# iRobot Create library Reference Manual

Generated by Doxygen 1.5.4

Thu Aug 28 13:54:20 2008

# Contents

# Chapter 1

# iRobot Create library Main Page

**Important: this library is not supported by iRobot (p. 13).**

## 1.1 Introduction

This library allows to control easily the **iRobot** (p. 13) Create robot. It implements the `Open Interface version 2`.

Using this library also ensures that the commands sent to the robot will be valid (if an uncompleted command is sent, the robot will remain in a "waiting" state). It also prevents you from sending invalid and malformed commands (all arguments are checked). Basically, if a command is successfully sent (no exception has been thrown), you can be sure that the command is valid and will be correctly executed.

The library also provides a set of types to avoid using "magic constants" in your code and make it more readable.

## 1.2 Tutorial

**iRobot** (p. 13) Create uses a specific protocol which is implemented by this library. It is called Open Interface (OI) version 2. When instantiating the robot class (**iRobot::Create** (p. 29)) you have to indicate which stream is used to communicate with the robot.

Typically you probably want to use a serial port device for the communication such as:

- `/dev/ttySX` (where X is the serial port number),
- `/dev/rfcommX` (where X is the serial-over-bluetooth port number).

The last solution requires the Bluetooth adapter module (BAM).

To communicate with the robot, you **need** the serial port support. To enable this, you have to use the `libserial library` (this library has to be installed on your computer).

Here is a complete example using the **LibSerial** (p. 15) library:

```
#include <SerialStream.h>
#include <irobot-create.hh>
```

```
int main (int argc, char** argv)
{
  using namespace iRobot;
  using namespace LibSerial;


  if (argc < 2)
    return 1;
  SerialStream stream (argv[1]);

  try
    {
      Create robot (stream);

      // Swith to full mode.
      robot.sendFullCommand ();

      // Let's stream some sensors.
      Create::sensorPackets_t sensors;
      sensors.push_back (Create::SENSOR_BUMPS_WHEELS_DROPS);
      sensors.push_back (Create::SENSOR_WALL);
      sensors.push_back (Create::SENSOR_BUTTONS);

      robot.sendStreamCommand (sensors);

      // Let's turn!
      int speed = 200;
      int ledColor = Create::LED_COLOR_GREEN;
      robot.sendDriveCommand (speed, Create::DRIVE_INPLACE_CLOCKWISE);
      robot.sendLedCommand (Create::LED_PLAY, 0, 0);

      while (!robot.playButton ())
        {
          if (robot.bumpLeft () || robot.bumpRight ())
            std::cout << "Bump !" << std::endl;
          if (robot.wall ())
            std::cout << "Wall !" << std::endl;
          if (robot.advanceButton ())
            {
              speed = -1 * speed;
              ledColor += 10;
              if (ledColor > 255)
                ledColor = 0;

              robot.sendDriveCommand (speed, Create::DRIVE_INPLACE_CLOCKWISE);
              if (speed < 0)
                robot.sendLedCommand (Create::LED_PLAY,
                                      ledColor,
                                      Create::LED_INTENSITY_FULL);
              else
                robot.sendLedCommand (Create::LED_ADVANCE,
                                      ledColor,
                                      Create::LED_INTENSITY_FULL);
            }

          // You can add more commands here.
          usleep(100 * 1000);
        }

      robot.sendDriveCommand (0, Create::DRIVE_STRAIGHT);
    }
  catch (InvalidArgument& e)
    {
      std::cerr << e.what () << std::endl;
      return 3;
    }
```

```
  catch (CommandNotAvailable& e)
    {
      std::cerr << e.what () << std::endl;
      return 4;
    }
}
```

The robot will start to turn clockwise, then if you push the advance button, it will change its direction. Each time you're pushing the button, the LEDs will also change.

## 1.3 What if I do not want to use LibSerial?

You can construct the object with any kind of std::iostream, it means that it can come from another serial library, it may be a std::stringstream or a std::fstream. However, do **not** use a fstream to open a serial port device (see FAQ).

Here is a simple example that displays on the standard output the binary command used to start the "cover" demo (warning: this will display non-ascii characters to your term if you do not redirect the standard output, you may want to use a tool like hexdump to see the result).

```
#include <sstream>
#include <irobot-create.hh>

int main ()
{
   using namespace iRobot;

   std::stringstream ss;
   Create robot (ss);
   robot.sendDemoCommand (Create::DEMO_COVER);
   std::cout << ss.str ();
}
```

## 1.4 Frequently Asked Questions (FAQ)

### 1.4.1 Why is the program failing when I use serial port communication?

If you try to use the serial port communication without having the **LibSerial** (p. 15) support, the constructor will fail and throw a LibSerialNotAvailable exception. It means that when the configure script has not detected the **LibSerial** (p. 15) library on your computer. Check your config.log, it should contain something like that:

```
configure:XXXXX: checking for main in -lserial
[...]
configure:XXXXX: result: yes
```

If the result is **no**, please specify the header and library path in CPPFLAGS and LDFLAGS. For instance, if you are using gcc (which is usually the case), it should be something like that:

```
./configure CPPFLAGS=-I/my/path/to/libserial LDFLAGS=-L/my/path/to/libserial
```

*Note: if you have installed a package to get this library, please ask the maintainer of the package for your distribution to enable the **LibSerial** (p. 15) support.*

### 1.4.2 Why does the communication through the serial port not work?

If you have opened the serial port through a std::fstream, **it will not work**. Always use the **LibSerial** (p. 15) library (or another 3rd party serial library). Failing to do so may lead to random errors, wrong values for the sensors, etc.

### 1.4.3 The sensor I'm using is never updated, why?

If you want an updated value of a sensor, you have to explicitly ask for an update. It can either be done with the Create::sendSensorsCommand or the Create::sendQueryListCommand command.

If you are using the sensors frequently, you probably want to use the Create::sendStreamCommand that will ask the robot to send the sensor's value each 15ms. It is the only way to keep a sensor updated automatically. If you are using a connection with poor real-time characteristics (such as a wireless network using the BAM for instance), the solution recommended by **iRobot** (p. 13) is to use the stream command.

```
iRobot::Create robot (...);

robot.wall (); // Always false, no update.
// [...]
robot.wall (); // Always false, no update.

Create::sensorPackets_t sensors;
sensors.push_back (Create::SENSOR_BUMPS_WHEELS_DROPS);

robot.sendQueryListCommand (robot);

robot.wall (); // Updated value.
robot.wall (); // *NO* update here.
robot.wall (); // *NO* update here.
// etc.

robot.sendStreamCommand (robot);

robot.wall (); // Updated value.
robot.wall (); // Updated value.
robot.wall (); // Updated value.
// etc.
```

### 1.4.4 How to use the Bluetooth adapter module (BAM)?

To use the BAM with this library, you will need two tools:

- hcitool
- rfcomm

Both tools are provided by the "bluez" package.

Install the Bluetooth adapter module and start the robot. Then, execute the following command:

```
$ hcitool scan
```

You should see something that looks like that:

```
Scanning ...
     00:0A:3A:26:49:AF       Element Serial
     [...]
```

The "Element Serial" is the **iRobot** (p. 13) Create. Remember the address as it will be required for the next step. If it fails, check that your kernel supports bluetooth (check your distribution's documentation).

Execute the following command:

```
$ rfcomm connect 1 '00:0A:3A:26:49:AF'
```

You should then see:

```
Connected /dev/rfcomm1 to 00:0A:3A:26:49:AF on channel 1
Press CTRL-C for hangup
```

You are now connected to the robot and you can read the `/dev/rfcomm1` file to communicate with it.

If the connection fails, wait for a couple of seconds and retry.

### 1.4.5 How to use the Bluetooth adapter module (BAM)?

Basically, if the library and the headers are installed in standard directories, you only have to link against the library. If you are using GCC, you probably want:

```
$ g++ myproject.cc -lirobot-create
```

If you have not installed the library in a standard directory, you have to indicate the header directory when compiling and the library directory when linking. With GCC:

```
$ g++ myproject.cc -I/my/include/path -L/my/library/path -Wl,-R/my/library/path -lirobot-create
```

The `-Wl,-R/my/library/pat` flag adds the library directory to the list used to search dynamic libraries at run-time (`-L` is only used at compile-time).

If you are using the Autotools, see the next section.

### 1.4.6 How to integrate this library to my Autotools software?

A m4 file is provided with the macros required to check for this library.

