnGyroWheel — интеллектуальный поворот с использованием одного мотора и гироскопа. Может использоваться, например, в роботах на базе "пятиминутки" для поворота вокруг одного колеса. Процедура при старте способна оценивать корректность поворота и реверсировать движение мотора, если заданные параметры заставляют ее удаляться от целевого угла. Для повышения точности позиционирования может снижать скорость поворота, приближаясь к целевому углу, используя пропорциональный регулятор.

TurnGyroAxis — интеллектуальный поворот с использованием двух моторов и гироскопа. Может использоваться, например, в роботах на базе "пятиминутки" для поворота вокруг оси робота. Процедура всегда обеспечивает корректный поворот с приближением к целевому углу. Для повышения точности позиционирования может снижать скорость поворота, приближаясь к целевому углу, используя пропорциональный регулятор.

RunGyroDirectDistance — обеспечивает прямолинейное движение на заданное расстояние с использованием гироскопа и двух моторов. Возврат на исходное направление, в случае принудительного или случайного смещения, производится с помощью пропорционального регулятора.

RunGyroDirectDistanceForParallel — по своим возможностям эквивалентна RunGyroDirectDistance, однако полностью подготовлена для параллельного вызова и может служить примером подготовки параллельных процедур. В случае необходимости может быть вызвана и в последовательном коде.

RunGyroDirectToBlackLine — обеспечивает прямолинейное движение с использованием гироскопа и двух моторов до черной линии, определяемой одним или двумя датчиками цвета/света. Возврат на исходное направление, в случае принудительного или случайного смещения, производится с помощью пропорционального регулятора.

doRunGyro - универсальная процедура прямолинейного движения с использованием гироскопа и двух моторов. Характеризуется стоп-функцией, которая должна быть передана в качестве параметра и определяет момент завершения прямолинейного движения. Процедуры RunGyroDirectDistance, RunGyroDirectDistanceForParallel и RunGyroDirectToBlackLine в конечном итоге вызывают doRunGyro.

Пример:

```
{
 Пример использования модуля алгоритмов (algo1.pp).
 Требуется робот-пятиминутка, датчик гироскопа и средний мотор.

 Робот запускает параллельную процедуру прямолинейного движения с помощью
 гироскопа. Во время выполнения параллельного движения запускаются два
 последовательных вращения среднего мотора в противоположные стороны.
 По окончании последовательных действий робот дожидается завершения
 параллельной процедуры, проигрывает небольшую мелодию и завершает программу.

}

{$i rubiro.inc}

uses rubiroBase,
     rubiroSensors,
     rubiroMotors,
     rubiroParallel,
     rubiroSound,
     rubiroAlgo;

var thread:dword;
begin
 ev3init();
 ev3gyro.calibrate;
 thread:=parallel(@RunGyroDirectDistanceForParallel,[0,20,1000]);
 ev3mmotor.RunTime(1000,30).wait;
 ev3mmotor.RunTime(1000,-30).wait;
 parallelWait(thread);
 ev3sound.beep(signal1).wait();
end.
```

Использование классов и объектов библиотеки RubiRobotLib

Класс TEv3, объект ev3 (модуль uev3)

Класс **TEv3** является базовым для всех остальных классов библиотеки. Класс поддерживает механизмы ожидания различных событий. С версии 0.2 библиотеки класс обеспечивает поддержку до 6 таймеров на объект. С версии 0.2.5 библиотеки класс обеспечивает поддержку до 6 мьютексов на объект (см. параграф **Средства параллельного исполнения кода.**). Инициализация встроенных таймеров и мьютексов реализуется самой библиотекой. Предусмотрена возможность определять свои собственные таймеры и мьютексы, но контроль их состояния является областью ответственности программиста.

Объект **ev3** класса **TEv3** автоматически создается при инициализации библиотеки вызовом процедуры ev3init(). В версии 0.1 библиотеки объект ev3 применялся только в отладочных целях. Начиная с версии 0.2 рекомендуется применять данный объект в ситуациях, когда необходима поддержка таймеров и мьютексов, не связанных с другими объектами библиотеки.

Описание класса TEv3

```
TEv3=class
  constructor Create;
  {
   с v0.2
   блокирует на ms миллисекунд
  }
  procedure sleep(ms:dword);
  {
   Запускает таймер, количество таймеров - 6, с 0 до 5,
   по умолчанию используется таймер с номером 0.
   С v0.2.5 можно использовать свой собственный таймер (ID:double),
   введен для предотвращения гонок в параллельном исполнении.
  }
  procedure timerStart(ID:integer=0);
  procedure timerStart(out ID:double); // собственный таймер
  // возвращает кол-во миллисекунд с момента запуска таймера
  function timer(ID:integer=0):dword;
  function timer(ID:double):dword;
  {
   блокирует на таймаут ms с момента старта таймера
   например: timerStart(5) выполнен в 13:05:23
   если timerWait(6000,5) запущен в 13:05:25,
   то блокировка по 5 таймеру закончится 13:05:29
   если timerWait(6000,5) запущен после 13:05:29,
   то произойдет немедленный выход из функции.
  }
  procedure timerWait(ms:dword; ID:integer=0);
  procedure timerWait(ms:dword; ID:double);
  // v0.2.5
  // Обнуляет таймер на 1970 год
  procedure timerClear(ID:integer=0);
  procedure timerClear(out ID:double);
```

```
    // функции блокировки, 6 мьютексов - с 0 до 5
    procedure lock(ID:integer=0);
    procedure unlock(ID:integer=0);
    function tryLock(ID:integer=0):boolean;
    // возможность использовать собственные мьютексы
    procedure initMutext(var Mutex:TEv3Mutex);
    procedure doneMutex(var Mutex:TEv3Mutex);
    procedure lock(var Mutex:TEv3Mutex);
    procedure unlock(var Mutex:TEv3Mutex);
    function tryLock(var Mutex:TEv3Mutex):boolean;
  end;
```

Примеры использования таймеров объекта ev3

```
{
 Демонстрация работы таймеров на EV3 (tmr1.pp):
}
{$i rubiro.inc}
uses rubiroBase;
begin
 ev3init();
 writeln('Timer0 and Timer1 start');
 ev3.timerStart;
 ev3.timerStart(1);
 writeln('Sleep 2 sec');
 ev3.sleep(2000);
 writeln('Timer0 value = ',ev3.timer,' msec');
 writeln('Timer1 value = ',ev3.timer(1),' msec');
 write('press enter NOW!');readln;
 writeln('Now sleep until timer0 value = 6000 msec.');
 ev3.timerWait(6000);
 writeln('Timer0 sleeping end, value = ',ev3.timer);
 write('press enter NOW!'); readln;
 writeln('Now sleep until timer1 value = 6000 msec.');
 ev3.timerWait(6000,1);
 writeln('Timer1 sleeping end, value = ',ev3.timer(1));
 write('press enter NOW!'); readln;
 writeln('Now sleep until timer1 value = 10000 msec.');
 ev3.timerWait(10000,1);
 writeln('Timer1 sleeping end, value = ',ev3.timer(1));
end.
```

Класс TEv3Buttons, объект ev3buttons (модуль rubiroButtons)

Доступ к кнопкам EV3 реализован через объект **ev3buttons** класса **TEv3Buttons**. Объект автоматически создается при инициализации библиотеки вызовом процедуры ev3init().

У EV3 имеется 6 кнопок: UP, DOWN, LEFT, RIGHT, CENTER, BACK.

Нажатие на кнопку CENTER дополнительно генерирует перевод строки. Таким образом, завершение вызова процедуры readln можно обеспечить не клавишей Enter, а кнопкой

CENTER. Это позволяет использовать единый способ ожидания как в ssh-сессии, так и при запуске программы на блоке.

Обработка нажатия кнопки BACK затруднена в новой версии прошивки ev3dev-stretch, т. к. программа, запущенная через BrickMan немедленно завершает работу по нажатию на данную кнопку. В предыдущей версии прошивки ev3dev-jessie аварийный останов производился только по длительному (2-3 секунды) нажатию на кнопку BACK. Указанная проблема не затрагивает запуск программы в ssh-сессии.

Объект ev3buttons позволяет:

1) Получать текущее состояние кнопок (нажата или отжата)

2) Ожидать возникновения любого события на кнопках

3) Ожидать возникновения заданных событий на кнопках

Существует несколько способов как идентификации кнопок в методах ожидания, так и идентификации состояния кнопок, при этом регистр символов значения не имеет:

Идентификация кнопок:
Кнопка UP, варианты идентификации: 'up','вверх','103','bup'
Кнопка DOWN, варианты идентификации: 'down','вниз','108','bdown'
Кнопка LEFT, варианты идентификации: 'left','влево','105','bleft'
Кнопка RIGHT, варианты идентификации: 'right','вправо','106','bright'
Кнопка CENTER, варианты идентификации: 'center','центр','28','bcenter'
Кнопка BACK, варианты идентификации: любые строки, рекомендуются следующие - 'back','cancel','назад','отмена','bback'

Идентификация состояния кнопок
Кнопка отжата: 'нет','no','0',''
Кнопка нажата: все остальные варианты

Описание класса TEv3Buttons

```
TEv3Buttons=class(TEv3)
    constructor Create();
    destructor destroy;override;
    {
     Ждет нажатия или отпускания клавиши, в Button возвращает одну из
     констант bBACK, bCENTER, bUP, bLEFT, bRIGHT или bDOWN,
     в Press возвращает true, если было нажатие.
    }
    function wait(out Button:variant; out Press:variant):TEv3Buttons;
    {
     Ждет нажатия-отпускания набора кнопок.
     Завершение - при возникновении ВСЕХ событий из переданного массива.
     Каждая кнопка в массиве кодируется строкой вида '14:0','назад:нет'
     (отпускание клавиши Back), '108:1','Down:NO' (нажатие клавиши Down)
     и т.п. Регистр символов неважен. Процесс ожидания сопровождается
     миганием цветоиндикации
    }
    function waitAll(Buttons:array of const):TEv3Buttons;
    {
     ждать нажатия-отпускания набора кнопок
```

```
       завершение - при возникновении ЛЮБОГО события из переданного массива
       каждая кнопка в массиве кодируется строкой (см.WaitEvent)
       процессе ожидания сопровождается миганием цветоиндикации
      }
      function waitAny(Buttons:array of const):TEv3Buttons;
      // возвращает состояние кнопки BACK, true - при нажатии
      property Back:variant read getBack;
      // ожидает нажатия (1 или true) или отпускания (0 или false) кнопки BACK
      function waitBack(press:byte=1):TEv3Buttons;
      function waitBack(press:boolean):TEv3Buttons;
      // возвращает состояние кнопки CENTER, true - при нажатии
      property Center:variant read getCenter;
      //ожидает нажатия (1 или true) или отпускания (0 или false) кнопки CENTER
      function waitCenter(press:byte=1):TEv3Buttons;
      function waitCenter(press:boolean):TEv3Buttons;
      // возвращает состояние кнопки UP, true - при нажатии
      property Up:variant read getUp;
      // ожидает нажатия (1 или true) или отпускания (0 или false) кнопки UP
      function waitUp(press:byte=1):TEv3Buttons;
      function waitUp(press:boolean):TEv3Buttons;
      // возвращает состояние кнопки LEFT, true - при нажатии
      property Left:variant read getLeft;
      // ожидает нажатия (1 или true) или отпускания (0 или false) кнопки LEFT
      function waitLeft(press:byte=1):TEv3Buttons;
      function waitLeft(press:boolean):TEv3Buttons;
      // возвращает состояние кнопки RIGHT, true - при нажатии
      property Right:variant read getRight;
      // ожидает нажатия (1 или true) или отпускания (0 или false) кнопки RIGHT
      function waitRight(press:byte=1):TEv3Buttons;
      function waitRight(press:boolean):TEv3Buttons;
      // возвращает состояние кнопки DOWN, true - при нажатии
      property Down:variant read getDown;
      // ожидает нажатия (1 или true) или отпускания (0 или false) кнопки DOWN
      function waitDown(press:byte=1):TEv3Buttons;
      function waitDown(press:boolean):TEv3Buttons;
    end;
```

