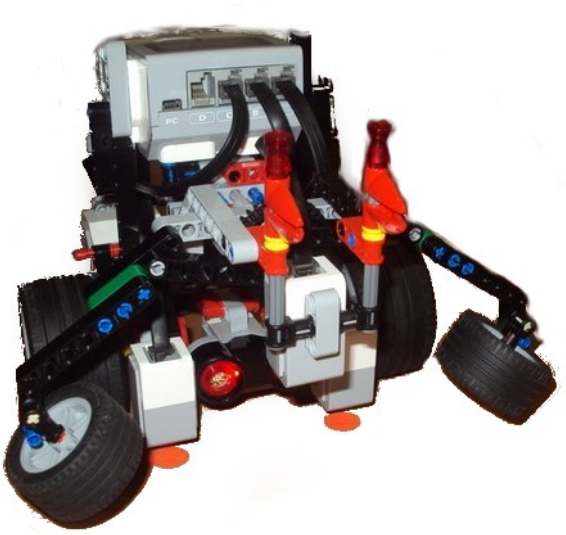
Библиотека RubiRobot

управление роботами на базе контроллера Lego EV3

Версия 0.1



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

Оглавление

Введение

Библиотека предназначена для управления роботами на базе контроллера Lego EV3 (в дальнейшем - EV3), содержит набор классов и объектов для доступа ко всем подсистемам EV3. В состав библиотеки входят более 15 модулей, более 20 демонстрационных примеров разной степени сложности, документация, файлы COPYING.LESSER и COPYING.LESSER.EN с текстом лицензии LGPLv3 на русском и английском языках, файлы README и VERSION.

Для использования библиотеки требуется загрузка EV3 с дистрибутива Linux Debian проекта ev3dev (http://www.ev3dev.org), установка дополнительных пакетов **fpc, gcc, ttf-dejavu**. Для русификации дистрибутива также рекомендуется установка пакета **locales-all** и выбор **ru_RU.UTF-8** в качестве основной локали командой **dpkg-reconfigure locales**. Саму библиотеку рекомендуется разместить в отдельном каталоге и добавить его в пути поиска модулей в конфигурационном файле FreePascal.

Автор библиотеки RubiRobot - Слинкин Д.А. (xdsl@list.ru). Идея создания библиотеки появилась во время подготовки школьников к робототехническим соревнованиям в конце 2016 - начале 2017 года. Причиной послужил тот факт, что программное обеспечение EV3 от компании Lego совершенно не предназначено для разработки сложных и объемных программ, хотя, благодаря графическому языку программирования, имеет низкий порог вхождения и может использоваться школьниками и студентами всех ступеней обучения.

Другие средства разработки предполагают использование сторонних прошивок и/или таких языков программирования как C, C++, Java, Python и некоторых других. В то-же время, в РФ для обучения программированию в большинстве школ используются разные диалекты языка программирования Pascal (Delphi, PascalABC, FreePascal и т. п.). До настоящего времени не существовало средств разработки на языке Pascal для EV3, что затрудняло применение школьниками полученных на уроках информатики знаний и умений для программирования робототехнических систем.

В то-же время, активно развивающийся проект ev3dev позволяет осуществлять загрузку на EV3 операционной системы Linux Debian Jessie, в репозитариях которой имеется компилятор FreePascal, что дает возможность создания эффективного инструментария для программирования роботов EV3 с использованием этого языка.

Разработка библиотеки RubiRobot 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.

Особенности библиотеки

1. Программировать робота с помощью библиотеки можно непосредственно на EV3, в ssh-сессии, с использованием любого текстового редактора (например - редактора файлового менеджера Midnight Commander) 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 программ с использованием всех возможностей библиотеки. **На скорость запуска и функционирования программ способ разработки не влияет.**

2. Если на хостовом компьютере используется ОС Linux, рекомендуется применять кросскомпиляцию при программировании EV3. Это резко повысит скорость разработки и снизит нагрузку на EV3. Документацию по настройке хостового компьютера на базе ОС Linux для использования кросскомпиляции планируется включить в следующую версию библиотеки.