Here is a sample configure.ac that check for the library:

```
AC_INIT([test], [0.1])
AM_INIT_AUTOMAKE

AC_LANG([C++])
AC_PROG_CXX
```

```
LIBIROBOT_CREATE_ARG_WITH

AC_CONFIG_HEADERS([config.h])

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

It also define the boolean LIBIROBOT_CREATE that can be used in Automake files.

# Chapter 2

# iRobot Create library Namespace Index

## 2.1 iRobot Create library Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 3

# iRobot Create library Class Index

## 3.1 iRobot Create library Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

# iRobot Create library File Index

## 4.1 iRobot Create library File List

Here is a list of all files with brief descriptions:

# Chapter 5

# iRobot Create library Namespace Documentation

## 5.1 iRobot Namespace Reference

Main namespace.

### Classes

- class **Create**

  *Robot communication class, wraps Open Interface version 2.*

### Functions

- **IROBOT_MAKE_EXCEPTION** (CommandNotAvailable,"This command is not available in this mode.")

  *Error thrown when the command is not valid in the current mode.*

- **IROBOT_MAKE_EXCEPTION** (InvalidArgument,"This argument is invalid.")

  *Exception thrown when an argument's value is invalid. Typically, it means that an argument is out of bound.*

- **IROBOT_MAKE_EXCEPTION** (LibSerialNotAvailable,"Serial port support has not been compiled.")

  *Exception thrown if no serial port support is available but one tries to use the feature anyway.*

### 5.1.1 Detailed Description

Main namespace.

HAVE_LIBSERIAL.

The namespace gathers all the classes of this library.

### 5.1.2 Function Documentation

#### 5.1.2.1 iRobot::IROBOT_MAKE_EXCEPTION (LibSerialNotAvailable, "Serial port support has not been compiled.")

Exception thrown if no serial port support is available but one tries to use the feature anyway.

#### 5.1.2.2 iRobot::IROBOT_MAKE_EXCEPTION (InvalidArgument, "This argument is invalid.")

Exception thrown when an argument's value is invalid. Typically, it means that an argument is out of bound.

#### 5.1.2.3 iRobot::IROBOT_MAKE_EXCEPTION (CommandNotAvailable, "This command is not available in this mode.")

Error thrown when the command is not valid in the current mode.

The **Create** (p. 29) class never changes the mode behind your back, so you have to make sure that the command you want to send is supported by the current mode.

**See also:**

**Create::sendStartCommand** (p. 89)
**Create::sendSafeCommand** (p. 89)
**Create::sendFullCommand** (p. 89)

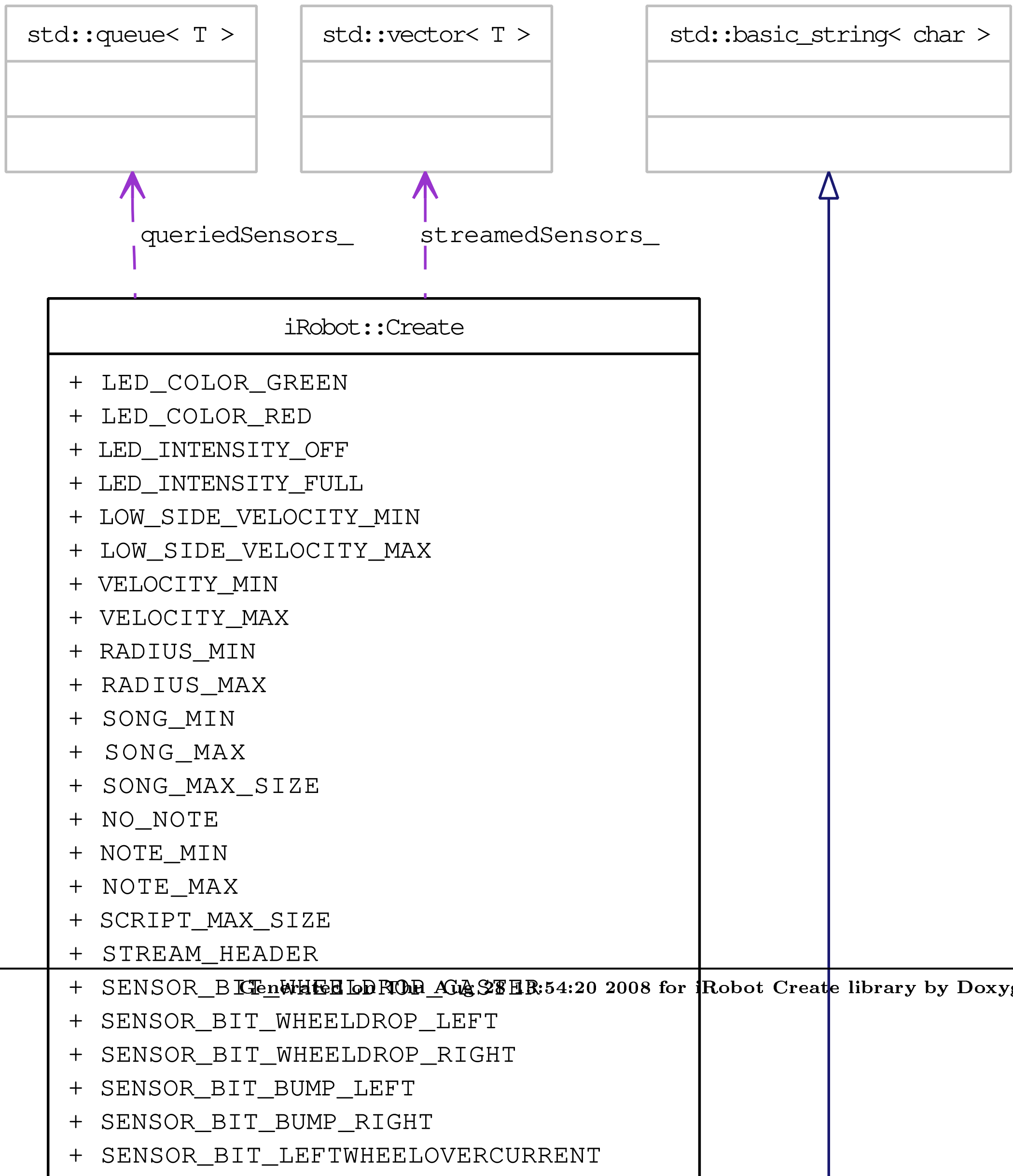
## 5.2 LibSerial Namespace Reference

# Chapter 6

# iRobot Create library Class Documentation

## 6.1 Application Struct Reference

Collaboration diagram for Application:



## Public Member Functions

- **Application** (int argc, char **argv, std::iostream &stream)
- **Create::Opcode getMode** (const std::string &m)
- void **help** (int status=0)
- void **sendScript** (int argc, char **argv)
- void **sendSong** (int argc, char **argv)
- void **sendQueryList** (int argc, char **argv)
- void **sendStream** (int argc, char **argv)
- void **showSensors** ()
- void **usage** ()
- void **version** ()
- **Application** (int argc, char **argv, std::iostream &stream)
- **Create::Opcode getMode** (const std::string &m)
- void **help** (int status=0)
- void **sendScript** (int argc, char **argv)
- void **sendSong** (int argc, char **argv)
- void **sendQueryList** (int argc, char **argv)
- void **sendStream** (int argc, char **argv)
- void **showSensors** ()
- void **usage** ()
- void **version** ()

### 6.1.1 Constructor & Destructor Documentation

#### 6.1.1.1 Application::Application (int *argc*, char ** *argv*, std::iostream & *stream*) [inline]

```
  : binaryName_ (argv[0]),
    mode_ (),
    robot_ (stream)
{
  if (argc < 2)
    usage ();
  std::string argv1 = argv[1];
  if (argv1 == "--help"
      || argv1 == "-h"
      || argv1 == "help")
    help ();
  if (argv1 == "--version"
      || argv1 == "-v"
      || argv1 == "version")
    version ();

  mode_ = getMode (argv[1]);

  try
    {
      if (mode_ == Create::OPCODE_START)
        return;
      robot_.sendSafeCommand ();
      switch (mode_)
        {
          APPLY_ON_OPCODES (SEND_CMD)
        default:
            usage ();
        }
    }
```

```
  catch (std::runtime_error& e)
    {
      std::cerr << e.what () << std::endl;
      exit (2);
    }
  catch (...)
    {
      std::cerr << "Unexpected error" << std::endl;
      exit (42);
    }
}
```

#### 6.1.1.2 Application::Application (int *argc*, char ** *argv*, std::iostream & *stream*) [inline]

```
  : binaryName_ (argv[0]),
    mode_ (),
    robot_ (stream)
{
  if (argc < 2)
    usage ();
  std::string argv1 = argv[1];
  if (argv1 == "--help"
      || argv1 == "-h"
      || argv1 == "help")
    help ();
  if (argv1 == "--version"
      || argv1 == "-v"
      || argv1 == "version")
    version ();

  mode_ = getMode (argv[1]);

  try
    {
      if (mode_ == Create::OPCODE_START)
        return;
      robot_.sendSafeCommand ();
      switch (mode_)
        {
          APPLY_ON_OPCODES (SEND_CMD)
        default:
            usage ();
        }
    }
  catch (std::runtime_error& e)
    {
      std::cerr << e.what () << std::endl;
      exit (2);
    }
  catch (...)
    {
      std::cerr << "Unexpected error" << std::endl;
      exit (42);
    }
}
```

### 6.1.2 Member Function Documentation

#### 6.1.2.1 Create::Opcode Application::getMode (const std::string & *m*) [inline]

```
{
```

```
  APPLY_ON_OPCODES (STR_TO_OPCODE);
  return static_cast<Create::Opcode> (0);
}
```

#### 6.1.2.2 void Application::help (int *status* = 0) [inline]

```
{
  std::cout
    << "Usage: irobot-create-ctl <action> [options]" << std::endl
    << std::endl
    << "Available actions are:" << std::endl;
  APPLY_ON_OPCODES (PRINT_ACTIONS);
  std::cout << std::endl;
  std::cout << "Additional actions are:" << std::endl
            << "\t" "help (this message)" << std::endl
            << "\t" "version" << std::endl;

  std::cout
    << std::endl
    << "Display the specified action on the standard output using the"
    << std::endl
    << "Open Interface protocol version 2. The stream can be redirected"
    << std::endl
    << "to a device file to send the action to a real robot." << std::endl
    << std::endl
    << "Example (serial port): irobot-create-ctl safe > /dev/ttyS0"
    << std::endl
    << "Example (bluetooth): irobot-create-ctl safe > /dev/rfcomm0"
    << std::endl;

  std::cout
    << std::endl
    << "Report bugs to <" << PACKAGE_BUGREPORT << ">" << std::endl;

  exit (status);
}
```

#### 6.1.2.3 void Application::sendScript (int *argc*, char ** *argv*) [inline]

```
{
  Create::opcodes_t script;

  for (int i = 2; i < argc; ++i)
    script.push_back (castArg<unsigned char> (argv[i]));
  robot_.sendScriptCommand (script);
}
```

#### 6.1.2.4 void Application::sendSong (int *argc*, char ** *argv*) [inline]

```
{
  Create::song_t song;
  if (argc % 2 == 0)
    usage ();

  const unsigned char sid = castArg<unsigned char> (argv[2]);
  for (int i = 3; i + 1 < argc; i += 2)
    song.push_back (std::make_pair (castArg<unsigned char> (argv[i]),
                                    castArg<unsigned char> (argv[i+1])));
  robot_.sendSongCommand (sid, song);
}
```

#### 6.1.2.5 void Application::sendQueryList (int *argc*, char ** *argv*) [inline]

```
{
  SENSOR_CMD (robot_.sendQueryListCommand (sensors));
  showSensors ();
}
```

#### 6.1.2.6 void Application::sendStream (int *argc*, char ** *argv*) [inline]

```
{
  SENSOR_CMD (robot_.sendStreamCommand (sensors));
  showSensors ();
}
```

#### 6.1.2.7 void Application::showSensors () [inline]

```
{
  std::cerr
    << "Wheeldrop: "
    << " caster: " << ((!!robot_.wheeldropCaster ()) ? "true" : "false")
    << " left wheel: " << ((!!robot_.wheeldropLeft ()) ? "true" : "false")
    << " right wheel: " << ((!!robot_.wheeldropRight ()) ? "true" : "false")
    << " bump left: " << ((!!robot_.bumpLeft ()) ? "true" : "false")
    << " bump right: " << ((!!robot_.bumpRight ()) ? "true" : "false")
    << std::endl;

  std::cerr
    << "Wall: " << ((!!robot_.wall ()) ? "true" : "false")
    << std::endl;

  std::cerr
    << "Cliff: "
    << " left: " << ((!!robot_.cliffLeft ()) ? "true" : "false")
    << " front-left: " << ((!!robot_.cliffFrontLeft ()) ? "true" : "false")
    << " front-right: " << ((!!robot_.cliffFrontRight ()) ? "true" : "false")
    << " right: " << ((!!robot_.cliffRight ()) ? "true" : "false") << std::endl;

  std::cerr
    << "Digital pin status: "
    << " device detect: " << ((!!robot_.deviceDetect ()) ? "true" : "false")
    << " 3: " << ((!!robot_.digitalInput3 ()) ? "true" : "false")
    << " 2: " << ((!!robot_.digitalInput2 ()) ? "true" : "false")
    << " 1: " << ((!!robot_.digitalInput1 ()) ? "true" : "false")
    << " 0: " << ((!!robot_.digitalInput0 ()) ? "true" : "false") << std::endl;

  std::cerr
    << "Analog signal: " << robot_.analogSignal () << std::endl;

  std::cerr
    << "Chargers available? "
    << " home base: "
    << ((!!robot_.homeBaseChargerAvailable ()) ? "true" : "false")
    << " internal: "
    << ((!!robot_.internalChargerAvailable ()) ? "true" : "false") << std::endl;

  std::cerr
    << "Virtual Wall: "
    << ((!!robot_.virtualWall ()) ? "true" : "false") << std::endl;

  std::cerr
    << "Overcurrent: "
    << " left wheel: "
```

```
        << ((!!robot_.leftWheelOvercurrent ()) ? "true" : "false")
        << " right wheel: "
        << ((!!robot_.rightWheelOvercurrent ()) ? "true" : "false")
        << " LD2: " << ((!!robot_.ld2Overcurrent ()) ? "true" : "false")
        << " LD1: " << ((!!robot_.ld1Overcurrent ()) ? "true" : "false")
        << " LD0: " << ((!!robot_.ld0Overcurrent ()) ? "true" : "false");

      std::cerr
        << "IR: " << robot_.ir () << std::endl;

      std::cerr
        << "Buttons: "
        << " advance: "<< ((!!robot_.advanceButton ()) ? "true" : "false")
        << " play: " << ((!!robot_.playButton ()) ? "true" : "false") << std::endl;

      std::cerr
        << "Distance: " << robot_.distance () << std::endl;

      std::cerr
        << "Angle: " << robot_.angle () << std::endl;

      std::cerr
        << "Charging state: " << robot_.chargingState () << std::endl;

      std::cerr
        << "Battery voltage: " << robot_.batteryVoltage () << std::endl;

      std::cerr
        << "Battery current: " << robot_.batteryCurrent () << std::endl;

      std::cerr
        << "Battery temperature: " << robot_.batteryTemperature () << std::endl;

      std::cerr
        << "Battery charge: " << robot_.batteryCharge () << std::endl;

      std::cerr
        << "Battery capacity: " << robot_.batteryCapacity () << std::endl;

      std::cerr
        << "Wall signal: " << robot_.wallSignal () << std::endl;

      std::cerr
        << "Cliff left signal: "
        << robot_.cliffLeftSignal () << std::endl;

      std::cerr
        << "Cliff front-left signal: "
        << robot_.cliffFrontLeftSignal () << std::endl;

      std::cerr
        << "Cliff front-right signal: "
        << robot_.cliffFrontRightSignal () << std::endl;

      std::cerr
        << "Cliff right signal: " << robot_.cliffRightSignal () << std::endl;

      std::cerr
        << "Song number: " << static_cast<int> (robot_.songNumber ())
        << std::endl;

      std::cerr
        << "Song playing: "
        << ((!!robot_.songPlaying ()) ? "true" : "false") << std::endl;

      std::cerr
        << "Streamed packets: " << robot_.streamPackets () << std::endl;
```