Примеры использования кнопок EV3

```
{
 Вывод на экран нажимаемых кнопок EV3 (btn1.pp):
}
{$i rubiro.inc}
uses rubiroBase,rubiroButtons;
begin
 ev3init();
 while not ev3buttons.back do begin
  if ev3buttons.up then writeln('UP');
  if ev3buttons.down then writeln('DOWN');
  if ev3buttons.left then writeln('LEFT');
  if ev3buttons.right then writeln('RIGHT');
  if ev3buttons.center then writeln('CENTER');
 end;
end.

{
 Блокирующее ожидание и вывод на экран
 нажимаемых/отпускаемых кнопок EV3 (btn2.pp)
}
```

```
{$i rubiro.inc}
uses rubiroBase,rubiroButtons;
var  button,press:variant;
begin
 ev3init();
 while true do begin
  ev3buttons.wait(button,press);
  case button of
   bback:writeln('BACK:',press);
   bcenter:writeln('CENTER:',press);
   bleft:writeln('LEFT:',press);
   bright:writeln('RIGHT:',press);
   bup:writeln('UP:',press);
   bdown:writeln('DOWN:',press);
   else writeln('ERROR!');
  end;
  if (button=bback)and(not press) then break;
 end;
end.

{
 Блокирующее ожидание различных комбинаций
 нажатия/отпускания кнопок LEFT и RIGHT (btn3.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroButtons;
begin
 ev3init();
 Writeln('Wait LEFT and RIGHT press');
 ev3buttons.waitAll(['left:да','right:1']);
 Writeln('Wait LEFT and RIGHT release');
 ev3buttons.waitAll(['влево:0','right:no']);
 Writeln('Wait LEFT or RIGHT press');
 ev3buttons.waitAny(['left:да','вправо:YES']);
 Writeln('end');
end.


{
 Блокирующее ожидание нажатий отдельных кнопок
 с цветовой индикацией (btn4.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroButtons,rubiroLeds;
begin
 ev3init();
 Writeln('Wait UP');
 ev3buttons.waitUp;
 ev3leds.all(255,0,255,0);
 Writeln('Wait LEFT');
 ev3buttons.waitLeft;
 ev3leds.start.left(255,0);
 Writeln('Wait RIGHT');
 ev3buttons.waitRight;
 ev3leds.start.right(255,0);
 Writeln('Wait DOWN');
 ev3buttons.waitDown;
 ev3leds.all(0,255,0,255);
 Writeln('Wait CENTER');
```

```
 ev3buttons.waitCenter;
end.
```

Класс TEv3Leds, объект ev3leds (модуль rubiroLeds)

Доступ к цветоиндикаторам EV3 реализован через объект **ev3leds** класса TEv3Leds. Объект автоматически создается при инициализации библиотеки вызовом процедуры ev3init().

EV3 имеет 4 цветоиндикатора: левый красный, левый зеленый, правый красный и правый зеленый. При смешении цветов индикаторов одной стороны можно получать различные оттенки зеленого, красного и оранжевого цветов. Интенсивность цвета определяется целым значением от 0 до 255. Это эксклюзивная возможность ev3dev, в штатной прошивке Lego EV3 индикаторы поддерживают только максимальную интенсивность.

При запуске программы автоматически вызывается метод start, устанавливающий светло-оранжевый цвет на цветоиндикаторах. Используется как визуальное подтверждение старта программы и успешной инициализации библиотеки. При завершении вызывается метод stop, восстанавливающий стандартный зеленый цвет на цветоиндикаторах.

Методы push и pop позволяют сохранять и восстанавливать все значения цветоиндикаторов одновременно. Для этого используется стек, размер которого ограничен только доступной оперативной памятью. Метод blink использует тот-же стек для имитации мигания и применяется, например, в методах waitAny и waitAll базового класса TEv3 (примеры применения - см. документацию по классу TEv3Buttons)

При установке цвета каждого индикатора используется повышенный таймаут - 30мс. Практика показывает, что при меньшем таймауте драйвер цветоиндикатора может "захлебнуться" быстро поступающими данными, что приведет к зависанию контроллера.

Описание класса TEv3Leds

```
TEv3Leds=class(TEv3)
   constructor create();
   destructor destroy; override;

   //запускается при старте, создает светло-оранжевый цвет
   function start:TEv3Leds;

   // запускается по завершении, ввостанавливает штаный зеленый
   function stop:TEv3Leds;

   // сохраняет текущие цвета в памяти (на вершине стека)
   function push:TEv3Leds;

    // ввостанавливает цвета из памяти (с вершины стека)
   function pop:TEv3Leds;

   // гасит все цвета
   function hide:TEv3Leds;

   {
    при последовательном использовании гасит-восстанавливает цвета
```

```
      применяется для имитация мигания
      не рекомендуется использовать совместно с
      push, pop, hide - возможны аберрации
     }
     function blink:TEv3Leds;

     {
       Если в цвета были погашены миганием, восстанавливает их,
       иначе ничего не делает
     }
     function unblink:TEv3Leds;

     {
      Устанавливает интенсивность всех цветоиндикаторов, диапазон значений -
      от 0 до 255 (от минимальной до максимальной интенсивности)
      значение -1 означает оставить соответствующий цвет без изменений
     }
     function All(_leftred,_leftgreen,_rightred,_rightgreen:variant):TEv3Leds;

     // устанавливает интенсивность левых цветоиндикаторов
     function Left(red,green:variant):TEv3Leds;

     // устанавливает интенсивность правых цветоиндикаторов
     function Right(red,green:variant):TEv3Leds;

     // устанавливает интенсивность левого красного цветоиндикатора
     function LeftRed(red:variant):TEv3Leds;

     // устанавливает интенсивность правого красного цветоиндикатора
     function RightRed(red:variant):TEv3Leds;

     // устанавливает интенсивность левого зеленого цветоиндикатора
     function LeftGreen(green:variant):TEv3Leds;


     // устанавливает интенсивность правого зеленого цветоиндикатора
     function RightGreen(green:variant):TEv3Leds;
  end;
```

Примеры использования цветоиндикаторов EV3

```
{
 троекратное "переливание" цветов (led1.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroLeds;
 var i,k:integer;
begin
 ev3Init();
 for k:=1 to 3 do begin
  for i:=0 to 255 do begin
   ev3Leds.all(255-i,i,i,255-i);
  end;
  for i:=0 to 255 do begin
   ev3Leds.all(i,255-i,255-i,i);
  end;
 end;
end.
```

```
{
 демонстрация мигания цветоиндикаторов (led2.pp)
 при отпущенной центральной кнопке - мигание оранжевым
 при нажатой центральной кнопке - мигание красным
 выход из программы - нажатие на кнопку Back
}
{$i rubiro.inc}
uses rubiroBase,rubiroLeds,rubiroButtons,sysutils;
 var
    centered:boolean=false;
begin
 ev3Init();
 while true do begin
  if ev3buttons.center and not centered then begin
   centered:=true;
   ev3leds.unblink.all(255,0,255,0);
  end;
  if not ev3buttons.center and centered then begin
   centered:=false;
   ev3leds.unblink.start;
  end;
  if ev3Buttons.back then break;
  ev3leds.blink();
  sleep(300);
 end;
end.
```

Класс TEv3Screen, объект ev3screen (модуль rubiroScreen)

Доступ к дисплею реализован через объект **ev3screen** класса TEv3Screen. Объект автоматически создается при инициализации библиотеки вызовом процедуры ev3init().

Устройство дисплея различно в ev3dev-jessie и ev3dev-stretch.

В ev3dev-jessie дисплей EV3 рассматривается как монохромный, размером 178x128 точек. В реальности может адресоваться 192x128 точек, однако точки 179-192 каждой строки выходят за пределы дисплея. При этом они существуют, могут быть при необходимости установлены и считаны. Черным цветом на дисплее считается значение 0, белым - 1.

В ev3dev-stretch дисплей EV3 рассматривается как черно-белый, размером 178x128 точек, каждая из которых отображается одним из 4 оттенков серого цвета, от черного до белого. Внутренне каждая точка описывается триплетом RGB (оттенками красного, зеленого и синего с интенсивностью от 0 до 255), что позволяет использовать порядка 16 миллионов цветов.

Библиотека RubiRobotLib обеспечивает корректную работу функций рисования на любом из дисплеев.

Изначально дисплей залит белым цветом, настройки цвета карандаша, кисти и шрифта - черные.

Вызов функций рисования не приводит к немедленному выводу на дисплей. Рисование производится в буфере оперативной памяти, вывод на дисплей реализуется методом **ev3screen.show**. Это позволяет создать достаточно сложный рисунок и одним действием, с минимальным мерцанием, перенести его на дисплей.

Содержимое дисплея может быть скопировано в буфер оперативной памяти с помощью метода **get**. Это может быть полезным для создания "скриншотов" дисплея при работе сторонних программ, с последующим сохранением полученных изображений в PNG-файлах методом **save**.

По умолчанию, текст выводится ttf-шрифтом DejaVuSans.ttf, который должен быть предварительно установлен на EV3 (например - установкой пакета ttf-dejavu). Шрифт по умолчанию может быть изменен заменой содержимого глобальной переменной _fontName до вызова ev3init. Начиная с версии 0.2.5 библиотеки после вызова ev3init шрифт может быть изменен вызовом метода fontType с передачей одной из 6 констант: ruSans, ruSerif, ruMono (шрифты DejaVuSans.ttf, DejaVuSerif.ttf, DejaVuSansMono.ttf с поддержкой русского языка, см. https://ru.wikipedia.org/wiki/DejaVu), enSans, enSerif, enMono (шрифты Vera.ttf, VeraSe.ttf, VeraMono.ttf с поддержкой только латиницы, см. https://ru.wikipedia.org/wiki/Bitstream_Vera). Шрифты с поддержкой только латиницы введены из-за их небольшого размера и могут использоваться при высоких требования к скорости визуализации текста. Шрифт enMono включен в код модуля rubiroScreen и доступен всегда. Если шрифт, заданный глобальной переменной _fontName, не обнаруживается в файловой системе при старте программы, объект ev3screen автоматически переключается на использование шрифта enMono.

Если необходимые для вывода на дисплей шрифты отсутствуют в операционной системе робота (например, при старте rr-программы на оригинальном образе проекта ev3dev), то существует возможность подключить встроенные в библиотеку RubiRobot шрифты. Для это требуется просто подключить один или несколько модулей из следующего списка: fontEnMono, fontEnSans, fontEnSerif, fontRuMono, fontRuSans, fontRuSerif. При подключении модуль автоматически извлекает из своего тела встроенный шрифт и размещает его во временном каталоге. Таким образом шрифт становится доступным для использования методом ev3screen.fontType. Следует учитывать, что размер файла каждого шрифта колеблется от 50 до 700 килобайт, включение их в программу увеличит время компиляции и размер исполняемого файла, а также замедлит старт программы. Поэтому не рекомендуется использовать указанные модули на робототехнических соревнованиях, где обычно дорога каждая секунда.