3. В библиотеке применяется тип Variant для большинства параметров процедур, функций и методов классов, что позволяет по разному формировать и интерпретировать данные. Обычно это строки и числа разных типов. Дополнительно, обязательный к подключению модуль uev3 содержит 286 переменных типа Variant, с именами a, a0, a1..a9, b, b0, b1..b9, ... z, z0, z1..z9. Это позволяет быстро создавать программы, не отвлекаясь на объявление переменных. Исключения составляют объекты подключаемых устройств (датчики и моторы), которые в обязательном порядке следует объявлять, т. к. тип Variant несовместим с объектным типом.

4. Большинство методов классов в качестве результата возвращает ссылку на текущий объект. Это позволяет осуществлять последовательный вызов методов объекта без повторного указания объекта. Например:

```
var LM: TEv3LargeMotor;
begin
 ev3init();
 LM:=TEv3LargeMotor.Create;
 ...
 LM.setReverse(true).run(100).wait();
 ...
end.
```

5. 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 (uev3sysfs.getInterval) и запись данных (uev3sysfs.setInterval). По умолчанию они равны соответственно 10 и 20 миллисекунд, но могут быть изменены как глобально, так и индивидуально для каждого класса или объекта. Для подавляющего большинства ситуаций

таймаут на чтение достаточен, однако таймаут на запись в некоторых случаях придется увеличить.

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).

Дорожная карта по развитию инфраструктуры библиотеки Rubirobot

1. На основе документации ev3dev разработать русскоязычную документацию по подготовке SD-карты, инсталляции, подключению, обновлению ПО, доустановке требуемых пакетов, русификации.
2. Подготовить документацию по разработке программ на EV3 в SSH-сессии
3. Подготовить документацию по кросскомпиляции.
4. Готовить и периодически публиковать образ SD-карты с необходимым ПО для использования библиотеки rubirobot и архив библиотек для кросскомпиляции

Дорожная карта по развитию кода библиотеки Rubirobot

Модуль rubiroSound:

1. Возможность генерации аудио-файлов
2. Создание очереди звуковых запросов
3. Поддержка мультиязычности

Модуль rubiroScreen:

1. Уменьшение зависимости от fcl-image при сохранении неизменной интерфейсной части.
2. Поддержка нескольких шрифтов, возможность указывать их местоположение в файловой системе.

Модули rubiroMotors и rubiroSensors:

1. Корректировка формулы поворота и способа остановки в TEv3Rule для максимального соответствия программному обеспечению Lego.
2. Поддержка рулевого управления для средних моторов.
3. Реализация режима ожидания (waitMode) для инфракрасного датчика.
4. 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.

Обнаруженные ошибки в библиотеке Rubirobot

Модули uev3Devices, uev3sysfs
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.

Модуль uev3Screen
Стили кисти функционируют некорректно при переходе на freepascal версии 3.0.2

Модули библиотеки

Модули библиотеки можно условно разделить на 3 части:

1. Базовый модуль uev3. Любая программа, использующая библиотеку, должна подключить как минимум модуль uev3 и первым действием вызвать функцию инициализации библиотеки ev3init(). Без этого обращение к библиотеке будет завершаться ошибкой.

2. Модули поддержки 6 основных подсистем EV3.

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

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
uev3const, uev3func, uev3utf8	Общеупотребительные константы, переменные, функции

Краткое описание классов и объектов библиотеки

Библиотека предоставляет для программисту набор классов и несколько предопределенных объектов.

Предопределенные объекты

Предопределенные объекты создаются вызовом процедуры ev3init() из модуля uev3.

Объект	Класс	Модуль	Краткое описание
ev3	TEv3	uev3	Может использоваться для ожидания возникновения событий на других объектах. Применяется редко. Необходимость в создании других экземпляров отсутствует.
ev3buttons	TEv3Buttons	rubiroButtons	Обеспечивает обработку кнопок EV3. Используется повсеместно. Необходимость в создании других экземпляров отсутствует.
ev3leds	TEv3Leds	rubiroLeds	Обеспечивает обработку цветоиндикации EV3.