```

  std::cerr
    << "Requested velocty: "
    << robot_.requestedVelocity () << std::endl;

  std::cerr
    << "Request radius: "
    << robot_.requestedRadius () << std::endl;

  std::cerr
    << "Request left velocity: "
    << robot_.requestedLeftVelocity () << std::endl;

  std::cerr
    << "Request right velocity: "
    << robot_.requestedRightVelocity () << std::endl;

}
```

#### 6.1.2.8 void Application::usage () [inline]

```
{
  help (1);
}
```

#### 6.1.2.9 void Application::version () [inline]

```
{
  std::cerr
    << PACKAGE_STRING << std::endl
    << "Copyright (C) 2008  iLab."
    << std::endl
    << "This is free software; see the source for copying conditions."
    << std::endl
    << "There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A"
    << std::endl
    << "PARTICULAR PURPOSE." << std::endl;
  exit (0);
}
```

#### 6.1.2.10 Create::Opcode Application::getMode (const std::string & *m*) [inline]

```
{
  APPLY_ON_OPCODES (STR_TO_OPCODE);
  return static_cast<Create::Opcode> (0);
}
```

#### 6.1.2.11 void Application::help (int *status* = 0) [inline]

```
{
  std::cout
    << "Usage: irobot-create-ctl <action> [options]" << std::endl
    << std::endl
    << "Available actions are:" << std::endl;
  APPLY_ON_OPCODES (PRINT_ACTIONS);
  std::cout << std::endl;
  std::cout << "Additional actions are:" << std::endl
```

```
            << "\t" "help (this message)" << std::endl
            << "\t" "version" << std::endl;

  std::cout
    << std::endl
    << "Display the specified action on the standard output using the"
    << std::endl
    << "Open Interface protocol version 2. The stream can be redirected"
    << std::endl
    << "to a device file to send the action to a real robot." << std::endl
    << std::endl
    << "Example (serial port): irobot-create-ctl safe > /dev/ttyS0"
    << std::endl
    << "Example (bluetooth): irobot-create-ctl safe > /dev/rfcomm0"
    << std::endl;

  std::cout
    << std::endl
    << "Report bugs to <" << PACKAGE_BUGREPORT << ">" << std::endl;

  exit (status);
}
```

#### 6.1.2.12 void Application::sendScript (int *argc*, char ** *argv*) [inline]

```
{
  Create::opcodes_t script;

  for (int i = 2; i < argc; ++i)
    script.push_back (castArg<unsigned char> (argv[i]));
  robot_.sendScriptCommand (script);
}
```

#### 6.1.2.13 void Application::sendSong (int *argc*, char ** *argv*) [inline]

```
{
  Create::song_t song;
  if (argc % 2 == 0)
    usage ();

  const unsigned char sid = castArg<unsigned char> (argv[2]);
  for (int i = 3; i + 1 < argc; i += 2)
    song.push_back (std::make_pair (castArg<unsigned char> (argv[i]),
                                    castArg<unsigned char> (argv[i+1])));
  robot_.sendSongCommand (sid, song);
}
```

#### 6.1.2.14 void Application::sendQueryList (int *argc*, char ** *argv*) [inline]

```
{
  SENSOR_CMD (robot_.sendQueryListCommand (sensors));
  showSensors ();
}
```

#### 6.1.2.15 void Application::sendStream (int *argc*, char ** *argv*) [inline]

```
{
```

```
  SENSOR_CMD (robot_.sendStreamCommand (sensors));
  showSensors ();
}
```

#### 6.1.2.16 void Application::showSensors () [inline]

```
{
  std::cerr
    << "Wheeldrop: "
    << " caster: " << ((!!robot_.wheeldropCaster ()) ? "true" : "false")
    << " left wheel: " << ((!!robot_.wheeldropLeft ()) ? "true" : "false")
    << " right wheel: " << ((!!robot_.wheeldropRight ()) ? "true" : "false")
    << " bump left: " << ((!!robot_.bumpLeft ()) ? "true" : "false")
    << " bump right: " << ((!!robot_.bumpRight ()) ? "true" : "false")
    << std::endl;

  std::cerr
    << "Wall: " << ((!!robot_.wall ()) ? "true" : "false")
    << std::endl;

  std::cerr
    << "Cliff: "
    << " left: " << ((!!robot_.cliffLeft ()) ? "true" : "false")
    << " front-left: " << ((!!robot_.cliffFrontLeft ()) ? "true" : "false")
    << " front-right: " << ((!!robot_.cliffFrontRight ()) ? "true" : "false")
    << " right: " << ((!!robot_.cliffRight ()) ? "true" : "false") << std::endl;

  std::cerr
    << "Digital pin status: "
    << " device detect: " << ((!!robot_.deviceDetect ()) ? "true" : "false")
    << " 3: " << ((!!robot_.digitalInput3 ()) ? "true" : "false")
    << " 2: " << ((!!robot_.digitalInput2 ()) ? "true" : "false")
    << " 1: " << ((!!robot_.digitalInput1 ()) ? "true" : "false")
    << " 0: " << ((!!robot_.digitalInput0 ()) ? "true" : "false") << std::endl;

  std::cerr
    << "Analog signal: " << robot_.analogSignal () << std::endl;

  std::cerr
    << "Chargers available? "
    << " home base: "
    << ((!!robot_.homeBaseChargerAvailable ()) ? "true" : "false")
    << " internal: "
    << ((!!robot_.internalChargerAvailable ()) ? "true" : "false") << std::endl;

  std::cerr
    << "Virtual Wall: "
    << ((!!robot_.virtualWall ()) ? "true" : "false") << std::endl;

  std::cerr
    << "Overcurrent: "
    << " left wheel: "
    << ((!!robot_.leftWheelOvercurrent ()) ? "true" : "false")
    << " right wheel: "
    << ((!!robot_.rightWheelOvercurrent ()) ? "true" : "false")
    << " LD2: " << ((!!robot_.ld2Overcurrent ()) ? "true" : "false")
    << " LD1: " << ((!!robot_.ld1Overcurrent ()) ? "true" : "false")
    << " LD0: " << ((!!robot_.ld0Overcurrent ()) ? "true" : "false");

  std::cerr
    << "IR: " << robot_.ir () << std::endl;

  std::cerr
    << "Buttons: "
    << " advance: "<< ((!!robot_.advanceButton ()) ? "true" : "false")
```

```
      << " play: " << ((!!robot_.playButton ()) ? "true" : "false") << std::endl;

    std::cerr
      << "Distance: " << robot_.distance () << std::endl;

    std::cerr
      << "Angle: " << robot_.angle () << std::endl;

    std::cerr
      << "Charging state: " << robot_.chargingState () << std::endl;

    std::cerr
      << "Battery voltage: " << robot_.batteryVoltage () << std::endl;

    std::cerr
      << "Battery current: " << robot_.batteryCurrent () << std::endl;

    std::cerr
      << "Battery temperature: " << robot_.batteryTemperature () << std::endl;

    std::cerr
      << "Battery charge: " << robot_.batteryCharge () << std::endl;

    std::cerr
      << "Battery capacity: " << robot_.batteryCapacity () << std::endl;

    std::cerr
      << "Wall signal: " << robot_.wallSignal () << std::endl;

    std::cerr
      << "Cliff left signal: "
      << robot_.cliffLeftSignal () << std::endl;

    std::cerr
      << "Cliff front-left signal: "
      << robot_.cliffFrontLeftSignal () << std::endl;

    std::cerr
      << "Cliff front-right signal: "
      << robot_.cliffFrontRightSignal () << std::endl;

    std::cerr
      << "Cliff right signal: " << robot_.cliffRightSignal () << std::endl;

    std::cerr
      << "Song number: " << static_cast<int> (robot_.songNumber ())
      << std::endl;

    std::cerr
      << "Song playing: "
      << ((!!robot_.songPlaying ()) ? "true" : "false") << std::endl;

    std::cerr
      << "Streamed packets: " << robot_.streamPackets () << std::endl;

    std::cerr
      << "Requested velocty: "
      << robot_.requestedVelocity () << std::endl;

    std::cerr
      << "Request radius: "
      << robot_.requestedRadius () << std::endl;

    std::cerr
      << "Request left velocity: "
      << robot_.requestedLeftVelocity () << std::endl;

```

```
  std::cerr
    << "Request right velocity: "
    << robot_.requestedRightVelocity () << std::endl;

}
```

#### 6.1.2.17 void Application::usage () [inline]

```
{
  help (1);
}
```

#### 6.1.2.18 void Application::version () [inline]