Модуль rubiroScreen является самым объемным в библиотеке, его подключение автоматически увеличивает исполняемый файл программы примерно на 350 килобайт. Поэтому, если особой необходимости в выводе на дисплей нет - рекомендуется отключать данный модуль, что ускорит процесс копирования исполняемого файла на EV3 и его запуска на роботе.

Описание класса TEv3Screen

```
var _fontName:string='/usr/share/fonts/truetype/dejavu/DejaVuSerif.ttf';

type TEv3Font=(ruSans,ruSerif,ruMono,enSans,enSerif,enMono);
```

```
TEv3Screen=class(TEv3)
    property fontName:string read FfontName write SetfontName;
    constructor create();
    destructor destroy; override;

    // очистка изображения
    function clear:TEv3Screen;
    // вывод изображения на дисплей
    function show:TEv3Screen;
    // получение изображения с дисплея
    function get:TEv3Screen;

    // запись изображения на диск в png-формате
    function save(name:variant; inverse:boolean=false):TEv3Screen;
    // чтение изображения с диска в png-формате
    function load(name:variant; inverse:boolean=false):TEv3Screen;

    // рисование различных фигур карандашом и кистью
    function line(args:array of const):TEv3Screen;
    function line(x,y:variant):TEv3Screen;
    function line(x1,y1,x2,y2:variant):TEv3Screen;
    function rectangle(x1,y1,x2,y2:variant):TEv3Screen;
    function ellipse(x1,y1,x2,y2:variant):TEv3Screen;
    function circle(x,y,R:variant):TEv3Screen;

    // печать теста стандартным шрифтом
    function print(x,y:variant; arg:variant):TEv3Screen;
    function print(x,y,angle:variant; arg:variant):TEv3Screen;
    function print(x,y:variant; args:array of const):TEv3Screen;
    function print(x,y,angle:variant; args:array of const):TEv3Screen;
    function textWidth(arg:variant):integer;
    function textHeight(arg:variant):integer;
    function boundText(x,y:variant; arg:variant):TRect;
    function boundLastText:TRect;

    // установка параметров кисти
    function brushStyle(style:variant):TEv3Screen; // число от 0
    function brushStd():TEv3Screen;
    function brushNone():TEv3Screen;
    function brushSolid():TEv3Screen;

    // установка цвета кисти
    function brushBlack():TEv3Screen;  // черный
    function brushWhite():TEv3Screen;  // белый
    function brushLight():TEv3Screen;  // светло-серый
    function brushDark():TEv3Screen;   // темно-серый
    function brushColor(Red,Green,Blue:byte):TEv3Screen;

    // установка параметров карандаша
    function penBig():TEv3Screen;
    function penSmall():TEv3Screen;
    function penSize(size:variant):TEv3Screen; // число от 1
    function penStyle(style:variant):TEv3Screen; // число от 0
    function penSolid():TEv3Screen;
    function penNone():TEv3Screen;
    function penStd():TEv3Screen;
    // установка цвета карандаша
    function penBlack():TEv3Screen;  // черный
    function penWhite():TEv3Screen;  // белый
```

```
      function penLight():TEv3Screen;  // светло-серый
      function penDark():TEv3Screen;   // темно-серый
      function penColor(Red,Green,Blue:byte):TEv3Screen;

      // установка параметров шрифта
      function fontType(ft:TEv3Font):TEv3Screen;
      function fontBig():TEv3Screen;
      function fontSmall():TEv3Screen;
      function fontRotate(angle:variant):TEv3Screen;
      function fontSize(size:variant):TEv3Screen;
      function fontStd():TEv3Screen;
      // установка цвета шрифта
      function fontBlack():TEv3Screen; // черный
      function fontWhite():TEv3Screen; // белый
      function fontLight():TEv3Screen; // светло-серый
      function fontDark():TEv3Screen;  // темно-серый
      function fontColor(Red,Green,Blue:byte):TEv3Screen;

   public
     // доступ к объекту холста
     property Canvas:TFPImageCanvas read fcanvas;
     // доступ к объекту рисунка
     property Image:TEv3Image read fimg;
  end;
```

Примеры использования дисплея EV3

```
{
 Вывод на экран формулы и результатов ее расчета (scr1.pp):
}
{$i rubiro.inc}
uses rubiroBase,rubiroScreen,sysutils;
begin
 ev3init();
 ev3screen
 .fontsmall.fontsmall
 .print(10,20,['34/56+8*3.2=',34/56+8*3.2])
 .show;
 sleep(3000);
end.

{
 Манипуляции с размером шрифта и поворотом текста (scr2.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroScreen,sysutils;
begin
 ev3init();
 ev3screen
 .print(10,20,'Привет!!!')
 .fontBig()
 .print(10,40,'Привет!!!')
 .fontRotate(-30)
 .fontStd
 .print(10,60,'Поворот -30')
 .show;
 sleep(3000);
 ev3screen
 .clear
```

```
 .fontRotate(180)
 .fontSize(18)
 .print(172,50,'Поворот 180')
 .show;
 sleep(3000);
end.

{
 Простейшая мультипликация (scr3.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroScreen,sysutils;
 var i:integer;
begin
 ev3init();
 ev3screen.fontsmall.fontsmall;
 for i:=1 to 178 do begin
  ev3screen
  .clear
  .print(i,20,'привет СЛЕВА!')
  .print(178-i,40,'привет СПРАВА!')
  .print(i,60,'привет СЛЕВА!')
  .print(178-i,80,'привет СПРАВА!')
  .print(i,100,'привет СЛЕВА!')
  .print(178-i,120,'привет СПРАВА!')
  .show;
 end;
 sleep(3000);
end.

{
 Заполнение дисплея набором отрезков
 со случайными стилем и шириной (scr4.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroScreen,sysutils;
 var i:integer;
begin
 ev3init();
 randomize;
 for i:=1 to 100 do begin
  ev3screen
  .penSize(random(6))
  .penStyle(random(4))
  .line(random(178),random(128),random(178),random(128))
  .show;
 end;
 sleep(3000);
end.

{
 Демонстрация стилей кисти и сохранения содержимого дисплея
 в файле png-формата (scr5.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroScreen,sysutils;
begin
 ev3init();
 ev3screen
```

```
 .brushStyle(6)
 .rectangle(0,0,172,128)
 .brushStyle(0)
 .circle(172/2,128/2,50)
 .show
 .save('scr5.png');
  sleep(3000);
end.
```

Класс TEv3Sound, объект ev3sound (модуль rubiroSound)

Доступ к звуковой подсистеме реализован через объект **ev3sound** класса TEv3Sound. Объект автоматически создается при инициализации библиотеки вызовом процедуры ev3init().

Методы класса осуществляет внешний вызов следующих консольных linux-утилит: amixer (установка громкости), beep (подача звуковых сигналов), play (озвучивание WAV-файла), espeak (синтезатор речи). Таким образом, существует 3 типа звуковых процессов, определяемых идентификаторами 'beep', 'play' и 'speak'.

Библиотека настраивает синтезатор речи на использование русского языка. Это означает, среди прочего, что числительные будут произноситься по русски, даже если находятся внутри английского текста. Используя дополнительный словарь с сайта разработчика espeak, можно значительно улучшить произношение русского текста. Для этого следует загрузить архив http://espeak.sourceforge.net/data/ru_dict-48.zip, разархивировать находящийся в нем файл ru_dict-48 и скопировать его на контроллер, заменяя файл /usr/lib/arm-linux-gnueabi/espeak-data/ru_dict.

Запуск каждого звукового процесса реализуется параллельно основной программе и параллельно другим типам звуковых процессов. Такое поведение можно изменить с помощью метода wait, который блокирует программу в ожидании завершения заданного типа (типов) звукового процесса.

Звуковые процессы разных типов не взаимодействуют между собой, что, с учетом параллельности их работы, позволяет возникать ситуациям, когда разные типы процессов будут озвучены не в том порядке, котором запускались. Например, последовательный запуск ресурсоемкого espeak и высокоскоростного beep без ожидания завершения процессов, скорее всего приведет сначала к подаче звукового сигнала, а затем - к озвучиванию текста синтезатором речи.

Запуск звукового процесса до завершения другого звукового процесса такого-же типа приведет либо к ожиданию завершения предыдущего процесса (метод setWaitBeforeStart(), используется по умолчанию), либо к прерыванию предыдущего процесса (метод setStopBeforeStart()), после чего будет произведен старт нового звукового процесса.

Если программа завершит свою работу до окончания работы звуковых процессов, то они все будут принудительно остановлены.

Следует учитывать, что запуск внешних процессов - относительно трудоемкая операция и не может применяться десятки и тем более сотни раз в секунду. В ситуации, когда частота подачи простых сигналов должна быть очень высокой, следует использовать метод beepDirect, который напрямую взаимодействует с звуковой подсистемой контроллера, без запуска внешних процессов.

Описание класса TEv3Sound

```
TEv3Sound=class
    constructor create;
    destructor destroy;override;

    // Устанавливает уровень громкости (от 0 до 100%)
    function volume(vol:variant):TEv3Sound;
    {
      Подает звуковой сигнал заданной частоты напрямую на устройство.
      Работает очень быстро, без вызовов внешних программ.
      Игнорирует звуковую очередь.
      Для остановки сигнала - передать 0, либо вызвать beep
    }
    function beepDirect(tone:variant):TEv3Sound;
    {
     Подает звуковой сигнал заданной частоты, заданной длительности,
     с заданной паузой по окончании.
     Длительность по умолчанию = 200мс, пауза - 0мс
    }
    function beep(tone:variant):TEv3Sound;
    function beep(tone:variant;duration:variant):TEv3Sound;
    function beep(tone:variant;duration:variant;delay:variant):TEv3Sound;

    {
     Подает набор звуковых сигналов заданной частоты, заданной длительности,
     с заданной паузой по окончании. Информация о сигналах передается в
     массив тройками частота-длительность-пауза.
    }
    function beep(tdws:array of const):TEv3Sound;

    // Озвучивает файл в WAV-формате, имя которого передается в функцию
    function play(wavfile:variant):TEv3Sound;

    {
     Озвучивает переданный текст синтезатором речи,
     скорость по умолчанию = 160 слов в минуту
    }
    function speak(txt:variant):TEv3Sound;
    function speak(txt:variant; speed:variant):TEv3Sound;
    function speak(txts:array of const):TEv3Sound;
    function speak(txts:array of const; speed:variant):TEv3Sound;

    // устанавливает кол-во слов в минуту для синтезатора речи
    function setSpeakSpeed(speed:variant):TEv3Sound;

    // Ожидает завершения любых запущенных звуковых процессов
    function wait():TEv3Sound;

    {
      Ожидает завершения запущенных звуковых процессов
      Параметр what определяет тип процессов:
      'all' - все звуковые процессы
      'beep' - звуковой сигнал
      'play' - озвучивание WAV-файла
      'speak' - синтезатор речи
    }
    function wait(what:variant):TEv3Sound;
```

```
      // Прерывает все запущенные звуковые процессы
      function stop():TEv3Sound;

      // Прерывает запущенные звуковые процессы, параметр what - см. wait()
      function stop(what:variant):TEv3Sound;

      {
       устанавливает режим ожидания завершения звукового процесса перед
       стартом нового звукового процесса такого-же типа. Используется
       по умолчанию
      }
      function setWaitBeforeStart():TEv3Sound;

      {
       устанавливает режим прерывания звукового процесса перед
       стартом нового звукового процесса такого-же типа.
      }
      function setStopBeforeStart():TEv3Sound;
  end;
```

Примеры использования звуковой подсистемы EV3

```
{
 Демонстрация beep с обязательным ожиданием перед завершением программы
 (snd1.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroSound;
begin
 ev3init();
 ev3sound.volume(100).beep(450);
 ev3sound.volume(10).beep(450).wait();
end.

{
 Построчный синтезатор речи. Предназначен для запуска в ssh-сессии.
 (snd2.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroSound;
 var s:string;
begin
 ev3init();
 ev3sound.volume(100);
 while true do begin
  readln(s);
  if s='' then break;
  ev3sound.speak(s,100);
 end;
end.

{
 Демонстрация speak и beep (snd3.pp)
}
{$i rubiro.inc}
uses rubiroBase,rubiroSound;
begin
 ev3init();
```

```
 ev3sound.speak('Имперский марш, слушать всем!',120).wait;
 ev3sound.beep([
  500,500,250,500,500,250,400,500,200,600,200,50,
  500,500,250,400,500,200,600,200,50,500,500,500,
  750,500,250,750,500,250,750,500,250,810,500,200,
  600,200,50,470,500,250,400,500,200,600,200,50,500,500,500
 ]).wait;
end.
```

Класс TEv3Battery, объект ev3battery (модуль rubiroBattery)

Доступ к источнику питания EV3 реализован через объект **ev3battery** класса TEv3Battery. Объект автоматически создается при инициализации библиотеки вызовом процедуры ev3init().