Объект	Класс	Модуль	Краткое описание
			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.
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.

Объекты подключаемых устройств (моторов и датчиков) должны создаваться и инициализироваться программистом самостоятельно, для них отсутствуют предопределенные объекты.

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.
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

Класс	Модуль	Краткое описание
		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 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.

Описание часто используемых классов и объектов

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.

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 отпускание кнопки. Кнопка кодируется событием вида
     '14:0','назад:нет'(отпускание клавиши Back), '108:1','Down:NO'
     (нажатие клавиши Down) и т.п., регистр символов неважен.
     timeout - максимальный таймаут, возвращает false при выходе по таймауту
    }
```

```
      function waitEvent(event:string; timeout:dword=high(dword)):boolean;
      override;

      {
       блокированно ждет нажатия или отпускания клавиши
       в Button возвращает клавишу - 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;

      {
       ждать нажатия-отпускания набора кнопок
       завершение - при возникновении ВСЕХ событий из переданного массива
       каждая кнопка в массиве кодируется строкой (см.WaitEvent)
       процессе 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;

      // возвращает состояние кнопки CENTER, true - при нажатии
      property Center:variant read getCenter;

      // возвращает состояние кнопки UP, true - при нажатии
      property Up:variant read getUp;

      // возвращает состояние кнопки LEFT, true - при нажатии
      property Left:variant read getLeft;

      // возвращает состояние кнопки RIGHT, true - при нажатии
      property Right:variant read getRight;

      // возвращает состояние кнопки DOWN, true - при нажатии
      property Down:variant read getDown;
    end;
```

Примеры использования кнопок EV3

```
{
 Вывод на экран нажимаемых кнопок EV3 (btn1.pp):
}
{$mode objfpc}
uses uev3,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)
}
{$mode objfpc}
uses uev3,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)
}
{$mode objfpc}
uses uev3,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.
```

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.

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)
}
{$mode objfpc}
uses uev3,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
}
{$mode objfpc}
uses uev3,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().

EV3 имеет монохромный дисплей 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. Изначально дисплей залит белым цветом, настройки цвета карандаша и кисти - черные.

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

Текст выводится ttf-шрифтом DejaVuSans.ttf, который должен быть предварительно установлен на EV3 (например - установкой пакета ttf-dejavu).

Описание класса TEv3Screen

```
  TEv3Screen=class(TEv3)
      constructor create();
      destructor destroy; override;

      // очистка изображения
      function clear:TEv3Screen;

      // вывод изображения на дисплей
      function show:TEv3Screen;

      // запись изображения на диск в png-формате
      function save(name:variant):TEv3Screen;

      // чтение изображения с диска в png-формате
      function load(name:variant):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;

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

      // установка параметров шрифта
      function fontBig():TEv3Screen;
      function fontSmall():TEv3Screen;
      function fontRotate(angle:variant):TEv3Screen;
      function fontSize(size:variant):TEv3Screen;
      function fontStd():TEv3Screen;
      function fontBlack():TEv3Screen;
      function fontWhite():TEv3Screen;

     // 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):
}
{$mode objfpc}
uses uev3,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)
}
{$mode objfpc}
uses uev3,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)
}
{$mode objfpc}
uses uev3,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)
}
{$mode objfpc}
```

```
uses uev3,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)
}
{$mode objfpc}
uses uev3,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.

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, 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)
}
{$mode objfpc}
uses uev3,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)
}
{$mode objfpc}
uses uev3,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)
}
{$mode objfpc}
uses uev3,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.
```

Моторы Lego EV3, модуль rubiroMotors

В текущей версии библиотеки поддерживается два Lego-мотора - большой и средний. Отличия между 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.
3. Поддержка реверса.
4. Перед запуском и при запуске задание скорости и одного из трех способов остановки.
5. Остановка мотора одним из трех способов остановки.
6. Ожидание остановки мотора.
7. Получение заданной и текущей скорости, позиции мотора, способа остановки.

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.

Описание классов TEv3TachoMotor, TEv3MediumMotor, TEv3LargeMotor

```
TEv3TachoMotor=class(TEv3Motor)