```
{
  std::cerr
    << PACKAGE_STRING << std::endl
    << "Copyright (C) 2008  iLab."
    << std::endl
    << "This is free software; see the source for copying conditions."
    << std::endl
    << "There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A"
    << std::endl
    << "PARTICULAR PURPOSE." << std::endl;
  exit (0);
}
```

The documentation for this struct was generated from the following files:

- /home/thomas/projects/usc/libirobot-create/_build/libirobot-create-0.1/src/**irobot-create-ctl.cc**
- /home/thomas/projects/usc/libirobot-create/src/**irobot-create-ctl.cc**

## 6.2 iRobot::Create Class Reference

Robot communication class, wraps Open Interface version 2.

`#include <irobot-create.hh>`

Collaboration diagram for iRobot::Create:

### Public Types

- enum **Mode** {
  **IROBOT_CREATE_OFF**, **IROBOT_CREATE_PASSIVE**, **IROBOT_CREATE_SAFE**, **IROBOT_CREATE_FULL**,
  **IROBOT_CREATE_OFF**, **IROBOT_CREATE_PASSIVE**, **IROBOT_CREATE_SAFE**, **IROBOT_CREATE_FULL** }

  ***iRobot*** *(p. 13)* ***Create*** *(p. 29) modes The mode defines which commands are available and how the robot behaves when some events happens like wheel drop or if a cliff is detected.*

- enum **Baud** {
  **BAUD_300** = 0, **BAUD_600** = 1, **BAUD_1200** = 2, **BAUD_2400** = 3,
  **BAUD_4800** = 4, **BAUD_9600** = 5, **BAUD_14400** = 6, **BAUD_19200** = 7,
  **BAUD_28800** = 8, **BAUD_38400** = 9, **BAUD_57600** = 10, **BAUD_115200** = 11,
  **BAUD_300** = 0, **BAUD_600** = 1, **BAUD_1200** = 2, **BAUD_2400** = 3,
  **BAUD_4800** = 4, **BAUD_9600** = 5, **BAUD_14400** = 6, **BAUD_19200** = 7,
  **BAUD_28800** = 8, **BAUD_38400** = 9, **BAUD_57600** = 10, **BAUD_115200** = 11
  }

  *Enumerate all possible communication speeds.*

- enum **Demo** {
  **DEMO_COVER** = 0, **DEMO_COVER_AND_DOCK** = 1, **DEMO_SPOT_-COVER** = 2, **DEMO_MOUSE** = 3,
  **DEMO_DRIVE_FIGURE_EIGHT** = 4, **DEMO_WIMP** = 5, **DEMO_HOME** = 6, **DEMO_TAG** = 7,
  **DEMO_PACHELBEL** = 8, **DEMO_BANJO** = 9, **DEMO_ABORT** = 255, **DEMO_COVER** = 0,
  **DEMO_COVER_AND_DOCK** = 1, **DEMO_SPOT_COVER** = 2, **DEMO_-MOUSE** = 3, **DEMO_DRIVE_FIGURE_EIGHT** = 4,
  **DEMO_WIMP** = 5, **DEMO_HOME** = 6, **DEMO_TAG** = 7, **DEMO_-PACHELBEL** = 8,
  **DEMO_BANJO** = 9, **DEMO_ABORT** = 255 }

  *Enumerate built-in demos.*

- enum **DriveCommand** {
  **DRIVE_STRAIGHT**, **DRIVE_INPLACE_CLOCKWISE**, **DRIVE_-INPLACE_COUNTERCLOCKWISE**, **DRIVE_STRAIGHT**,
  **DRIVE_INPLACE_CLOCKWISE**, **DRIVE_INPLACE_-COUNTERCLOCKWISE** }

  *Enumerates special driving commands.*

- enum **Led** {
  **LED_NONE** = 0, **LED_PLAY** = 2, **LED_ADVANCE** = 8, **LED_ALL** = LED_-PLAY | LED_ADVANCE,
  **LED_NONE** = 0, **LED_PLAY** = 2, **LED_ADVANCE** = 8, **LED_ALL** = LED_-PLAY | LED_ADVANCE }

  *Enumerate available leds (and combinations). The power led is not in this list because it is handled differently (as its color is not fixed).*

- enum **StreamState** { **STREAM_STATE_OFF** = 0, **STREAM_STATE_ON** = 1, **STREAM_STATE_OFF** = 0, **STREAM_STATE_ON** = 1 }

  *Activate or disable sensor streaming.*

- enum **ChargingState** {
  **CHARGING_STATE_NOT_CHARGING** = 0, **CHARGING_STATE_-RECONDITIONING_CHARGING** = 1, **CHARGING_STATE_FULL_-CHARGING** = 2, **CHARGING_STATE_TRICKLE_CHARGING** = 3,

**CHARGING_STATE_WAITING** = 4, **CHARGING_STATE_CHARGING_-FAULT_CONDITION** = 5, **CHARGING_STATE_NOT_CHARGING** = 0, **CHARGING_STATE_RECONDITIONING_CHARGING** = 1,

**CHARGING_STATE_FULL_CHARGING** = 2, **CHARGING_STATE_-TRICKLE_CHARGING** = 3, **CHARGING_STATE_WAITING** = 4, **CHARGING_STATE_CHARGING_FAULT_CONDITION** = 5 }

*Enumerate possible charging states.*

- enum **Event** {

  **EVENT_WHEEL_DROP** = 1, **EVENT_FRONT_WHEEL_DROP** = 2, **EVENT_LEFT_WHEEL_DROP** = 3, **EVENT_RIGHT_WHEEL_DROP** = 4,

  **EVENT_BUMP** = 5, **EVENT_LEFT_BUMP** = 6, **EVENT_RIGHT_BUMP** = 7, **EVENT_VIRTUAL_WALL** = 8,

  **EVENT_WALL** = 9, **EVENT_CLIFF** = 10, **EVENT_LEFT_CLIFF** = 11, **EVENT_FRONT_LEFT_CLIFF** = 12,

  **EVENT_FRONT_RIGHT_CLIFF** = 13, **EVENT_RIGHT_CLIFF** = 14, **EVENT_HOME_BASE** = 15, **EVENT_ADVANCE_BUTTON** = 16,

  **EVENT_PLAY_BUTTON** = 17, **EVENT_DIGITAL_INPUT_0** = 18, **EVENT_DIGITAL_INPUT_1** = 19, **EVENT_DIGITAL_INPUT_2** = 20,

  **EVENT_DIGITAL_INPUT_3** = 21, **EVENT_OI_MODE_PASSIVE** = 22, **EVENT_WHEEL_DROP** = 1, **EVENT_FRONT_WHEEL_DROP** = 2,

  **EVENT_LEFT_WHEEL_DROP** = 3, **EVENT_RIGHT_WHEEL_DROP** = 4, **EVENT_BUMP** = 5, **EVENT_LEFT_BUMP** = 6,

  **EVENT_RIGHT_BUMP** = 7, **EVENT_VIRTUAL_WALL** = 8, **EVENT_-WALL** = 9, **EVENT_CLIFF** = 10,

  **EVENT_LEFT_CLIFF** = 11, **EVENT_FRONT_LEFT_CLIFF** = 12, **EVENT_FRONT_RIGHT_CLIFF** = 13, **EVENT_RIGHT_CLIFF** = 14,

  **EVENT_HOME_BASE** = 15, **EVENT_ADVANCE_BUTTON** = 16, **EVENT_PLAY_BUTTON** = 17, **EVENT_DIGITAL_INPUT_0** = 18,

  **EVENT_DIGITAL_INPUT_1** = 19, **EVENT_DIGITAL_INPUT_2** = 20, **EVENT_DIGITAL_INPUT_3** = 21, **EVENT_OI_MODE_PASSIVE** = 22 }

  *Enumerate available events.*

- enum **EventState** { **EVENT_OCCURRING**, **EVENT_NOT_OCCURRING**, **EVENT_OCCURRING**, **EVENT_NOT_OCCURRING** }

  *Indicate an event state.*

- enum **Opcode** {

  **OPCODE_START** = 128, **OPCODE_BAUD** = 129, **OPCODE_CONTROL** = 130, **OPCODE_SAFE** = 131,

  **OPCODE_FULL** = 132, **OPCODE_SPOT** = 134, **OPCODE_COVER** = 135, **OPCODE_DEMO** = 136,

  **OPCODE_DRIVE** = 137, **OPCODE_LOW_SIDE_DRIVERS** = 138, **OPCODE_LEDS** = 139, **OPCODE_SONG** = 140,

  **OPCODE_PLAY** = 141, **OPCODE_SENSORS** = 142, **OPCODE_COVER_-AND_DOCK** = 143, **OPCODE_PWM_LOW_SIDE_DRIVERS** = 144,

**OPCODE_DRIVE_DIRECT** = 145, **OPCODE_DIGITAL_OUTPUTS** = 147, **OPCODE_STREAM** = 148, **OPCODE_QUERY_LIST** = 149,

**OPCODE_PAUSE_RESUME_STREAM** = 150, **OPCODE_SEND_IR** = 151, **OPCODE_SCRIPT** = 152, **OPCODE_PLAY_SCRIPT** = 153,

**OPCODE_SHOW_SCRIPT** = 154, **OPCODE_WAIT_TIME** = 155, **OPCODE_WAIT_DISTANCE** = 156, **OPCODE_WAIT_ANGLE** = 157,

**OPCODE_WAIT_EVENT** = 158, **OPCODE_START** = 128, **OPCODE_BAUD** = 129, **OPCODE_CONTROL** = 130,

**OPCODE_SAFE** = 131, **OPCODE_FULL** = 132, **OPCODE_SPOT** = 134, **OPCODE_COVER** = 135,

**OPCODE_DEMO** = 136, **OPCODE_DRIVE** = 137, **OPCODE_LOW_SIDE_-DRIVERS** = 138, **OPCODE_LEDS** = 139,

**OPCODE_SONG** = 140, **OPCODE_PLAY** = 141, **OPCODE_SENSORS** = 142, **OPCODE_COVER_AND_DOCK** = 143,

**OPCODE_PWM_LOW_SIDE_DRIVERS** = 144, **OPCODE_DRIVE_-DIRECT** = 145, **OPCODE_DIGITAL_OUTPUTS** = 147, **OPCODE_STREAM** = 148,

**OPCODE_QUERY_LIST** = 149, **OPCODE_PAUSE_RESUME_STREAM** = 150, **OPCODE_SEND_IR** = 151, **OPCODE_SCRIPT** = 152,

**OPCODE_PLAY_SCRIPT** = 153, **OPCODE_SHOW_SCRIPT** = 154, **OPCODE_WAIT_TIME** = 155, **OPCODE_WAIT_DISTANCE** = 156,

**OPCODE_WAIT_ANGLE** = 157, **OPCODE_WAIT_EVENT** = 158 }

*Enumerate available opcodes. Opcode are the basics instructions that the protocol support. This class wraps them to avoid using them directly. The only case where you have to use them is when you want to send an Open Interface script.*

- enum **SensorPacket** {

**SENSOR_GROUP_0** = 0, **SENSOR_GROUP_1** = 1, **SENSOR_GROUP_2** = 2, **SENSOR_GROUP_3** = 3,

**SENSOR_GROUP_4** = 4, **SENSOR_GROUP_5** = 5, **SENSOR_GROUP_6** = 6, **SENSOR_BUMPS_WHEELS_DROPS** = 7,

**SENSOR_WALL** = 8, **SENSOR_CLIFF_LEFT** = 9, **SENSOR_CLIFF_-FRONT_LEFT** = 10, **SENSOR_CLIFF_FRONT_RIGHT** = 11,

**SENSOR_CLIFF_RIGHT** = 12, **SENSOR_VIRTUAL_WALL** = 13, **SENSOR_OVERCURRENTS** = 14, **SENSOR_IR** = 17,

**SENSOR_BUTTONS** = 18, **SENSOR_DISTANCE** = 19, **SENSOR_ANGLE** = 20, **SENSOR_CHARGING_STATE** = 21,

**SENSOR_VOLTAGE** = 22, **SENSOR_CURRENT** = 23, **SENSOR_-BATTERY_TEMPERATURE** = 24, **SENSOR_BATTERY_CHARGE** = 25,

**SENSOR_BATTERY_CAPACITY** = 26, **SENSOR_WALL_SIGNAL** = 27, **SENSOR_CLIFF_LEFT_SIGNAL** = 28, **SENSOR_CLIFF_FRONT_-LEFT_SIGNAL** = 29,

**SENSOR_CLIFF_FRONT_RIGHT_SIGNAL** = 30, **SENSOR_CLIFF_-RIGHT_SIGNAL** = 31, **SENSOR_CARGO_BAY_DIGITAL_INPUT** = 32, **SENSOR_CARGO_BAY_ANALOG_SIGNAL** = 33,

**SENSOR_CHARGING_SOURCES_AVAILABLE** = 34, **SENSOR_OI_MODE** = 35, **SENSOR_SONG_NUMBER** = 36, **SENSOR_SONG_PLAYING** = 37,

**SENSOR_NUMBER_STREAM_PACKETS** = 38, **SENSOR_REQUESTED_VELOCITY** = 39, **SENSOR_REQUESTED_RADIUS** = 40, **SENSOR_REQUESTED_RIGHT_VELOCITY** = 41,

**SENSOR_REQUESTED_LEFT_VELOCITY** = 42, **SENSOR_GROUP_0** = 0, **SENSOR_GROUP_1** = 1, **SENSOR_GROUP_2** = 2,

**SENSOR_GROUP_3** = 3, **SENSOR_GROUP_4** = 4, **SENSOR_GROUP_5** = 5, **SENSOR_GROUP_6** = 6,

**SENSOR_BUMPS_WHEELS_DROPS** = 7, **SENSOR_WALL** = 8, **SENSOR_CLIFF_LEFT** = 9, **SENSOR_CLIFF_FRONT_LEFT** = 10,

**SENSOR_CLIFF_FRONT_RIGHT** = 11, **SENSOR_CLIFF_RIGHT** = 12, **SENSOR_VIRTUAL_WALL** = 13, **SENSOR_OVERCURRENTS** = 14,

**SENSOR_IR** = 17, **SENSOR_BUTTONS** = 18, **SENSOR_DISTANCE** = 19, **SENSOR_ANGLE** = 20,

**SENSOR_CHARGING_STATE** = 21, **SENSOR_VOLTAGE** = 22, **SENSOR_CURRENT** = 23, **SENSOR_BATTERY_TEMPERATURE** = 24,

**SENSOR_BATTERY_CHARGE** = 25, **SENSOR_BATTERY_CAPACITY** = 26, **SENSOR_WALL_SIGNAL** = 27, **SENSOR_CLIFF_LEFT_SIGNAL** = 28,

**SENSOR_CLIFF_FRONT_LEFT_SIGNAL** = 29, **SENSOR_CLIFF_FRONT_RIGHT_SIGNAL** = 30, **SENSOR_CLIFF_RIGHT_SIGNAL** = 31, **SENSOR_CARGO_BAY_DIGITAL_INPUT** = 32,

**SENSOR_CARGO_BAY_ANALOG_SIGNAL** = 33, **SENSOR_CHARGING_SOURCES_AVAILABLE** = 34, **SENSOR_OI_MODE** = 35, **SENSOR_SONG_NUMBER** = 36,

**SENSOR_SONG_PLAYING** = 37, **SENSOR_NUMBER_STREAM_PACKETS** = 38, **SENSOR_REQUESTED_VELOCITY** = 39, **SENSOR_REQUESTED_RADIUS** = 40,

**SENSOR_REQUESTED_RIGHT_VELOCITY** = 41, **SENSOR_REQUESTED_LEFT_VELOCITY** = 42 }

*Enumerate available sensor packets.*

- enum **Mode** {

**IROBOT_CREATE_OFF**, **IROBOT_CREATE_PASSIVE**, **IROBOT_CREATE_SAFE**, **IROBOT_CREATE_FULL**,

**IROBOT_CREATE_OFF**, **IROBOT_CREATE_PASSIVE**, **IROBOT_CREATE_SAFE**, **IROBOT_CREATE_FULL** }

***iRobot*** *(*p. *13)* ***Create*** *(*p. *29) modes The mode defines which commands are available and how the robot behaves when some events happens like wheel drop or if a cliff is detected.*

- enum **Baud** {

**BAUD_300** = 0, **BAUD_600** = 1, **BAUD_1200** = 2, **BAUD_2400** = 3,

**BAUD_4800** = 4, **BAUD_9600** = 5, **BAUD_14400** = 6, **BAUD_19200** = 7,

**BAUD_28800** = 8, **BAUD_38400** = 9, **BAUD_57600** = 10, **BAUD_115200** = 11,

**BAUD_300** = 0, **BAUD_600** = 1, **BAUD_1200** = 2, **BAUD_2400** = 3,

**BAUD_4800** = 4, **BAUD_9600** = 5, **BAUD_14400** = 6, **BAUD_19200** = 7,

**BAUD_28800** = 8, **BAUD_38400** = 9, **BAUD_57600** = 10, **BAUD_115200** = 11 }

*Enumerate all possible communication speeds.*

- enum **Demo** {

  **DEMO_COVER** = 0, **DEMO_COVER_AND_DOCK** = 1, **DEMO_SPOT_-COVER** = 2, **DEMO_MOUSE** = 3,

  **DEMO_DRIVE_FIGURE_EIGHT** = 4, **DEMO_WIMP** = 5, **DEMO_HOME** = 6, **DEMO_TAG** = 7,

  **DEMO_PACHELBEL** = 8, **DEMO_BANJO** = 9, **DEMO_ABORT** = 255, **DEMO_COVER** = 0,

  **DEMO_COVER_AND_DOCK** = 1, **DEMO_SPOT_COVER** = 2, **DEMO_-MOUSE** = 3, **DEMO_DRIVE_FIGURE_EIGHT** = 4,

  **DEMO_WIMP** = 5, **DEMO_HOME** = 6, **DEMO_TAG** = 7, **DEMO_-PACHELBEL** = 8,

  **DEMO_BANJO** = 9, **DEMO_ABORT** = 255 }

  *Enumerate built-in demos.*

- enum **DriveCommand** {

  **DRIVE_STRAIGHT**, **DRIVE_INPLACE_CLOCKWISE**, **DRIVE_-INPLACE_COUNTERCLOCKWISE**, **DRIVE_STRAIGHT**,

  **DRIVE_INPLACE_CLOCKWISE**, **DRIVE_INPLACE_-COUNTERCLOCKWISE** }

  *Enumerates special driving commands.*

- enum **Led** {

  **LED_NONE** = 0, **LED_PLAY** = 2, **LED_ADVANCE** = 8, **LED_ALL** = LED_-PLAY | LED_ADVANCE,

  **LED_NONE** = 0, **LED_PLAY** = 2, **LED_ADVANCE** = 8, **LED_ALL** = LED_-PLAY | LED_ADVANCE }

  *Enumerate available leds (and combinations). The power led is not in this list because it is handled differently (as its color is not fixed).*

- enum **StreamState** { **STREAM_STATE_OFF** = 0, **STREAM_STATE_ON** = 1, **STREAM_STATE_OFF** = 0, **STREAM_STATE_ON** = 1 }

  *Activate or disable sensor streaming.*

- enum **ChargingState** {

  **CHARGING_STATE_NOT_CHARGING** = 0, **CHARGING_STATE_-RECONDITIONING_CHARGING** = 1, **CHARGING_STATE_FULL_-CHARGING** = 2, **CHARGING_STATE_TRICKLE_CHARGING** = 3,

  **CHARGING_STATE_WAITING** = 4, **CHARGING_STATE_CHARGING_-FAULT_CONDITION** = 5, **CHARGING_STATE_NOT_CHARGING** = 0, **CHARGING_STATE_RECONDITIONING_CHARGING** = 1,

  **CHARGING_STATE_FULL_CHARGING** = 2, **CHARGING_STATE_-TRICKLE_CHARGING** = 3, **CHARGING_STATE_WAITING** = 4, **CHARGING_STATE_CHARGING_FAULT_CONDITION** = 5 }

  *Enumerate possible charging states.*

- enum **Event** {

  **EVENT_WHEEL_DROP** = 1, **EVENT_FRONT_WHEEL_DROP** = 2, **EVENT_LEFT_WHEEL_DROP** = 3, **EVENT_RIGHT_WHEEL_DROP** = 4,

  **EVENT_BUMP** = 5, **EVENT_LEFT_BUMP** = 6, **EVENT_RIGHT_BUMP** = 7, **EVENT_VIRTUAL_WALL** = 8,

  **EVENT_WALL** = 9, **EVENT_CLIFF** = 10, **EVENT_LEFT_CLIFF** = 11, **EVENT_FRONT_LEFT_CLIFF** = 12,

  **EVENT_FRONT_RIGHT_CLIFF** = 13, **EVENT_RIGHT_CLIFF** = 14, **EVENT_HOME_BASE** = 15, **EVENT_ADVANCE_BUTTON** = 16,

  **EVENT_PLAY_BUTTON** = 17, **EVENT_DIGITAL_INPUT_0** = 18, **EVENT_DIGITAL_INPUT_1** = 19, **EVENT_DIGITAL_INPUT_2** = 20,

  **EVENT_DIGITAL_INPUT_3** = 21, **EVENT_OI_MODE_PASSIVE** = 22, **EVENT_WHEEL_DROP** = 1, **EVENT_FRONT_WHEEL_DROP** = 2,

  **EVENT_LEFT_WHEEL_DROP** = 3, **EVENT_RIGHT_WHEEL_DROP** = 4, **EVENT_BUMP** = 5, **EVENT_LEFT_BUMP** = 6,

  **EVENT_RIGHT_BUMP** = 7, **EVENT_VIRTUAL_WALL** = 8, **EVENT_-WALL** = 9, **EVENT_CLIFF** = 10,

  **EVENT_LEFT_CLIFF** = 11, **EVENT_FRONT_LEFT_CLIFF** = 12, **EVENT_FRONT_RIGHT_CLIFF** = 13, **EVENT_RIGHT_CLIFF** = 14,

  **EVENT_HOME_BASE** = 15, **EVENT_ADVANCE_BUTTON** = 16, **EVENT_PLAY_BUTTON** = 17, **EVENT_DIGITAL_INPUT_0** = 18,

  **EVENT_DIGITAL_INPUT_1** = 19, **EVENT_DIGITAL_INPUT_2** = 20, **EVENT_DIGITAL_INPUT_3** = 21, **EVENT_OI_MODE_PASSIVE** = 22 }

  *Enumerate available events.*

- enum **EventState** { **EVENT_OCCURRING**, **EVENT_NOT_OCCURRING**, **EVENT_OCCURRING**, **EVENT_NOT_OCCURRING** }

  *Indicate an event state.*

- enum **Opcode** {

  **OPCODE_START** = 128, **OPCODE_BAUD** = 129, **OPCODE_CONTROL** = 130, **OPCODE_SAFE** = 131,

  **OPCODE_FULL** = 132, **OPCODE_SPOT** = 134, **OPCODE_COVER** = 135, **OPCODE_DEMO** = 136,

  **OPCODE_DRIVE** = 137, **OPCODE_LOW_SIDE_DRIVERS** = 138, **OPCODE_LEDS** = 139, **OPCODE_SONG** = 140,

  **OPCODE_PLAY** = 141, **OPCODE_SENSORS** = 142, **OPCODE_COVER_-AND_DOCK** = 143, **OPCODE_PWM_LOW_SIDE_DRIVERS** = 144,

  **OPCODE_DRIVE_DIRECT** = 145, **OPCODE_DIGITAL_OUTPUTS** = 147, **OPCODE_STREAM** = 148, **OPCODE_QUERY_LIST** = 149,

  **OPCODE_PAUSE_RESUME_STREAM** = 150, **OPCODE_SEND_IR** = 151, **OPCODE_SCRIPT** = 152, **OPCODE_PLAY_SCRIPT** = 153,

  **OPCODE_SHOW_SCRIPT** = 154, **OPCODE_WAIT_TIME** = 155, **OPCODE_WAIT_DISTANCE** = 156, **OPCODE_WAIT_ANGLE** = 157,

**OPCODE_WAIT_EVENT** = 158, **OPCODE_START** = 128, **OPCODE_BAUD** = 129, **OPCODE_CONTROL** = 130,

**OPCODE_SAFE** = 131, **OPCODE_FULL** = 132, **OPCODE_SPOT** = 134, **OPCODE_COVER** = 135,

**OPCODE_DEMO** = 136, **OPCODE_DRIVE** = 137, **OPCODE_LOW_SIDE_-DRIVERS** = 138, **OPCODE_LEDS** = 139,

**OPCODE_SONG** = 140, **OPCODE_PLAY** = 141, **OPCODE_SENSORS** = 142, **OPCODE_COVER_AND_DOCK** = 143,

**OPCODE_PWM_LOW_SIDE_DRIVERS** = 144, **OPCODE_DRIVE_-DIRECT** = 145, **OPCODE_DIGITAL_OUTPUTS** = 147, **OPCODE_STREAM** = 148,

**OPCODE_QUERY_LIST** = 149, **OPCODE_PAUSE_RESUME_STREAM** = 150, **OPCODE_SEND_IR** = 151, **OPCODE_SCRIPT** = 152,

**OPCODE_PLAY_SCRIPT** = 153, **OPCODE_SHOW_SCRIPT** = 154, **OPCODE_WAIT_TIME** = 155, **OPCODE_WAIT_DISTANCE** = 156,

**OPCODE_WAIT_ANGLE** = 157, **OPCODE_WAIT_EVENT** = 158 }

*Enumerate available opcodes. Opcode are the basics instructions that the protocol support. This class wraps them to avoid using them directly. The only case where you have to use them is when you want to send an Open Interface script.*

- enum **SensorPacket** {

**SENSOR_GROUP_0** = 0, **SENSOR_GROUP_1** = 1, **SENSOR_GROUP_2** = 2, **SENSOR_GROUP_3** = 3,

**SENSOR_GROUP_4** = 4, **SENSOR_GROUP_5** = 5, **SENSOR_GROUP_6** = 6, **SENSOR_BUMPS_WHEELS_DROPS** = 7,

**SENSOR_WALL** = 8, **SENSOR_CLIFF_LEFT** = 9, **SENSOR_CLIFF_-FRONT_LEFT** = 10, **SENSOR_CLIFF_FRONT_RIGHT** = 11,

**SENSOR_CLIFF_RIGHT** = 12, **SENSOR_VIRTUAL_WALL** = 13, **SENSOR_OVERCURRENTS** = 14, **SENSOR_IR** = 17,

**SENSOR_BUTTONS** = 18, **SENSOR_DISTANCE** = 19, **SENSOR_ANGLE** = 20, **SENSOR_CHARGING_STATE** = 21,

**SENSOR_VOLTAGE** = 22, **SENSOR_CURRENT** = 23, **SENSOR_-BATTERY_TEMPERATURE** = 24, **SENSOR_BATTERY_CHARGE** = 25,

**SENSOR_BATTERY_CAPACITY** = 26, **SENSOR_WALL_SIGNAL** = 27, **SENSOR_CLIFF_LEFT_SIGNAL** = 28, **SENSOR_CLIFF_FRONT_-LEFT_SIGNAL** = 29,

**SENSOR_CLIFF_FRONT_RIGHT_SIGNAL** = 30, **SENSOR_CLIFF_-RIGHT_SIGNAL** = 31, **SENSOR_CARGO_BAY_DIGITAL_INPUT** = 32, **SENSOR_CARGO_BAY_ANALOG_SIGNAL** = 33,

**SENSOR_CHARGING_SOURCES_AVAILABLE** = 34, **SENSOR_OI_-MODE** = 35, **SENSOR_SONG_NUMBER** = 36, **SENSOR_SONG_PLAYING** = 37,

**SENSOR_NUMBER_STREAM_PACKETS** = 38, **SENSOR_REQUESTED_-VELOCITY** = 39, **SENSOR_REQUESTED_RADIUS** = 40, **SENSOR_-REQUESTED_RIGHT_VELOCITY** = 41,

**SENSOR_REQUESTED_LEFT_VELOCITY** = 42, **SENSOR_GROUP_0** = 0, **SENSOR_GROUP_1** = 1, **SENSOR_GROUP_2** = 2,

**SENSOR_GROUP_3** = 3, **SENSOR_GROUP_4** = 4, **SENSOR_GROUP_5** = 5, **SENSOR_GROUP_6** = 6,

**SENSOR_BUMPS_WHEELS_DROPS** = 7, **SENSOR_WALL** = 8, **SENSOR_-CLIFF_LEFT** = 9, **SENSOR_CLIFF_FRONT_LEFT** = 10,

**SENSOR_CLIFF_FRONT_RIGHT** = 11, **SENSOR_CLIFF_RIGHT** = 12, **SENSOR_VIRTUAL_WALL** = 13, **SENSOR_OVERCURRENTS** = 14,

**SENSOR_IR** = 17, **SENSOR_BUTTONS** = 18, **SENSOR_DISTANCE** = 19, **SENSOR_ANGLE** = 20,

**SENSOR_CHARGING_STATE** = 21, **SENSOR_VOLTAGE** = 22, **SENSOR_-CURRENT** = 23, **SENSOR_BATTERY_TEMPERATURE** = 24,

**SENSOR_BATTERY_CHARGE** = 25, **SENSOR_BATTERY_CAPACITY** = 26, **SENSOR_WALL_SIGNAL** = 27, **SENSOR_CLIFF_LEFT_SIGNAL** = 28,

**SENSOR_CLIFF_FRONT_LEFT_SIGNAL** = 29, **SENSOR_CLIFF_-FRONT_RIGHT_SIGNAL** = 30, **SENSOR_CLIFF_RIGHT_SIGNAL** = 31, **SENSOR_CARGO_BAY_DIGITAL_INPUT** = 32,

**SENSOR_CARGO_BAY_ANALOG_SIGNAL** = 33, **SENSOR_-CHARGING_SOURCES_AVAILABLE** = 34, **SENSOR_OI_MODE** = 35, **SENSOR_SONG_NUMBER** = 36,

**SENSOR_SONG_PLAYING** = 37, **SENSOR_NUMBER_STREAM_-PACKETS** = 38, **SENSOR_REQUESTED_VELOCITY** = 39, **SENSOR_-REQUESTED_RADIUS** = 40,

**SENSOR_REQUESTED_RIGHT_VELOCITY** = 41, **SENSOR_-REQUESTED_LEFT_VELOCITY** = 42 }

*Enumerate available sensor packets.*

- typedef std::pair< unsigned char, unsigned char > **note_t**

  *Define a song note (frequency, duration).*

- typedef std::vector< **note_t** > **song_t**

  *Define a song (a list of notes).*

- typedef std::vector< unsigned char > **opcodes_t**

  *Define a vector of opcodes and arguments (a script).*

- typedef std::vector< **SensorPacket** > **sensorPackets_t**

  *Define a vector of sensor packets.*

- typedef std::queue< **SensorPacket** > **queriedPackets_t**
- typedef std::pair< unsigned char, unsigned char > **note_t**

  *Define a song note (frequency, duration).*

- typedef std::vector< **note_t** > **song_t**

  *Define a song (a list of notes).*

- typedef std::vector< unsigned char > **opcodes_t**

  *Define a vector of opcodes and arguments (a script).*

- typedef std::vector< **SensorPacket** > **sensorPackets_t**