В качестве источника питания EV3 может использовать собственный аккумулятор (артикул 45501), либо 6 обычных или аккумуляторных батареек AA.

Основной характеристикой, важной для прикладного программиста, является текущее выходное напряжение (метод TEv3Battery.volt), которое может колебаться в пределах от 5 до 9 вольт. С практической точки зрения, напряжение ниже 7 вольт является критичным, влияет на корректную работу внешних устройств и может в любой момент привести к прекращению работы EV3. Однако в случае малой нагрузки возможна корректная работа EV3 даже при выходном напряжении меньше 6 вольт, поэтому минимальное напряжение, при котором возможно продолжение функционирования EV3, программист должен определять эмпирически.

Вторичной характеристикой, которая носит во многом формальный характер, является уровень зарядки. При этом минимальное и максимальное выходные напряжения формируется драйвером источника питания в виде константных значений и текущее выходное напряжение может выходить за эти пределы. В таком случае метод TEv3Battery.live будет возвращать значения 0 и 100 соответственно.

Описание класса TEv3Battery

```
  TEv3Battery=class(TEv3)
      constructor create();
      destructor destroy; override;

      { возвращает напряжение в вольтах (вещественное значение)}
      function volt:variant;

      { возвращает уровень зарядки в процентах от 0 до 100 (вещ.значение)}
      function live:variant;
  end;
```

Примеры использования источника питания EV3

```
{
 Получение информации об источнике питания (btr1.pp)
}
{$i rubiro.inc}
```

```
uses rubiroBase,rubiroBattery;
begin
  ev3Init();
  writeln(ev3battery.volt);
  writeln(ev3battery.live);
end.
```

Классы моторов и рулевого управления Lego EV3 (модуль rubiroMotors)

В текущей версии библиотеки поддерживается два Lego-мотора EV3 - большой и средний. Также поддерживается мотор Lego NXT, библиотека определяет его как большой мотор EV3. Отличия между большим и средним моторами - в мощности и скорости вращения (средний мотор более скоростной, но менее мощный, чем большой). Для среднего мотора используется класс **TEv3MediumMotor**, для большого - класс **TEv3LargeMotor**. Оба класса порождены от базового **TEv3TachoMotor**, который предлагает следующие возможности:

1. Запуск мотора.
2. Запуск мотора с остановкой по истечении времени, по достижению количества оборотов или градусов, по достижению абсолютной позиции.
3. Перед запуском и при запуске задание скорости и одного из трех способов остановки.
4. Остановка мотора одним из трех способов остановки.
5. Ожидание остановки/блокирования мотора.
6. Ожидание достижения мотора определенной позиции в различных единицах измерения (вращениях/оборотах/градусах)
7. Получение заданной и текущей скорости, позиции мотора в различных единицах измерения (вращениях/оборотах/градусах), способа остановки.
8. Преобразования между различными единицами измерения (вращения/обороты/градусы).
9. Поддержка до 10 независимых счетчиков вращений/оборотов/градусов на мотор.

До версии 0.2.5 библиотеки программисту была доступна возможность установки реверса. Начиная с версии 0.2.5 реверс не используется в управлении моторами, что позволило значительно ускорить выполнение некоторых операций, а также избавиться от неопределенностей в управлении счетчиками оборотов. Методы управления реверсом перемещены в защищенную область класса.

При создании объектов-моторов можно указывать номер порта. Действительны следующие значения (регистр символов неважен)
для порта A: 5,'5','outA','A';
для порта B: 6,'6','outB','B';
для порта C: 7,'7','outC','C';
для порта D: 8,'8','outD','D';

Если порт не указывается, то объект-мотор присоединяется к ближайшему (слева-направо, от A до D) минимально нагруженному порту, на котором подключен мотор соответствующего типа. Под нагрузкой порта понимается количество присоединенных к порту объектов. В текущей версии библиотеки не имеет смысла нагружать порт более чем одним объектом.

Все методы запуска мотора - асинхронные. Каждый из них запускает мотор определенным способом и немедленно возвращает управление программе. Если мотор запущен одним из

методов, подразумевающих его остановку по возникновению некоторого события (достижения кол-ва оборотов, градусов или истечения временного интервала), то дождаться остановки можно вызовом метода wait. Например:

```
ev3motor.runTurn(1,30,saHold).wait();
```

В данном примере первый подключенный большой мотор запускается на один оборот со скоростью 30% от максимальной и удержанием позиции по окончании оборота, после чего программа ждет его остановки.

Остановка мотора бывает двух видов: 1) **естественная остановка**, по завершению выполнения команды; 2) **искусственная остановка** (используется по умолчанию), например - блокирование мотора на каком-либо препятствии. Причина остановки важна для метода ожидания остановки мотора wait и методов запуска моторов рулевого управления, совмещенных с ожиданием (TEv3Rule.runTurnWait и др.). Следует учитывать, что на крайне низких скоростях в промежутке [-2,2] мотор постоянно находится в состоянии автоблокировки, и вызов метода wait при выборе искусственной остановки немедленно завершит работу мотора. Данный факт учитывается в классе рулевого управления TEv3Rule, и в ситуациях, когда один мотор быстро вращается, а второй находится в автоблокировке (например, при повороте вокруг одного колеса), метод wait рассматривает только естественную остановку второго мотора.

Начиная с версии 0.2.6 искусственная остановка контролируется переменной motorArtificalWait. Это время (в миллисекундах) непрерывной блокировки мотора, по окончании которого мотор считается остановленным. Начальное значение переменной равно 200 миллисекунд.

Изменить способ ожидания остановки можно напрямую, вызовом методом setWaitMode, или косвенно, используя метод wait. Рассмотрим данный механизм на примере метода wait.

Метод wait имеет необязательный параметр realStop, который по умолчанию равен false и позволяет учитывать как искусственную, так и естественностью остановки. Установка значения realStop в true позволит учитывать только естественную остановку, т. е. игнорировать любые виды блокировки мотора во время выполнения, например, следующей команды:

```
ev3motor.runTurn(1,30,saHold).wait(true);
```

Аналогичного эффекта можно добиться использованием других методов ожидания, с последующей принудительной остановкой мотора, например:

```
ev3motor.run(30).waitTurn(1).stop(saHold);
```

Данным механизмом следует пользоваться аккуратно, т. к. при возникновении блокировки мотора, программа может зависнуть в ожидании его естественной остановки.

Переданное в метод wait значение становится значением по умолчанию.