    // ждет остановки мотора
    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;

    {
      Устанавливает и возвращает текущую позицию мотора
    }
    function setPosition(p:variant):TEv3TachoMotor;
    function position(out p:variant):TEv3TachoMotor;
    function position:variant;

    {
      Устанавливает скорость вращения от -100% до 100%
      Действует при запуске мотора.
      На текущее вращение мотора влияния не оказывает.
    }
    function setSpeed(sp:variant):TEv3TachoMotor;
    {
      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;
```

```
    {
      Устанавливает реверс
      Действует при запуске мотора.
      На текущее вращение мотора влияния не оказывает.
    }
    function setReverse(yes:variant):TEv3TachoMotor;
    {
      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;

    {
      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 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;
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.
}
{$mode objfpc}
uses uev3,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. В текущей версии библиотеки поддерживается рулевое управление на больших моторах, однако в дорожной карте развития библиотеки есть планы поддержки средних моторов.

Класс **TEv3Rule** предлагает следующие возможности:

1. Поддержка двух больших 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. Поддержка реверса.
6. Перед запуском и при запуске: задание скорости и одного из трех способов остановки.
7. Ожидание остановки моторов.
8. 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.

Описание класса TEv3Rule

```
TEv3Rule=class

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

    constructor create(LPort,RPort:variant);
    constructor create();
    destructor destroy;override;

    // 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;

    // устанавливает и возвращает реверс (см.TEv3TachoMotor)
```

```
    function setReverse(yes:variant):TEv3Rule;
    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;
  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.
}
{$mode objfpc}
uses uev3,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.
```

Датчики Lego EV3, модуль rubiroSensors

В текущей версии библиотеки поддерживается весь набор датчиков Lego-EV3

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.

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. При этом, 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.
}
{$mode objfpc}
uses uev3,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.
}
{$mode objfpc}
uses uev3,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

Использование датчика-гироскопа возможно только при его подключении или калибровке в момент, когда робот находится в состоянии покоя. В противном случае данные датчик будет возвращать некорректные данные.

Описание класса TEv3GyroSensor

```
  TEv3GyroSensor=class(TEv3Sensor)
    constructor create(port:variant);
    constructor create();
    // калибровка гироскопа, должна проводится в состоянии покоя
    function calibrate:TEv3GyroSensor;
    // угол поворота в градусах
    function angle:variant;
    function angle(out ang:variant):TEv3GyroSensor;
    // скорость поворота
    function speed:variant;
    function speed(out sp:variant):TEv3GyroSensor;
    // угол наклона в градусах
    function tiltAngle:variant;
    function tiltAngle(out ang:variant):TEv3GyroSensor;
    // скорость наклона
    function tiltSpeed:variant;
    function tiltSpeed(out sp:variant):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.

}
{$mode objfpc}
uses sysutils,dateutils,uev3,uev3Devices,rubiroSensors,rubiroMotors,math;
 var Rule:TEv3Rule;
     gyr:TEv3GyroSensor;
     dt:TDateTime;
     angle:integer;
begin
  ev3Init();
  Rule:=Tev3Rule.create();
  gyr:=TEv3GyroSensor.create();
  gyr.calibrate;
  dt:=now();
  while true do begin
    angle:=gyr.angle;
    if (angle>-10)and(angle<10) then begin
     Rule.Stop();
     dt:=now();
    end else begin
     if MilliSecondsBetween(now(),dt)<5000 then continue;
     Rule.Run(-100*sign(angle),10+sqrt(abs(angle)));
    end;
  end;
end.
```

Класс TEv3UltraSensor, модуль rubiroSensors

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.
}
{$mode objfpc}
uses uev3,rubiroSensors,rubiroMotors,rubiroButtons;
  var u:TEv3UltraSensor;
      r:TEv3Rule;
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

Описание класса TEv3ColorSensor

```
  TEv3ColorSensor=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. 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.
}
{$mode objfpc}
uses uev3, 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.
}
{$mode objfpc}
uses uev3, 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 TEv3TouchSensor, модуль rubiroSensors

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.
}
uses uev3, 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.
```