  *Define a vector of sensor packets.*

- typedef std::queue< **SensorPacket** > **queriedPackets_t**

## Public Member Functions

- **Create** (std::iostream &stream) throw (InvalidArgument)
  *Construct an instance of* ***Create*** *(*p. *29).*
- **Create** (LibSerial::SerialStream &stream) throw (InvalidArgument, LibSerialNotAvailable)
  *Construct an instance of* ***Create*** *(*p. *29) using serial port communication.*
- ~**Create** () throw ()
  *Destroy an instance of* ***Create*** *(*p. *29).*

- void **sendBaudCommand** (**Baud** speed) throw (CommandNotAvailable, InvalidArgument)
  *Change the communication speed. After changing the communication speed, it is required to wait at least 100ms.*
- void **sendStartCommand** ()
  *Switch to passive mode. Automatically done when the class is instanciated.*
- void **sendSafeCommand** () throw (CommandNotAvailable)
  *Switch to safe mode.*
- void **sendFullCommand** () throw (CommandNotAvailable)
  *Switch to full mode.*
- void **sendDemoCommand** (**Demo** demo) throw (CommandNotAvailable, InvalidArgument)
  *Run a built-in demo.*
- void **sendDriveCommand** (short v, short r) throw (CommandNotAvailable, InvalidArgument)
  *Drive the robot.*
- void **sendDriveCommand** (short v, **DriveCommand** dc) throw (CommandNotAvailable, InvalidArgument)
  *Drive the robot using a special drigin mode.*
- void **sendDriveDirectCommand** (short vr, short vl) throw (CommandNotAvailable, InvalidArgument)
  *Drive the two wheels separately.*
- void **sendLedCommand** (**Led** l, unsigned char c, unsigned char i) throw (CommandNotAvailable, InvalidArgument)
  *Change led status.*
- void **sendDigitalOutputsCommand** (bool d1, bool d2, bool d3) throw (CommandNotAvailable)
  *Change the digital output values.*

- void **sendPwmLowSideDriversCommand** (unsigned char lsd1, unsigned char lsd2, unsigned char lsd3) throw (CommandNotAvailable, InvalidArgument)

  *Drive low side drivers with variable power.*

- void **sendLowSideDriversCommand** (bool lsd1, bool lsd2, bool lsd3) throw (CommandNotAvailable)

  *Switch on/off low side drivers.*

- void **sendIrCommand** (unsigned char v) throw (CommandNotAvailable)

  *Send an IR command on low side driver 1. The data will be encoded using the iRobot's **Create** (p. 29) receiver format.*

- void **sendSongCommand** (unsigned char sid, const **song_t** &song) throw (CommandNotAvailable, InvalidArgument)

  *Define a song.*

- void **sendPlaySongCommand** (unsigned char sid) throw (CommandNotAvailable, InvalidArgument)

  *Play a song.*

- void **sendSensorsCommand** (**SensorPacket** sp) throw (CommandNotAvailable, InvalidArgument)

  *Request the robot to send a particular sensor packet.*

- void **sendQueryListCommand** (const **sensorPackets_t** &lsp) throw (CommandNotAvailable, InvalidArgument)

  *Request the robot to send a list of sensor packets.*

- void **sendStreamCommand** (const **sensorPackets_t** &lsp) throw (CommandNotAvailable, InvalidArgument)

  *Request the robot to start streaming some sensor packets. After the streaming is started, the requested values will be sent every 15ms.*

- void **sendPauseStreamCommand** (**StreamState** st) throw (CommandNotAvailable, InvalidArgument)

  *Pause or resume sensor streaming.*

- void **sendScriptCommand** (const **opcodes_t** &script) throw (CommandNotAvailable, InvalidArgument)

  *Send a script.*

- void **sendPlayScriptCommand** () throw (CommandNotAvailable)

  *Play the current stored script.*

- void **sendShowScriptCommand** () throw (CommandNotAvailable)

  *Show the stored script.*

- void **sendWaitTimeCommand** (unsigned char t) throw (CommandNotAvailable)

  *Make the robot wait for a specific amount of time.*

- void **sendWaitDistanceCommand** (short d) throw (CommandNotAvailable)

  *Make the robot wait until it travelled a certain distance.*

- void **sendWaitAngleCommand** (short a) throw (CommandNotAvailable)

  *Make the robot wait until it has rotated enough.*

- void **sendWaitEventCommand** (**Event** e, **EventState** es=EVENT_OCCURRING) throw (CommandNotAvailable, InvalidArgument)

  *Make the robot wait for a specific event.*

- void **updateSensors** ()

  *Read the stream to update sensors values. This handles both streamed data and specific queries through querylist or sensors commands.*

- void **readStream** ()

  *Read streamed sensors and update sensors values.*

- **Mode mode** () throw ()
- bool **wheeldropCaster** ()
- bool **wheeldropLeft** ()
- bool **wheeldropRight** ()
- bool **bumpLeft** ()
- bool **bumpRight** ()
- bool **wall** ()

  *Get wall sensor value.*

- bool **cliffLeft** ()
- bool **cliffFrontLeft** ()
- bool **cliffFrontRight** ()
- bool **cliffRight** ()
- bool **deviceDetect** ()
- bool **digitalInput3** ()
- bool **digitalInput2** ()
- bool **digitalInput1** ()
- bool **digitalInput0** ()
- short **analogSignal** ()
- bool **homeBaseChargerAvailable** ()
- bool **internalChargerAvailable** ()
- bool **virtualWall** ()
- bool **leftWheelOvercurrent** ()
- bool **rightWheelOvercurrent** ()
- bool **ld2Overcurrent** ()
- bool **ld1Overcurrent** ()
- bool **ld0Overcurrent** ()
- unsigned char **ir** ()
- bool **advanceButton** ()
- bool **playButton** ()
- short **distance** ()
- short **angle** ()
- **ChargingState chargingState** ()
- short **batteryVoltage** ()
- short **batteryCurrent** ()
- short **batteryTemperature** ()
- short **batteryCharge** ()
- short **batteryCapacity** ()
- short **wallSignal** ()
- short **cliffLeftSignal** ()
- short **cliffFrontLeftSignal** ()
- short **cliffFrontRightSignal** ()
- short **cliffRightSignal** ()

- unsigned char **songNumber** ()
- bool **songPlaying** ()
- unsigned char **streamPackets** ()
- short **requestedVelocity** ()
- short **requestedRadius** ()
- short **requestedLeftVelocity** ()
- short **requestedRightVelocity** ()

- **Create** (std::iostream &stream) throw (InvalidArgument)

  *Construct an instance of* ***Create*** *(*p. *29).*

- **Create** (LibSerial::SerialStream &stream) throw (InvalidArgument, LibSerialNotAvailable)

  *Construct an instance of* ***Create*** *(*p. *29) using serial port communication.*

- ~**Create** () throw ()

  *Destroy an instance of* ***Create*** *(*p. *29).*

- void **sendBaudCommand** (**Baud** speed) throw (CommandNotAvailable, InvalidArgument)

  *Change the communication speed. After changing the communication speed, it is required to wait at least 100ms.*

- void **sendStartCommand** ()

  *Switch to passive mode. Automatically done when the class is instanciated.*

- void **sendSafeCommand** () throw (CommandNotAvailable)

  *Switch to safe mode.*

- void **sendFullCommand** () throw (CommandNotAvailable)

  *Switch to full mode.*

- void **sendDemoCommand** (**Demo** demo) throw (CommandNotAvailable, InvalidArgument)

  *Run a built-in demo.*

- void **sendDriveCommand** (short v, short r) throw (CommandNotAvailable, InvalidArgument)

  *Drive the robot.*

- void **sendDriveCommand** (short v, **DriveCommand** dc) throw (CommandNotAvailable, InvalidArgument)

  *Drive the robot using a special drigin mode.*

- void **sendDriveDirectCommand** (short vr, short vl) throw (CommandNotAvailable, InvalidArgument)

  *Drive the two wheels separately.*

- void **sendLedCommand** (**Led** l, unsigned char c, unsigned char i) throw (CommandNotAvailable, InvalidArgument)

  *Change led status.*

- void **sendDigitalOutputsCommand** (bool d1, bool d2, bool d3) throw (CommandNotAvailable)

*Change the digital output values.*

- void **sendPwmLowSideDriversCommand** (unsigned char lsd1, unsigned char lsd2, unsigned char lsd3) throw (CommandNotAvailable, InvalidArgument)
  *Drive low side drivers with variable power.*
- void **sendLowSideDriversCommand** (bool lsd1, bool lsd2, bool lsd3) throw (CommandNotAvailable)
  *Switch on/off low side drivers.*
- void **sendIrCommand** (unsigned char v) throw (CommandNotAvailable)
  *Send an IR command on low side driver 1. The data will be encoded using the iRobot's* ***Create*** *(* p. *29) receiver format.*
- void **sendSongCommand** (unsigned char sid, const **song_t** &song) throw (CommandNotAvailable, InvalidArgument)
  *Define a song.*
- void **sendPlaySongCommand** (unsigned char sid) throw (CommandNotAvailable, InvalidArgument)
  *Play a song.*
- void **sendSensorsCommand** (**SensorPacket** sp) throw (CommandNotAvailable, InvalidArgument)
  *Request the robot to send a particular sensor packet.*
- void **sendQueryListCommand** (const **sensorPackets_t** &lsp) throw (CommandNotAvailable, InvalidArgument)
  *Request the robot to send a list of sensor packets.*
- void **sendStreamCommand** (const **sensorPackets_t** &lsp) throw (CommandNotAvailable, InvalidArgument)
  *Request the robot to start streaming some sensor packets. After the streaming is started, the requested values will be sent every 15ms.*
- void **sendPauseStreamCommand** (**StreamState** st) throw (CommandNotAvailable, InvalidArgument)
  *Pause or resume sensor streaming.*
- void **sendScriptCommand** (const **opcodes_t** &script) throw (CommandNotAvailable, InvalidArgument)
  *Send a script.*
- void **sendPlayScriptCommand** () throw (CommandNotAvailable)
  *Play the current stored script.*
- void **sendShowScriptCommand** () throw (CommandNotAvailable)
  *Show the stored script.*
- void **sendWaitTimeCommand** (unsigned char t) throw (CommandNotAvailable)
  *Make the robot wait for a specific amount of time.*
- void **sendWaitDistanceCommand** (short d) throw (CommandNotAvailable)
  *Make the robot wait until it travelled a certain distance.*
- void **sendWaitAngleCommand** (short a) throw (CommandNotAvailable)

*Make the robot wait until it has rotated enough.*

- void **sendWaitEventCommand** (**Event** e, **EventState** es=EVENT_OCCURRING) throw (CommandNotAvailable, InvalidArgument)

  *Make the robot wait for a specific event.*

- void **updateSensors** ()

  *Read the stream to update sensors values. This handles both streamed data and specific queries through querylist or sensors commands.*

- void **readStream** ()

  *Read streamed sensors and update sensors values.*

- **Mode mode** () throw ()
- bool **wheeldropCaster** ()
- bool **wheeldropLeft** ()
- bool **wheeldropRight** ()
- bool **bumpLeft** ()
- bool **bumpRight** ()
- bool **wall** ()

  *Get wall sensor value.*

- bool **cliffLeft** ()
- bool **cliffFrontLeft** ()
- bool **cliffFrontRight** ()
- bool **cliffRight** ()
- bool **deviceDetect** ()
- bool **digitalInput3** ()
- bool **digitalInput2** ()
- bool **digitalInput1** ()
- bool **digitalInput0** ()
- short **analogSignal** ()
- bool **homeBaseChargerAvailable** ()
- bool **internalChargerAvailable** ()
- bool **virtualWall** ()
- bool **leftWheelOvercurrent** ()
- bool **rightWheelOvercurrent** ()
- bool **ld2Overcurrent** ()
- bool **ld1Overcurrent** ()
- bool **ld0Overcurrent** ()
- unsigned char **ir** ()
- bool **advanceButton** ()
- bool **playButton** ()
- short **distance** ()
- short **angle** ()
- **ChargingState chargingState** ()
- short **batteryVoltage** ()
- short **batteryCurrent** ()
- short **batteryTemperature** ()
- short **batteryCharge** ()
- short **batteryCapacity** ()
- short **wallSignal** ()
- short **cliffLeftSignal** ()
- short **cliffFrontLeftSignal** ()
- short **cliffFrontRightSignal** ()

- short **cliffRightSignal** ()
- unsigned char **songNumber** ()
- bool **songPlaying** ()
- unsigned char **streamPackets** ()
- short **requestedVelocity** ()
- short **requestedRadius** ()
- short **requestedLeftVelocity** ()
- short **requestedRightVelocity** ()

### Static Public Attributes

- static const unsigned char **LED_COLOR_GREEN** = 0

  *Define the green color for power led.*

- static const unsigned char **LED_COLOR_RED** = 255

  *Define the red color for power led.*

- static const unsigned char **LED_INTENSITY_OFF** = 0

  *Define minimum intensity for power led (off).*

- static const unsigned char **LED_INTENSITY_FULL** = 255

  *Define full intensity for power led.*

- static const unsigned char **LOW_SIDE_VELOCITY_MIN** = 0

  *Define minimum velocity for low side drivers.*

- static const unsigned char **LOW_SIDE_VELOCITY_MAX** = 128

  *Define maximum velocity for low side drivers.*

- static const int **VELOCITY_MIN** = -500

  *Define minimum velocity for robot wheels motors.*

- static const int **VELOCITY_MAX** = 500

  *Define maximum velocity for robot wheels motors.*

- static const int **RADIUS_MIN** = -2000

  *Define minimum radius turn of the robot.*

- static const int **RADIUS_MAX** = 2000

  *Define maximum radius turn of the robot.*

- static const unsigned char **SONG_MIN** = 0

  *Define minimum song id.*

- static const unsigned char **SONG_MAX** = 15

  *Define maximum song id.*

- static const unsigned char **SONG_MAX_SIZE** = 16

  *Define song maximum size.*

- static const unsigned char **NO_NOTE** = 30

*Define the value for a rest node (no sound).*

- static const unsigned char **NOTE_MIN** = 31
  *Define the minimum note (G).*
- static const unsigned char **NOTE_MAX** = 127
  *Define the maximum note (G).*
- static const unsigned char **SCRIPT_MAX_SIZE** = 100
  *Define maximum script size.*
- static const unsigned char **STREAM_HEADER** = 19
  *Stream header "magic value".*
- static const unsigned char **SENSOR_BIT_WHEELDROP_CASTER** = 4
  *Bit used to retrieve the wheeldrop caster status.*
- static const unsigned char **SENSOR_BIT_WHEELDROP_LEFT** = 3
  *Bit used to retrieve the wheeldrop caster status.*
- static const unsigned char **SENSOR_BIT_WHEELDROP_RIGHT** = 2
  *Bit used to retrieve the wheeldrop caster status.*
- static const unsigned char **SENSOR_BIT_BUMP_LEFT** = 1
  *Bit used to retrieve the bump left status.*
- static const unsigned char **SENSOR_BIT_BUMP_RIGHT** = 0
  *Bit used to retrieve the bump right status.*
- static const unsigned char **SENSOR_BIT_LEFTWHEELOVERCURRENT** = 4
  *Bit used to retrieve the left wheel overcurrent status.*
- static const unsigned char **SENSOR_BIT_RIGHTWHEELOVERCURRENT** = 3
  *Bit used to retrieve the right wheel overcurrent status.*
- static const unsigned char **SENSOR_BIT_LD2OVERCURRENT** = 2
  *Bit used to retrieve the LD2 overcurrent status.*
- static const unsigned char **SENSOR_BIT_LD1OVERCURRENT** = 1
  *Bit used to retrieve the LD1 overcurrent status.*
- static const unsigned char **SENSOR_BIT_LD0OVERCURRENT** = 0
  *Bit used to retrieve the LD0 overcurrent status.*
- static const unsigned char **SENSOR_BIT_ADVANCEBUTTON** = 2
  *Bit used to retrieve the advance button status.*
- static const unsigned char **SENSOR_BIT_PLAYBUTTON** = 0
  *Bit used to retrieve the play button status.*

- static const unsigned char **SENSOR_BIT_DEVICEDETECT** = 4

  *Bit used to retrieve the device detect pin status.*

- static const unsigned char **SENSOR_BIT_DIGITALINPUT3** = 3

  *Bit used to retrieve the digital input 3 pin status.*

- static const unsigned char **SENSOR_BIT_DIGITALINPUT2** = 2

  *Bit used to retrieve the digital input 2 pin status.*

- static const unsigned char **SENSOR_BIT_DIGITALINPUT1** = 1

  *Bit used to retrieve the digital input 1 pin status.*

- static const unsigned char **SENSOR_BIT_DIGITALINPUT0** = 0

  *Bit used to retrieve the digital input 0 pin status.*

- static const unsigned char **SENSOR_BIT_HOMEBASECHARGERAVAILABLE** = 1

  *Bit used to retrieve whether the home base charger is available or not.*

- static const unsigned char **SENSOR_BIT_INTERNALCHARGERAVAILABLE** = 0

  *Bit used to retrieve whether the internal charger is available or not.*

## Protected Member Functions

- bool **readSensorPacket** ()

  *Read a specific sensor packet and update sensors values. The next expected sensor packet will be read.*

- bool **readSensorPacket** (**SensorPacket**, std::istream &)

  *Read sensor packet on a specified stream, update sensor values.*

- bool **readSensorPacket** ()

  *Read a specific sensor packet and update sensors values. The next expected sensor packet will be read.*

- bool **readSensorPacket** (**SensorPacket**, std::istream &)

  *Read sensor packet on a specified stream, update sensor values.*

### 6.2.1 Detailed Description

Robot communication class, wraps Open Interface version 2.

This class allows easy and safe communication with an **iRobot** (p. 13) **Create** (p. 29) robot. When this class is instanciated, the specified std::iostream will be used for the communication.

After being initialized, the object automatically set the robot into the passive mode.

### 6.2.2 Member Typedef Documentation

#### 6.2.2.1 typedef std::pair<unsigned char, unsigned char> iRobot::Create::note_t

Define a song note (frequency, duration).

#### 6.2.2.2 typedef std::vector<note_t> iRobot::Create::song_t

Define a song (a list of notes).

**See also:**

**note_t** (p. 47)

#### 6.2.2.3 typedef std::vector<unsigned char> iRobot::Create::opcodes_t

Define a vector of opcodes and arguments (a script).

**See also:**

**Opcode** (p. 67)

#### 6.2.2.4 typedef std::vector<SensorPacket> iRobot::Create::sensorPackets_t

Define a vector of sensor packets.

**See also:**

**sendQueryListCommand** (p. 92)
**sendStreamCommand** (p. 92)

#### 6.2.2.5 typedef std::queue<SensorPacket> iRobot::Create::queriedPackets_t

#### 6.2.2.6 typedef std::pair<unsigned char, unsigned char> iRobot::Create::note_t

Define a song note (frequency, duration).

#### 6.2.2.7 typedef std::vector<note_t> iRobot::Create::song_t

Define a song (a list of notes).

**See also:**

**note_t** (p. 47)

#### 6.2.2.8 typedef std::vector<unsigned char> iRobot::Create::opcodes_t

Define a vector of opcodes and arguments (a script).

**See also:**

**Opcode** (p. 67)

#### 6.2.2.9 typedef std::vector<SensorPacket> iRobot::Create::sensorPackets_t

Define a vector of sensor packets.

**See also:**

**sendQueryListCommand** (p. 92)
**sendStreamCommand** (p. 92)

#### 6.2.2.10 typedef std::queue<SensorPacket> iRobot::Create::queriedPackets_t

### 6.2.3 Member Enumeration Documentation

#### 6.2.3.1 enum iRobot::Create::Mode

**iRobot** (p. 13) **Create** (p. 29) modes The mode defines which commands are available and how the robot behaves when some events happens like wheel drop or if a cliff is detected.

**Enumerator:**

***IROBOT_CREATE_OFF*** Start-up mode, no command can be sent except start.

**See also:**

**Create::sendStartCommand** (p. 89)

***IROBOT_CREATE_PASSIVE*** Passive mode, actuators can not be controlled.

***IROBOT_CREATE_SAFE*** Safe mode, all commands are available. **iRobot** (p. 13) **Create** (p. 29) will stop if the wheels are dropped, if a cliff is detected or if the charger is plugged in and powered.

***IROBOT_CREATE_FULL*** Full mode, all commands are available. No safety check is done in this mode. The robot may fall from cliffs and will keep running if the wheels are dropped.

***IROBOT_CREATE_OFF*** Start-up mode, no command can be sent except start.

**See also:**

**Create::sendStartCommand** (p. 89)

***IROBOT_CREATE_PASSIVE*** Passive mode, actuators can not be controlled.

***IROBOT_CREATE_SAFE*** Safe mode, all commands are available. **iRobot** (p. 13) **Create** (p. 29) will stop if the wheels are dropped, if a cliff is detected or if the charger is plugged in and powered.

***IROBOT_CREATE_FULL*** Full mode, all commands are available. No safety check is done in this mode. The robot may fall from cliffs and will keep running if the wheels are dropped.