Описание классов TEv3TachoMotor, TEv3MediumMotor, TEv3LargeMotor

```
Вспомогательные типы данных:
{
 тип для определения и установки позиции мотора, для управления счетчиками
 cRot - вращения (обычно 360 вращений = 1 обороту, 1 вращение = 1 градусу)
 cTurn - обороты
 cDeg - градусы
}
TTypeCounter=(cRot,cTurn,cDeg);

Основные классы:

 TEv3TachoMotor=class(TEv3Motor)
    {
      Устанавливает режим ожидания остановки мотора.

      1. Искусственная остановка, вариант по умолчанию в RubiRobotLib.
      Если параметр realStop=false/ноль, то wait ожидает любую
      остановку мотора, в том числе и временную, когда мотор
      блокируется внешним препятствием.

      2. Естественная остановка, вариант по умолчанию в
      LEGO® MINDSTORMS® Education EV3.
      Если параметр realStop=true/не ноль, то wait ожидает
      окончательную остановку мотора.
      Таким образом, если мотор не сможет окончательно остановиться,
      например - из-за внешнего препятствия, wait зависнет.
     }
     function setWaitMode(realStop:variant):TEv3TachoMotor;

     {
      Возвращает текущий режим ожидания остановки,
      естественный (true) или искусственный (false)
     }
     function waitMode(out realStop:variant):TEv3TachoMotor;
     function waitMode:variant;

     {
      Ждет остановки мотора.
      wait без параметров использует режим ожидания остановки,
      установленный по умолчанию (искусственный)
      либо ранее примененный (wait с параметром, см. setWaitMode)
     }
     function wait(realStop:variant):TEv3TachoMotor;
     function wait():TEv3TachoMotor;


     {
      Устанавливает способ остановки мотора.
      Действует при запуске мотора.
      На текущее вращение мотора влияния не оказывает.
      Варианты:
      1) Остановка без отключения подачи электроэнергии,
         средний тормозной путь
         saBrake,'brake','тормоз','тормози','тормозить',1
      2) Остановка с удержанием позиции, минимальный тормозной путь
         saHold,'hold','стоп','стоять','стой',2
      0) Движение накатом, с отключением подачи электроэнергии,
```

```
    максимальный тормозной путь
    saCoast,0,'накат' и другие варианты, не входящие в 1) и 2)
}
function setHowStop(sa:variant):TEv3TachoMotor;

{
 Возвращает способ остановки мотора
  1 - Остановка без отключения подачи электроэнергии,
      средний тормозной путь
  2 - Остановка с удержанием позиции, минимальный тормозной путь
  0 - Движение накатом, с отключением подачи электроэнергии,
      максимальный тормозной путь
}
function howStop(out sa:variant):TEv3TachoMotor;
function howStop:variant;

{
  запускает Счетчик оборотов, количество счетчиков - 10, с 0 до 9
  по умолчанию используется счетчик с номером 0
}
function counterStart(ID:integer=0):TEv3TachoMotor;

{
  возвращает кол-во вращений/оборотов/градусов с момента
  запуска счетчика
  Результат будет отрицательным при обратном вращении
}
function counter(typeCounter:TTypeCounter=cTurn;
   ID:integer=0):variant;
function counter(typeCounter:TTypeCounter;
   ID:integer; out count:variant):TEv3TachoMotor;

{
  блокирует на кол-во вращений/оборотов/градусов с момента
  запуска счетчика.
  count может быть как положительным, так и отрицательным
  (позволяет контролировать реверсивное вращение мотора)
}
function counterWait(count:variant; typeCounter:TTypeCounter=cTurn;
   ID:integer=0):TEv3TachoMotor;

{
  блокирует на кол-во вращений/оборотов/градусов
  с момента запуска счетчика
  count может быть только положительным
  не делает разницы между прямым и реверсивным вращениями
}
function counterAbsWait(count:variant; typeCounter:TTypeCounter=cTurn;
   ID:integer=0):TEv3TachoMotor;

{
  блокирует на кол-во градусов с текущего момента
  deg (градусы) - всегда положительное значение
  не изменяет состояние мотора (не запускает и не останавливает)
  может блокировать бесконечно, если мотор не вращается.
  Для работы использует счетчик номер ID
}
function waitDeg(deg:integer;
   ID:integer=0):TEv3TachoMotor;
```

```
{
  блокирует на кол-во оборотов с текущего момента
  turn (обороты) - всегда положительное значение
  не изменяет состояние мотора (не запускает и не останавливает)
  может блокировать бесконечно, если мотор не вращается.
  Для работы использует счетчик номер ID
}
function waitTurn(turn:double;
   ID:integer=0):TEv3TachoMotor;

{
  пребразования вида градусы <-> вращения <-> обороты
}
function deg2rot(deg:variant):variant;
function rot2deg(rot:variant):variant;
function turn2rot(turn:variant):variant;
function rot2turn(rot:variant):variant;

{
  Устанавливает и возвращает текущую позицию мотора в различных
  единицах измерения
}
function setPosition(p:variant;
  typeCounter:TTypeCounter=cRot):TEv3TachoMotor;
function position(out p:variant;
  typeCounter:TTypeCounter=cRot):TEv3TachoMotor;
function position(typeCounter:TTypeCounter=cRot):variant;

{
  Устанавливает скорость мотора в заданное значение.
  На текущее вращение мотора влияния не оказывает.
}
procedure setSpeed(sp:variant);
{
  Возвращает установленную ранее скорость вращения от -100 до 100
}
function Speed(out sp:variant):TEv3TachoMotor;
function Speed:variant;

{
  Возвращает реальную скорость вращения от 0 до 100
}
function currentSpeed(out sp:variant):TEv3TachoMotor;
function currentSpeed:variant;

{
  Возвращает 1, если установлен реверс и 0 в противном случае
}
function Reverse(out yes:variant):TEv3TachoMotor;
function Reverse:variant;

{
  Останавливает мотор, способ остановки - см. setHowStop
}
function Stop(sa:variant):TEv3TachoMotor;
function Stop():TEv3TachoMotor;

{ v0.2.5
```

```
  Сообщает о статусе остановки мотора
  результат:
  ev3motorRunning/0/false - мотор вращается
  ev3motorStopped/1/true - мотор остановлен
  ev3motorHolding/2/true - мотор остановлен с удержанием позиции
  ev3motorStalled/3/true - мотор временно приостановлен
                           (заблокирован внешним препятствием)
}
function Stoped():variant;
function Stoped(out yes:variant):TEv3TachoMotor;

{
  Запускает мотор (асинхронный вызов). Возможные параметры -
  скорость и способ остановки по умолчанию при вызове stop()
}
function Run(sp:variant; sa:variant):TEv3TachoMotor;
function Run(sp:variant):TEv3TachoMotor;
function Run():TEv3TachoMotor;

{
  Запускает мотор вплоть до достижения абсолютной позиции
  (асинхронный вызов). Знак скорости игнорируется, движение
  направлено всегда в сторону заданной позиции.
  Позиция рассчитывается во вращениях.
  Возможные параметры - скорость и способ остановки
}
function RunToAbsPos(pos:variant; sp:variant; sa:variant):TEv3TachoMotor;
function RunToAbsPos(pos:variant; sp:variant):TEv3TachoMotor;
function RunToAbsPos(pos:variant):TEv3TachoMotor;

{
  Запускает мотор на количество миллисекунд (асинхронный вызов).
  Возможные параметры - скорость и способ остановки
}
function RunTime(ms:variant; sp:variant; sa:variant):TEv3TachoMotor;
function RunTime(ms:variant; sp:variant):TEv3TachoMotor;
function RunTime(ms:variant):TEv3TachoMotor;

{
  Запускает мотор на количество градусов (асинхронный вызов).
  Возможные параметры - скорость и способ остановки
}
function RunDeg(deg:variant; sp:variant; sa:variant):TEv3TachoMotor;
function RunDeg(deg:variant; sp:variant):TEv3TachoMotor;
function RunDeg(deg:variant):TEv3TachoMotor;

{
  Запускает мотор на количество оборотов (асинхронный вызов).
  Возможные параметры - скорость и способ остановки
}
function RunTurn(turn:variant; sp:variant; sa:variant):TEv3TachoMotor;
function RunTurn(turn:variant; sp:variant):TEv3TachoMotor;
function RunTurn(turn:variant):TEv3TachoMotor;

{
  Запуск мотора с прямой подачей напряжения с блока питания
  sp - процент подаваемого напряжения, от -100 до 100
  Отрицательное значение обеспечивает инверсию вращения.
  Скорость вращения дополнительно зависит от уровня зарядки
```

```
      блока питания.
    }
    function directRun(sp:variant):TEv3TachoMotor;

    {
      Изменение напряжения, подаваемого на мотор
      (предварительно однократно вызвать directRun)
    }

    function directSpeed(sp:variant):TEv3TachoMotor;

    {
      Текущее напряжение, подаваемое на мотор
    }
    function currentDirectSpeed(out sp:variant):TEv3TachoMotor;
    function currentDirectSpeed:variant;

    {
      v0.2.5 Разгон и торможение
    }
    // устанавливает время разгона в миллисекундах
    function setRampUp(ms:variant):TEv3TachoMotor;
    // возвращает время разгона в миллисекундах
    function RampUp(out ms:variant):TEv3TachoMotor;
    function RampUp:variant;

    // устанавливает время тороможения в миллисекундах
    function setRampDown(ms:variant):TEv3TachoMotor;
    // возвращает время тороможения в миллисекундах
    function RampDown(out ms:variant):TEv3TachoMotor;
    function RampDown:variant;

    // устанавливает время разгона и торможения в миллисекундах
    function setRamp(ms:variant):TEv3TachoMotor;
    function setRamp(msUp,msDown:variant):TEv3TachoMotor;
  end;

  TEv3LargeMotor=class(TEv3TachoMotor)
   constructor create(port:variant);
   constructor create();
  end;

  TEv3MediumMotor=class(TEv3TachoMotor)
   constructor create(port:variant);
   constructor create();
  end;
```

Примеры использования моторов EV3

```
{
 Демонстрация запуска и остановки
 больших моторов (mot1.pp)
 При запуске с блока вместо клавиши Enter
 можно нажимать кнопку CENTER

 Требования:
 Два больших мотора, левый подключен к
 порту с меньшим номером, правый - к порту
 с большим номером.
```

```
}
{$i rubiro.inc}
uses rubiroBase,rubiroMotors;
 var M1,M2:TEv3LargeMotor;
begin
 ev3init();

 // Подключение к первому мотору
 M1:=TEv3LargeMotor.create();
 // запуск на максимальной скорости
 M1.Run(100);
 // единственная возможность остановить мотор - enter
 readln;
 // остановка с торможением
 M1.stop(saBrake);
 readln;

 // Подключение ко второму мотору
 M2:=TEv3LargeMotor.create();
 // запуск на 3 секунды
 M2.RunTime(3000,100,saBrake);
 // можно остановить мотор до завершения вращения
 readln;
 // остановка с заменой торможения на удержание
 M2.stop(saHold);
 readln;

 // запуск моторов в обратную сторону на 5 оборотов
 // с ожиданием завершения их работы
 M1.runTurn(5,-100,saBrake);
 M2.runTurn(5,-100,saBrake).wait();
 writeln ('Motors stopped');
 readln;
end.
```

Рулевое управление, модуль rubiroMotors

Класс **TEv3Rule** используется для создания рулевого управления на двух моторах и предлагает следующие возможности:

1. Поддержка двух больших или двух средних моторов.
2. Запуск моторов с указанием направления движения.
3. Запуск моторов с указанием направления движения и с остановкой по истечении времени.
4. Блокирующий запуск моторов с указанием направления движения и ожиданием остановки по достижению количества оборотов или градусов.
5. Перед запуском и при запуске: задание скорости и одного из трех способов остановки.
6. Ожидание остановки моторов.
7. Получение заданной и текущей скорости, способа остановки.

При создании объекта-рулевого управления можно указывать номера портов для левого и правого больших моторов (идентификация портов - см. выше). Если порты не указывать, будут задействованы два первых свободных больших мотора. Также можно создавать рулевое управление, указывая класс моторов. Это позволяет реализовать рулевое управление на базе пары любых моторов, ка больших, так и средних.

Описание класса TEv3Rule

```
TEv3Rule=class(TEv3)
   // создание рулевого управления на моторах указанного класса
   constructor create(LPort,RPort:variant; MotorClass:TEv3TachoMotorClass);
   // создание рулевого управления на больших моторах
   constructor create(LPort,RPort:variant);
   {
    создание рулевого управления на моторах указанного класса
    порты определяются автоматически
   }
   constructor create(MotorClass:TEv3TachoMotorClass);
   {
    создание рулевого управления на больших моторах
    порты определяются автоматически
   }
   constructor create();
   destructor destroy;override;

   function Connected:boolean;
   procedure setIntervals(RR,WW:integer);

   // ждет остановки моторов
   function wait():TEv3Rule;
   function wait(realStop:variant):TEv3Rule;

   // Устанавливает способ остановки моторов (см.TEv3TachoMotor)
   function setHowStop(sa:variant):TEv3Rule;
   // Возвращает способ остановки моторов (см.TEv3TachoMotor)
   function howStop(out sa:variant):TEv3Rule;
   function howStop:variant;

   {
     Устанавливает скорость вращения от -100% до 100% (см.TEv3TachoMotor)
   }
   function setSpeed(sp:variant):TEv3Rule;
   {
     Возвращает установленную ранее скорость вращения от -100 до 100
     (см.TEv3TachoMotor)
   }
   function Speed(out sp:variant):TEv3Rule;
   function Speed:variant;

   {
     Возвращает установленную ранее скорость вращения для левого колеса
   }
   function SpeedLeft(out sp:variant):TEv3Rule;
   function SpeedLeft:variant;

   {
     Возвращает установленную ранее скорость вращения для правого колеса
   }
   function SpeedRight(out sp:variant):TEv3Rule;
   function SpeedRight:variant;

   // возвращает реверс (см.TEv3TachoMotor)
   function Reverse(out yes:variant):TEv3Rule;
```

```
  function Reverse:variant;

  // останавливает оба мотора (см.TEv3TachoMotor)
  function Stop(sa:variant):TEv3Rule;
  function Stop():TEv3Rule;

  {
   запускает оба мотора (асинхронный вызов) (см.TEv3TachoMotor)
   direct - направление движения
   Примеры поведения рулевого управления при положительной скорости
   и значения direct:
    100 - поворот вправо вокруг оси,
    -100 - поворот влево вокруг оси,
    50 - поворот вправо вокруг правого колеса,
    -50 - поворот влево вокруг левого колеса
  }
  function Run(direct:variant; sp:variant; sa:variant):TEv3Rule;
  function Run(direct:variant; sp:variant):TEv3Rule;
  function Run(direct:variant):TEv3Rule;

  {
    Запускает моторы на количество миллисекунд (асинхронный вызов).
    Возможные параметры - скорость и способ остановки
  }
  function RunTime(direct:variant; ms:variant;
                   sp:variant; sa:variant):TEv3Rule;
  function RunTime(direct:variant; ms:variant; sp:variant):TEv3Rule;
  function RunTime(direct:variant; ms:variant):TEv3Rule;

  {
    Запускает моторы на количество градусов (БЛОКИРУЮЩИЙ вызов).
    Возможные параметры - скорость и способ остановки
  }
  function RunDegWait(direct:variant; deg:variant;
                      sp:variant; sa:variant):TEv3Rule;
  function RunDegWait(direct:variant; deg:variant; sp:variant):TEv3Rule;
  function RunDegWait(direct:variant; deg:variant):TEv3Rule;

  {
    Запускает моторы на количество оборотов (БЛОКИРУЮЩИЙ вызов).
    Возможные параметры - скорость и способ остановки
  }
  function RunTurnWait(direct:variant; turn:variant;
                       sp:variant; sa:variant):TEv3Rule;
  function RunTurnWait(direct:variant; turn:variant; sp:variant):TEv3Rule;
  function RunTurnWait(direct:variant; turn:variant):TEv3Rule;

   // v0.2.5 Разгон и торможение
   // устанавливает на оба мотора время разгона в миллисекундах
   function setRampUp(ms:variant):TEv3Rule;
   // устанавливает на оба мотора время тороможения в миллисекундах
   function setRampDown(ms:variant):TEv3Rule;
   // устанавливает на оба мотора время разгона и торможения в миллисекундах
   function setRamp(ms:variant):TEv3Rule;
   function setRamp(msUp,msDown:variant):TEv3Rule;
end;
```

Примеры использования рулевого управления на базе моторов EV3

```
{
 Движение по линии до перекрестка
 с помощью двух датчиков света (mot2.pp)

 Требования:
 Два больших мотора, левый подключен к
 порту с меньшим номером, правый - к порту
 с большим номером.
 Два датчика света, левый подключен к
 порту с меньшим номером, правый - к порту
 с большим номером.
}
{$i rubiro.inc}
uses rubiroBase,rubiroMotors,rubiroSensors;
var
 L,R:TEv3ColorSensor;
 Rule:TEv3Rule;

{
 Процедура движения по линии до перекрестка
 с помощью двух датчиков цвета

 maxSpeed - максимально развиваемая скорость
 minSpeed - минимальная скорость
 koef - коэффициент поворота, от 0 (поворот отсутсвует)
      до 1 (крайне резкий поворот)
 borderBreak - граница черного, выход из процедуры
      при сумме значений датчиков, меньшей borderBreak
      (выход по перекрестку)
 borderAcc - граница точности, движение прямо при различии
      значений датчиков меньшем borderAcc

}

procedure run(maxSpeed, minSpeed, koef:double; borderBreak, borderAcc:integer);
 var lref,rref,speed:double;
     direct,prevDirect:double;
begin
 try
  speed:=maxSpeed;
  direct:=0;
  prevDirect:=0;
  Rule.Run(direct,speed);
  while true do begin
    lref:=l.reflect; rref:=r.reflect;
    if lref+rref<borderBreak then exit;
    if abs(lref-rref)<borderAcc then direct:=0
    else direct:=(lref-rref)*koef;
    if (direct=prevDirect)and(speed>=maxSpeed) then continue;
    if (direct=prevDirect) then speed+=1
    else speed:=minSpeed;
    prevDirect:=direct;
    rule.Run(direct,speed);
  end;
 finally
  rule.stop(saBrake);
 end;
```

```
end;

begin
 ev3init();
 Rule:=Tev3Rule.Create;
 l:=TEv3ColorSensor.Create;
 r:=TEv3ColorSensor.Create;
 run(100,40,0.45,50,20);
 readln;
end.



{
 Ручное управление роботом с клавиатуры (mot3_1.pp)

 Требования:
 Два больших мотора, левый подключен к
 порту с меньшим номером, правый - к порту
 с большим номером.

 Запуск - в ssh-сессии, рекомендуется беспроводное
 подключение к роботу.
 При старте программы выводится справка по
 клавишам управления.
}

{$i rubiro.inc}
uses rubiroBase,crt,rubiroMotors,sysutils;

var ch:char; command:char=#0; speed:integer=20;
begin
 writeln('Робот с ручным управлением');
 writeln('Клавиша ESC - завершение работы программы');
 writeln('Клавиши курсора ↑,←,→,↓ - движение робота');
 writeln('Клавиши от 1 до 9 - скорость робота');
 writeln('Остальные клавиши - остановка робота');
 ev3init();
 repeat
   ch:=readkey;
   if ch in ['1'..'9'] then speed:=10*strToInt(ch)
   else if ch=#0 then command:=readkey
   else command:=#0;
   case command of
    #72:ev3rule.Run(0,speed);
    #80:ev3rule.Run(0,-speed);
    #75:ev3rule.Run(-100,speed);
    #77:ev3rule.Run(100,speed)
    else ev3rule.Stop();
   end;
 until ch=#27;
end.
```

Датчики Lego EV3 (модуль rubiroSensors)

В текущей версии библиотеки поддерживается весь набор датчиков Lego-EV3 и датчик-кнопка NXT.

TEv3InfraSensor	Инфракрасный датчик (код 45509). Позволяет определять расстояние до препятствия (до примерно 70 см), направление и расстояние до маяка (код 45508), набор нажатых кнопок на маяке. Операции на маяке проводятся с учетом выбранного канала связи.
TEv3GyroSensor	Гироскопический датчик (код 45505). Позволяет определить скорость и направление при движении в горизонтальной плоскости, скорость и угол наклона.
TEv3UltraSensor	Ультразвуковой датчик (код 45504). Позволяет определять расстояние до препятствия в миллиметрах, сантиметрах, метрах; оценивать наличие шумов от других датчиков; получать данные сразу, либо ожидать изменившегося значения. Штатно поддерживает режим постоянного непрерывного определения расстояния. Экспериментально поддерживает режим одноразового определения расстояния (не рекомендуется к использованию)
TEv3ColorSensor	Датчик цвета/света (код 45506). Поддерживает режимы определения цвета, определения составляющих цвета (RGB), определения уровня отраженного света и определения освещенности. Позволяет получать данные сразу, либо ожидать изменившегося значения
TEv3TouchSensor	Датчик касания (код 45507). Позволяет определить текущее состояние кнопки, позволяет ожидать нажатия, отжатия и щелчка.
TNXTTouchSensor	Датчик касания (код 9843). Позволяет определить текущее состояние кнопки, позволяет ожидать нажатия, отжатия и щелчка.

При создании объектов-датчиков можно указывать номер порта. Действительны следующие значения (регистр символов неважен)
для порта 1: 1,'1','in1';
для порта 2: 2,'2','in2';
для порта 3: 3,'3','in3';
для порта 4: 4,'4','in4';

Если порт не указывается, то объект-датчик присоединяется к ближайшему (слева-направо, от 1 до 4) минимально нагруженному порту, на котором подключен датчик соответствующего типа. Под нагрузкой порта понимается количество присоединенных к порту объектов. В текущей версии библиотеки не имеет смысла нагружать порт более чем одним объектом.

Класс TEv3InfraSensor (модуль rubiroSensors)

Особенностью работы инфракрасного датчика является его тесное взаимодействие с инфракрасным маяком. Указанные устройства следует рассматривать совместно, т. к. маяк предназначен для расширения набора возможностей инфракрасного датчика, который, в свою очередь, имеет все необходимые аппаратно-программные возможности для коммуникации с маяком, а именно - направление и расстояние до маяка, набор нажатых кнопок на маяке. Безотносительно к наличию или отсутствию маяка, инфракрасный датчик поддерживает возможность определения расстояния до препятствия (Proximity mode) в диапазоне [0;100], 0.7 сантиметра на одну единицу измерения, осуществляя замеры каждые 50 миллисекунд.

Еще два режима работы датчика предназначены для коммуникаций с маяком. Доступность маяка определяется двумя параметрами: 1) установкой определенного канала связи (общее количество каналов - 4), который фиксируется переключателем на маяке, что позволяет одновременно использовать как несколько маяков для одного датчика, так и несколько датчиков для одного маяка; и 2) нахождением маяка в передней полусфере датчика. В обоих режимах при нажатой кнопке/кнопках маяк начинает излучать на выбранном канале ИК-сигналы каждые 105-220 миллисекунд

Режим определения расстояния и направления до маяка (IR Seeker mode) позволяет определить расстояние, на котором находится маяк от датчика в диапазоне [0;100], 2 см на 1 единицу измерения, и направление до маяка в диапазоне [-25;25]. Для работы в данном режиме маяк должен быть переведен в состояние импульсного излучения кнопкой BECON, однократное нажатие на которую включает данное состояние (определяется индикатором маяка), либо выключает его. С логической точки зрения нажатие BECON фиксирует ее в нажатом состоянии, а повторное нажатие - отпускает кнопку.

Режим определения нажатых на маяке кнопок (IR Remote mode) позволяет получить комбинацию нажатых на маяке кнопок. В отличие от кнопки BEKON, остальные кнопки не фиксируются при своем нажатии и находятся в этом состоянии ровно столько времени, сколько они физически были нажаты.

После каждого переключения на новый режим, датчик требует реинициализации маяка. В результате практически невозможно программировать датчик со смешением режимов, например - ОДНОВРЕМЕННО получать и обрабатывать

1) расстояние и направление до маяка,
2) наборы кнопок на маяке

Класс TEv3InfraSensor не запрещает переключение режимов, однако средствами датчика проконтролировать техническое состояние маяка невозможно. Поэтому, при отсутствии реинициализации, датчик будет возвращать некорректные результаты (например - неверно определять расстояние до маяка).
Таким образом, основной рекомендацией по программированию инфракрасного датчика является МИНИМИЗИЗАЦИЯ ОДНОВРЕМЕННОГО ИСПОЛЬЗОВАНИЯ следующих наборов методов

1) len
2) direct,distance
3) anybuttons, buttons

Описание класса TEv3InfraSensor и вспомогательных типов данных

```
{
  все допустимые комбинации кнопок инфракрасного маяка,
  для большинства из них в названии указывается цвет и
  местоположение:
     R - red, красная кнопка
     B - blue, синяя кнопка
     U - up, верхняя кнопка
     D - down, нижняя кнопка
  Например, константа irRUBD означает одновременное нажатие
  красной верхней и синей нижней кнопок
}
TEv3IRButton=(irNone,irRU,irRD,irBU,irBD,
              irRUBU,irRUBD,irRDBU,irRDBD,
              irBeacon,irRURD,irBUBD);
```

```
{
 множество, которое может содержать
 любые комбинации кнопок irRU, irRD, irBU и irBD
}
TEv3IRButtons=set of TEv3IRButton;

TEv3InfraSensor=class(TEv3Sensor)
   constructor create(port:variant);
   constructor create();

   // расстояние до препятствия в %, до 0.7 метра
   function len:variant;
   function len(out ln:variant):TEv3InfraSensor;

   {
    направление до маяка, от -25 (слева) до +25 (справа)
    канал - от 1 до 4
   }
   function direct(const channel:variant):variant;
   function direct(const channel:variant; out dir:variant):TEv3InfraSensor;

   {
    расстояние до маяка в %, до 2 метров
    канал - от 1 до 4
   }
   function distance(const channel:variant):variant;
   function distance(const channel:variant; out dist:variant):TEv3InfraSensor;

   {
     нажатые на маяке кнопки, при использовании первого канала;
     любые комбинации кнопок irRU, irRD, irBU и irBD
   }
   function anyButtons():TEv3IRButtons;
   function anyButtons(out butt:TEv3IRButtons):TEv3InfraSensor;

   {
     нажатые на маяке кнопки, при использовании канала от 1 до 4;
     возвращает факт нажатия одной кнопки или комбинации двух кнопок.
     может анализироваться как значение типа TEv3IRButton,
     либо как целое значение из диапазона от 0 до 11
   }
   function buttons(const channel:variant):variant;
   function buttons(const channel:variant; out butt:variant):TEv3InfraSensor;
  end;
```