```
{
  IROBOT_CREATE_OFF,
  IROBOT_CREATE_PASSIVE,
  IROBOT_CREATE_SAFE,
  IROBOT_CREATE_FULL
};
```

#### 6.2.3.2 enum iRobot::Create::Baud

Enumerate all possible communication speeds.

**Enumerator:**

- ***BAUD_300*** Communicate at 300 bauds per second.
- ***BAUD_600*** Communicate at 600 bauds per second.
- ***BAUD_1200*** Communicate at 1200 bauds per second.
- ***BAUD_2400*** Communicate at 2400 bauds per second.
- ***BAUD_4800*** Communicate at 4800 bauds per second.
- ***BAUD_9600*** Communicate at 9600 bauds per second.
- ***BAUD_14400*** Communicate at 14400 bauds per second.
- ***BAUD_19200*** Communicate at 19200 bauds per second.
- ***BAUD_28800*** Communicate at 28800 bauds per second.
- ***BAUD_38400*** Communicate at 38400 bauds per second.
- ***BAUD_57600*** Communicate at 57600 bauds per second.
- ***BAUD_115200*** Communicate at 115200 bauds per second. In this mode, be sure to keep at least 15 µs between each command.
- ***BAUD_300*** Communicate at 300 bauds per second.
- ***BAUD_600*** Communicate at 600 bauds per second.
- ***BAUD_1200*** Communicate at 1200 bauds per second.
- ***BAUD_2400*** Communicate at 2400 bauds per second.
- ***BAUD_4800*** Communicate at 4800 bauds per second.
- ***BAUD_9600*** Communicate at 9600 bauds per second.
- ***BAUD_14400*** Communicate at 14400 bauds per second.
- ***BAUD_19200*** Communicate at 19200 bauds per second.
- ***BAUD_28800*** Communicate at 28800 bauds per second.
- ***BAUD_38400*** Communicate at 38400 bauds per second.
- ***BAUD_57600*** Communicate at 57600 bauds per second.
- ***BAUD_115200*** Communicate at 115200 bauds per second. In this mode, be sure to keep at least 15 µs between each command.