Примеры использования инфракрасного датчика EV3

```
{
 Демонстрация работы инфракрасного датчика и маяка
 "Робот на поводке" (sns5.pp)

 Требования:
 Два больших мотора, левый подключен к
 порту с меньшим номером, правый - к порту
 с большим номером.
 Инфракрасный датчик, подключен к любому порту.
 Инфракрасный маяк на первом канале.
```

```
}
{$i rubiro.inc}
uses rubiroBase,rubiroSensors,rubiroMotors;
 var
 Infra:TEv3InfraSensor;
 rule:TEv3Rule;
 i,d:integer;
begin
  ev3Init();
  infra:=TEv3InfraSensor.create();
  Rule:=TEv3Rule.create();
  while true do begin
    i:=infra.distance(1);
    d:=infra.direct(1);
    writeln(i,' ',d);
    if (i<30)or(i=100)or(i<0) then rule.stop
    else rule.run(d*4,10+i/2);
  end;
end.

{
 Демонстрация инфракрасного датчика и маяка (sns6.pp)
 "Дистанционное управление"

 Требования:
 Два больших мотора, левый подключен к
 порту с меньшим номером, правый - к порту
 с большим номером.
 Инфракрасный датчик, подключен к любому порту.
 Инфракрасный маяк на первом канале.
}
{$i rubiro.inc}
uses rubiroBase,rubiroSensors,rubiroMotors,rubiroButtons;
var
 Infra:TEv3InfraSensor;
 LM,RM:TEv3LargeMotor;
 s:TEv3IRButtons;
begin
  ev3Init();
  infra:=TEv3InfraSensor.create();
  LM:=TEv3LargeMotor.create();
  RM:=TEv3LargeMotor.create();
  while true do begin
    if ev3buttons.back then break;
    s:=infra.anybuttons;
    if [irRU,irRD]*s=[] then RM.stop.setSpeed(0);
    if [irBU,irBD]*s=[] then LM.stop.setSpeed(0);
    if irRU in s then RM.Run(RM.Speed+1);
    if irRD in s then RM.Run(RM.Speed-1);
    if irBU in s then LM.Run(LM.Speed+1);
    if irBD in s then LM.Run(LM.Speed-1);
  end;
end.
```

Класс TEv3GyroSensor (модуль rubiroSensors)

Первые реализации гироскопического датчика EV3 использовали одноосевой MEMS (микроэлектромеханический) гироскоп. Современные версии гироскопа, используемого в EV3, являются двухосевымы, позволяют получать значения угла и скорости поворота при движении в горизонтальной плоскости, угла и скорости наклона, выполнять обнуление угла поворота и калибровку. Точность измерения декларируется в ±3 градуса на 90 градусов, максимально допустимая скорость вращения/наклона - 440 градусов в секунду.

В использовании гироскопического датчика существует несколько особенностей. Во первых, при подаче питания на гироскоп он должен находиться в состоянии покоя. Если робот будет в состоянии движения при инициализации гироскопа, то скорость его вращения будет принята за 0, а последующая остановка приведет к ненулевому значению скорости поворота в состоянии фактического покоя и перманентному наращиванию абсолютного значения угла поворота, что известно под терминами «смещение» или «дрейф». Проблему можно решить реинициализацией гироскопа калибровкой, зафиксировав предварительно датчик в состоянии покоя, что позволяет программно решить проблему "плывущих" значений угла поворота при вращении вокруг главной оси. Второй особенностью датчика является необходимость поворота строго вокруг оси гироскопа для обеспечения точности измерения угла поворота. Третьей особенностью - латентность процессов измерения скорости и расчета угла поворота благодаря свойству фазовой задержки MEMS-гироскопа, которая может достигать нескольких десятков миллисекунд.

Начиная с версии 0.2.6 библиотеки появилась возможность компенсировать некоторые из проблем гироскопического датчика заданием специальных коэффициентов. Это компенсация погрешности измерений, возникающих при поворотах со смещением оси вращения работа по отношению к оси гироскопа (в дальнейшем — осевые погрешности), а также погрешностей, связанных с аналогово-цифровым преобразованием (в дальнейшем - АЦП-погрешности).

Осевые погрешности зависят от корректности расположения датчика и плотностью его закрепления на роботе. Большую роль в величинах и постоянстве осевых погрешностей играет плавность движения робота и ровность поверхности, по которой он перемещается. Обычно осевая погрешность тем больше, чем больший угол поворота, вне зависимости от его знака, зафиксирован в данный момент на датчике, и равна нулю при нулевом угле поворота.

АЦП-погрешности носят обычно интегральный характер, накапливаются при каждом измерении, индивидуальны для каждого датчика. Для некоторых датчиков они пренебрежимо малы, для других могут доходить до нескольких градусов на полный оборот. Обычно, АЦП-погрешность тем больше, чем больше поворотов совершал робот, может быть как положительной, так и отрицательной, в зависимости от особенностей конкретного датчика.

Оба типа погрешностей контролируются с помощью коэффициентов, т. е. величине погрешности на 1 градус поворота. Коэффициент осевой погрешности доступен через свойство axisDriftRate, АЦП-погрешности — через свойство integralDriftRate. Конкретные величины коэффициентов могут быть вычислены только экспериментальным путем. Для АЦП-погрешности это можно сделать один раз для каждого гироскопа. Для осевой погрешности это следует сделать для каждой конструкции робота и для каждой поверхности, где он будет использоваться.

Описание класса TEv3GyroSensor

```
  TEv3GyroDrift=(drAxis,drIntegral);
  TEv3GyroSensor=class(TEv3Sensor)
    constructor create(port:variant);
    constructor create();

    // Установка-получение базовой угловой позиции
    function setBaseAngle(baseAngle:variant):TEv3GyroSensor;
    function baseAngle:variant;
    // Увеличение значения базовой угловой позиции
    function addBaseAngle(addAngle:variant):TEv3GyroSensor;

    // Установка-получение коэффициентов смещения
    function setDriftRate(    drifttype:TEv3GyroDrift;
                              driftValue:double):TEv3GyroSensor;
    function driftRate(drifttype:TEv3GyroDrift):double;
    property axisDriftRate:double;
    property integralDriftRate:double;

    // Получение актуальной коррекции смещения указанного типа
    function Correction(drifttype:TEv3GyroDrift):double;
    property axisCorrection:double;
    property integralCorrection:double;

    // калибровка гироскопа, должна проводится в состоянии покоя,
    // действует только на угол/скорость поворота,
    // не оказывает влияния на угол/скорость наклона
    function calibrate:TEv3GyroSensor;

    // угол поворота в градусах
    // если src=true, то результат возвращается непосредственно с гироскопа
    // если src=false, то результат возвращается с учетом
    // базового угла и всех коррекций смещения
    function angle(src:boolean=false):variant;
    function angle(out ang:variant;src:boolean=false):TEv3GyroSensor;
  end;
```

Примеры использования датчика-гироскопа EV3

```
{
 Демонстрация датчика гироскопа (sns4.pp)
 При старте программы робот должен находиться в состоянии покоя.
 Калибровкой датчика гироскопа фиксируется первоначальное
 направление робота.
 Если направление робота изменяется (например - ручным перемещением
 в другое положение), то выждав 5-секундний
 интервал, робот плавно возвращается в исходное положение
 с помощью вращения вокруг своей оси.

 Требования:
 Два больших мотора, левый подключен к
 порту с меньшим номером, правый - к порту
 с большим номером.
 Датчик гироскопа, подключен к любому порту.

}
{$i rubiro.inc}
uses rubiroBase,rubiroSensors,rubiroMotors,math;
```

```
 var Rule:TEv3Rule;
     gyr:TEv3GyroSensor;
     angle:integer;
begin
  ev3Init();
  Rule:=Tev3Rule.create();
  gyr:=TEv3GyroSensor.create();
  gyr.calibrate;
  gyr.timerStart();
  while true do begin
    angle:=gyr.angle;
    if (angle>-10)and(angle<10) then begin
     Rule.Stop();
     gyr.timerStart();
    end else begin
     if gyr.timer()<5000 then continue;
     Rule.Run(-100*sign(angle),10+sqrt(abs(angle)));
    end;
  end;
end.
```

Класс TEv3UltraSensor (модуль rubiroSensors)

Датчик расстояния работает по принципу эхолокатора на частоте в 40 килогерц и позволяет определять расстояние до впереди стоящего препятствия. Следует учитывать конусообразное (90 градусов) распространение сигнала, побочным эффектом которого является возможный "захват" препятствий, находящихся за пределами осевой линии его распространения. Диапазон измеряемого расстояния - от 3 до 255 сантиметров, с точностью до десятых долей. Датчик может определять наличие шумов, т. е. сигналов от других аналогичных датчиков или устройств. В таком случае полагаться на полученный результат нельзя и следует повторить замеры расстояния. Датчик может функционировать в двух режимах: в режиме непрерывного замера и режиме единичного замера. В первом случае датчик постоянно посылает звуковой и сигнал и определяет расстояние до препятствия. Во втором случае датчик осуществляет единичный замер в момент обращения к нему.

Факт и точность определения расстояния до препятствия зависит от многих параметров. Во первых, точность замера, в соответствии с технической документацией, равна ±1 сантиметру. Во вторых, при замере расстояния до препятствия, расположенного ближе 3см, возвращаемый результат будет произвольным. В третьих, возвращаемое расстояние в 255 сантиметров означает, что препятствие не удалось обнаружить. Таким образом, значение 255 - это не расстояние, а индикатор отсутствия препятствия. В четвертых, если присутствуют аудиосигналы на близкой к 40кГц частоте, они могут повлиять на результат замера. Эту проблему может частично решить механизм определения шумов. Если перед выполнением замера расстояния определить отсутствие посторонних шумов, то вероятность корректности замера значительно повысится. В пятых, эксперименты показывают возможность появления так называемых "фантомов", когда в отсутствие препятствия датчик определяет его наличие. В шестых, точность замера во многом зависит от формы препятствия: наилучший результат дают предметы с плоской стороной, обращенной к датчику; цилиндрические или сферические предметы дают худший результат, особенно при смещении в сторону от оси; наихудший результат дают смещенные относительно оси предметы с плоскими сторонами, не перпендикулярными оси датчика. В совокупности, все указанные факторы определяют невозможность гарантировать точность

отдельного замера, и корректный результат может быть получен только с использованием статистических методов.

Описание класса TEv3UltraSensor

```
  TEv3UltraSensor=class(TEv3Sensor)
     constructor create(port:variant);
     constructor create();

     {
      Включает (1/true) или отключает (0/false) режим ожидания
      нового расстояния. В этом режиме получение расстояния до
      препятствия блокирует программу до тех пор, пока оно не
      изменится по сравнения с предыдущим замером.
     }
     function waitMode(yes:variant):TEv3UltraSensor;

     {
      Включает (1/true) или отключает (0/false) режим единичного
      чтения, с интервалом запросов не менее 300ms. Полезен при
      наличии нескольких датчиков во избежание помех. В текущей
      версии библиотеки НЕ РЕКОМЕНДУЕТСЯ к использованию по причине
      нестабильной работы драйвера
     }
     function singleMode(yes:variant):TEv3UltraSensor;

     // определение помех, возвращает истину, если рядом другой мешающий сенсор
     function noise:variant;
     function noise(out yesnoise:variant):TEv3UltraSensor;

     // расстояние до препятствия в миллиметрах
     function lenMM:variant;
     function lenMM(out len:variant):TEv3UltraSensor;

     // расстояние до препятствия в сантиметрах
     function lenCM:variant;
     function lenCM(out len:variant):TEv3UltraSensor;

     // расстояние до препятствия в метрах
     function lenM:variant;
     function lenM(out len:variant):TEv3UltraSensor;
  end;
```

Примеры использования ультразвукового датчика расстояния EV3

```
{
 Демонстрация работы ультразвукового датчика расстояния (sns7.pp)
 "Робот-путешественник"

 Требования:
 Два больших мотора, левый подключен к
 порту с меньшим номером, правый - к порту
 с большим номером.
 Ультразвуковой датчик расстояния, подключен к любому порту.
}
{$i rubiro.inc}
uses rubiroBase,rubiroSensors,rubiroMotors,rubiroButtons;
```

```
  var u:TEv3UltraSensor;
      r:TEv3Rule;
      l:integer;
begin
  ev3Init();
  u:=TEv3UltraSensor.Create;
  r:=TEv3Rule.Create;
  while true do begin
   if ev3buttons.back then break;
   L:=u.lenCM;
   if L>50 then
    r.Run(0,30)
   else
    r.RunTurnWait(100-200*random(2),0.2+random/2,-15);
  end;
end.
```

Класс TEv3ColorSensor (модуль rubiroSensors)

Указанный датчик является многопрофильным устройством и может функционировать в четырех различных режимах. На работу датчика сильное влияние оказывает изменение внешней освещенности и крайне негативное - работа фотовспышек. Поэтому следует минимизировать использование фотокамер при работе с датчиком цвета/света.

Режим определения цвета (COLOR mode) определяет цвет поверхности, находящейся на расстоянии 0.5-1.0 сантиметров от датчика. Результат измерения возвращается в виде целочисленного значения в диапазоне [0,7], где 0 - отсутствие или некорректное определение цвета, 1 - черный, 2 - синий, 3 - зеленый, 4 - желтый, 5 - красный, 6 - белый, 7 - коричневый. К сожалению, компания LEGO не предоставляет идентификацию цветов в какой-либо известной цветовой модели (Pantone, CMYK, RGB и т.п.), поэтому невозможно гарантировать 100-процентное корректное распознавание цвета с использованием указанного датчика. Из практики использования следует, что лучше всего распознается красный цвет, друг с другом путаются 1) желтый, белый и коричневый, 2) черный, синий и зеленый. Корректность распознавания зависит от оттенка цвета; от расстояния, на котором находится датчик от цветной поверхности; от формы цветной поверхности; от однородности цвета поверхности; от внешней освещенности. Можно обеспечить индивидуальную подгонку всех указанных параметров таким образом, чтобы получить гарантированную корректность распознавания, однако перемещение робота в другие условия снова приведет к путанице цветов. Поэтому, при возможности, следует использовать не более чем трехцветовую схему, например, синий-красный-желтый, зеленый-красный-белый и т. д.

Режим определения RGB-цвета (RGB-REF mode) определяет RGB-цвет поверхности, находящейся на расстоянии 0.5-1.0 сантиметров от датчика. В данном режиме датчик возвращает три целочисленных значения, каждое - в диапазоне от 0 до 1020. Значения представляют собой интенсивность красного, зеленого и синего. Анализируя комбинацию указанных значений, можно работать с гораздо большим количеством оттенков цветов, чем предоставляет предыдущий режим.

Режим определения интенсивности отраженного света (REFLECT mode). Данный режим обычно используют для определения черного и белого цветов, а также различных оттенков серого. Расположение датчика предполагается на расстоянии 0.5-2см над поверхностью. От черной поверхности отражение минимально, от белой - максимально.

Датчик возвращает целочисленное значение от 0 (минимальное отражение) до 100 (максимальное отражение). Коррективы в результат, возвращаемый датчиком, вносит внешняя освещенность, поэтому не рекомендуется применять константные значения для определения границ белого или черного цвета. Вместо этого рекомендуется использовать переменные значения, модифицируемые для каждого конкретного случая.

Режим определения внешней освещенности (AMBIENT mode). В соответствие с документацией датчик возвращает целочисленное значение от 0 (минимальная освещенность) до 100 (максимальная освещенность). В реальных условиях значение освещенности варьируется в более узких пределах, с разностью в 30-40 единиц между минимальным и максимальным уровнями освещения. Это единственный режим, способный определять интенсивность свечения дисплея монитора, планшета или мобильного телефона, что позволяет использовать его для ввода большого объема закодированных данных. Для этого достаточно разработать программу, которая будет чередовать в определенной области дисплея черный и белый цвет с заданной частотой, используя тот или иной алфавит кодирования (азбуку Морзе, двоичный код и т.п.), а EV3, с использованием датчика цвета/света в режиме AMBIENT, будет считывать и декодировать указанные данные.

Описание класса TEv3ColorSensor

```
TEv3ColorSensor=class(TEv3Sensor)
   constructor create(port:variant);
   constructor create();
   {
    Включает (1/true) или отключает (0/false) режим ожидания
    нового значения цвета/света. В этом режиме получение значения цвета/света
    блокирует программу до тех пор, пока оно не
    изменится по сравнения с предыдущим замером. По умолчанию
    отключено.
   }
   function waitMode(yes:variant):TEv3ColorSensor;

   // получение цвета от 0 до 7
   function color:variant;
   function color(out col:variant):TEv3ColorSensor;

   // получение RGB цветов, каждый в диапазоне от 0 до 255
   function color(out colRed,colGreen,colBlue:variant):TEv3ColorSensor;
   function colorRed:variant;
   function colorRed(out colRed:variant):TEv3ColorSensor;
   function colorGreen:variant;
   function colorGreen(out colGreen:variant):TEv3ColorSensor;
   function colorBlue:variant;
   function colorBlue(out colBlue:variant):TEv3ColorSensor;

   // получение яркости отраженного света, от 0 до 100
   function reflect:variant;
   function reflect(out ref:variant):TEv3ColorSensor;

   // получение яркости освещения, от 0 до 100
   function ambient:variant;
   function ambient(out amb:variant):TEv3ColorSensor;
end;
```

Примеры использования датчика света EV3

```
{
 Демонстрация работы датчика света (sns1.pp) со звуковой поддержкой.
 Частота звуковых сигналов (от 300 до 900 Гц) зависит от уровня
 отраженного света. Длительность звуковых сигналов увеличена в 10 раз
 при минимальном и максимальном значениях уровня отраженного света.

 Требования:
 Датчик света, подключен к любому порту.
}
{$i rubiro.inc}
uses rubiroBase, rubiroSound, rubiroSensors, rubiroButtons;
var
 s:TEv3ColorSensor;
 r,t:integer;
begin
 ev3init();
 s:=TEv3ColorSensor.Create;
 ev3sound.volume(10);
 while true do begin
   r:=s.reflect;
   if r in [0,100] then t:=100
   else t:=10;
   ev3Sound.beep(300+6*r,t).wait();
   if ev3buttons.back then exit;
 end;
end.
```

```
{
 Демонстрация работы датчика света (sns2.pp) с использованием
 синтезатора речи и выводом на дисплей.

 Требования:
 Датчик света, подключен к любому порту.
}
{$i rubiro.inc}
uses rubiroBase, rubiroSound, rubiroSensors, rubiroButtons, rubiroScreen;
var
 s:TEv3ColorSensor;
 r:integer;
begin
 ev3init();
 s:=TEv3ColorSensor.Create;
 ev3sound.volume(100);
 ev3screen.fontSize(96);
 while true do begin
   r:=s.reflect;
   if r in [0..9] then
    ev3screen.clear.print(50,100,r).show
   else if r=100 then
    ev3screen.clear.fontSize(72).print(0,90,r).show.fontSize(96)
   else
    ev3screen.clear.print(10,100,r).show;
   ev3Sound.speak(r).wait();
   if ev3buttons.back then exit;
 end;
end.
```

Класс TNXTLightSensor (модуль rubiroSensors)

Датчик света NXT, в отличии от датчика цвета EV3, возвращает вещественный результат оценки яркости отраженного света и освещенности, с точностью до одного знака после запятой.

Описание класса TNXTLightSensor

```
  TNXTLightSensor=class(TEv3Sensor)
      // получение яркости отраженного света, от 0 до 100, вещественное значение
     function reflect:variant;
     function reflect(out ref:variant):TNXTLightSensor;
     // получение яркости освещения, от 0 до 100, вещественное значение
     function ambient:variant;
     function ambient(out amb:variant):TNXTLightSensor;
  end;
```

Класс TEv3TouchSensor (модуль rubiroSensors)

Датчик-кнопка может находиться в нажатом или отпущенном состояниях. Программное обеспечение контроллера EV3 позволяет, кроме указанных состояний, определить и "щелчок" кнопки, то есть быструю последовательность нажатия и отпускания.

Описание класса TEv3TouchSensor

```
  TEv3TouchSensor=class(TEv3Sensor)
    constructor create(port:variant);
    constructor create();

    // Истина при нажатой кнопке
    function pressed:boolean;
    function pressed(out yes:boolean):TEv3TouchSensor;

    // Ожидание нажатия кнопки
    function waitPress:TEv3TouchSensor;

    // Ожидание отпускания кнопки
    function waitRelease:TEv3TouchSensor;

    // Ожидание щелчка кнопки
    function waitClick:TEv3TouchSensor;
  end;
```

Примеры использования датчика-кнопки EV3

```
{
 Демонстрация датчика-кнопки со звуковым сопровождением
 (sns3.pp)

 Требования:
 Датчик-кнопка, подключен к любому порту.
}
{$i rubiro.inc}
```

```
uses rubiroBase, rubiroSensors, rubiroSound;
var touch:TEv3TouchSensor;
begin
  ev3Init();
  touch:=TEv3TouchSensor.Create;
  while true do begin
   touch.waitpress;
   ev3Sound.beep(200,5000);
   touch.waitrelease;
   ev3Sound.stop;
  end;
end.
```

Класс TNXTTouchSensor (модуль rubiroSensors)

Возможности указанного датчика полностью соответствуют возможностям датчика-кнопки EV3.

Описание класса TNXTTouchSensor

```
  TNXTTouchSensor=class(TEv3Sensor)
    constructor create(port:variant);
    constructor create();

    // Истина при нажатой кнопке
    function pressed:boolean;
    function pressed(out yes:boolean):TNXTTouchSensor;

    // Ожидание нажатия кнопки
    function waitPress:TNXTTouchSensor;

    // Ожидание отпускания кнопки
    function waitRelease:TNXTTouchSensor;

    // Ожидание щелчка кнопки
    function waitClick:TNXTTouchSensor;
  end;
```

Примеры использования датчика-кнопки NXT

```
{
 Демонстрация работы датчика-кнопки NXT (snsNXTT.pp)

 Требования:
 Подключенный датчик-кнопка NXT
}

{$i rubiro.inc}
uses rubiroBase, rubiroSensors, rubiroButtons;
var
 s:TNXTTouchSensor;
begin
 ev3init();
 s:=TNxtTouchSensor.Create();
 while not ev3buttons.back do begin
  s.waitpress;
  writeln('pressed');
```

```
  s.waitrelease;
  writeln('released');
 end;
end.
```