```
{
  BAUD_300 = 0,
  BAUD_600 = 1,
  BAUD_1200 = 2,
  BAUD_2400 = 3,
  BAUD_4800 = 4,
  BAUD_9600 = 5,
  BAUD_14400 = 6,
```

```
      BAUD_19200 = 7,
      BAUD_28800 = 8,
      BAUD_38400 = 9,
      BAUD_57600 = 10,
      BAUD_115200 = 11
    };
```

#### 6.2.3.3 enum iRobot::Create::Demo

Enumerate built-in demos.

**Enumerator:**

- ***DEMO_COVER*** **iRobot** (p. 13) **Create** (p. 29) covers an entire room using a combination of behaviors,such as bouncing off of walls, following walls, and spiraling.
- ***DEMO_COVER_AND_DOCK*** Identical to the Cover demo, with one exception; if **iRobot** (p. 13) **Create** (p. 29) sees the Home Base's∗ infrared signals, it uses these to move towards the Home Base and dock with it.
- ***DEMO_SPOT_COVER*** **iRobot** (p. 13) **Create** (p. 29) spirals outward, then inward, to cover an area around its starting position.
- ***DEMO_MOUSE*** **iRobot** (p. 13) **Create** (p. 29) tries to follow around the edges of a room using its wall sensor and bumper.
- ***DEMO_DRIVE_FIGURE_EIGHT*** **iRobot** (p. 13) **Create** (p. 29) continuously drives in a figure 8 pattern.
- ***DEMO_WIMP*** **iRobot** (p. 13) **Create** (p. 29) drives forward when pushed from behind. If **iRobot** (p. 13) **Create** (p. 29) hits an obstacle while driving, it drives away from the obstacle.
- ***DEMO_HOME*** **iRobot** (p. 13) **Create** (p. 29) drives toward a Virtual Wall when the back and sides of its Omnidirectional IR Receiver are covered with black electrical tape. When it touches the Virtual Wall or another obstacle, it stops.
- ***DEMO_TAG*** Identical to the Home demo, except **iRobot** (p. 13) **Create** (p. 29) goes back and forth between multiple Virtual Walls by bumping into one, turning around, driving to the next Virtual Wall, bumping into it and turning around to bump into the next Virtual Wall.
- ***DEMO_PACHELBEL*** **iRobot** (p. 13) **Create** (p. 29) plays the notes of Pachelbel's Canon in sequence when its cliff sensors are activated.
- ***DEMO_BANJO*** **iRobot** (p. 13) Create's four cliff sensors play the notes of a chord, depending on how the bumper is pressed: No bumper: G major Right or left bumper: D major7 Both bumpers (center): C major
- ***DEMO_ABORT*** Abort current running demo.
- ***DEMO_COVER*** **iRobot** (p. 13) **Create** (p. 29) covers an entire room using a combination of behaviors,such as bouncing off of walls, following walls, and spiraling.
- ***DEMO_COVER_AND_DOCK*** Identical to the Cover demo, with one exception; if **iRobot** (p. 13) **Create** (p. 29) sees the Home Base's∗ infrared signals, it uses these to move towards the Home Base and dock with it.
- ***DEMO_SPOT_COVER*** **iRobot** (p. 13) **Create** (p. 29) spirals outward, then inward, to cover an area around its starting position.
- ***DEMO_MOUSE*** **iRobot** (p. 13) **Create** (p. 29) tries to follow around the edges of a room using its wall sensor and bumper.

***DEMO_DRIVE_FIGURE_EIGHT*** **iRobot** (p. 13) **Create** (p. 29) continuously drives in a figure 8 pattern.

***DEMO_WIMP*** **iRobot** (p. 13) **Create** (p. 29) drives forward when pushed from behind. If **iRobot** (p. 13) **Create** (p. 29) hits an obstacle while driving, it drives away from the obstacle.

***DEMO_HOME*** **iRobot** (p. 13) **Create** (p. 29) drives toward a Virtual Wall when the back and sides of its Omnidirectional IR Receiver are covered with black electrical tape. When it touches the Virtual Wall or another obstacle, it stops.

***DEMO_TAG*** Identical to the Home demo, except **iRobot** (p. 13) **Create** (p. 29) goes back and forth between multiple Virtual Walls by bumping into one, turning around, driving to the next Virtual Wall, bumping into it and turning around to bump into the next Virtual Wall.

***DEMO_PACHELBEL*** **iRobot** (p. 13) **Create** (p. 29) plays the notes of Pachelbel's Canon in sequence when its cliff sensors are activated.

***DEMO_BANJO*** **iRobot** (p. 13) Create's four cliff sensors play the notes of a chord, depending on how the bumper is pressed: No bumper: G major Right or left bumper: D major7 Both bumpers (center): C major

***DEMO_ABORT*** Abort current running demo.

```
    {
      DEMO_COVER = 0,

      DEMO_COVER_AND_DOCK = 1,

      DEMO_SPOT_COVER = 2,

      DEMO_MOUSE = 3,

      DEMO_DRIVE_FIGURE_EIGHT = 4,

      DEMO_WIMP = 5,

      DEMO_HOME = 6,

      DEMO_TAG = 7,

      DEMO_PACHELBEL = 8,

      DEMO_BANJO = 9,

      DEMO_ABORT = 255
    };
```

#### 6.2.3.4 enum iRobot::Create::DriveCommand

Enumerates special driving commands.

**Enumerator:**

***DRIVE_STRAIGHT*** Drive straight.

***DRIVE_INPLACE_CLOCKWISE*** Turn in place clockwise.

***DRIVE_INPLACE_COUNTERCLOCKWISE*** Turn in place counter-clockwise.

***DRIVE_STRAIGHT*** Drive straight.

***DRIVE_INPLACE_CLOCKWISE*** Turn in place clockwise.

***DRIVE_INPLACE_COUNTERCLOCKWISE*** Turn in place counter-clockwise.

```
{
  DRIVE_STRAIGHT,
  DRIVE_INPLACE_CLOCKWISE,
  DRIVE_INPLACE_COUNTERCLOCKWISE
};
```

#### 6.2.3.5 enum iRobot::Create::Led

Enumerate available leds (and combinations). The power led is not in this list because it is handled differently (as its color is not fixed).

**Enumerator:**

***LED_NONE*** No led.

***LED_PLAY*** Play led only.

***LED_ADVANCE*** Advance led only.

***LED_ALL*** All leds.

***LED_NONE*** No led.

***LED_PLAY*** Play led only.

***LED_ADVANCE*** Advance led only.

***LED_ALL*** All leds.

```
{
  LED_NONE = 0,
  LED_PLAY = 2,
  LED_ADVANCE = 8,
  LED_ALL = LED_PLAY | LED_ADVANCE
};
```

#### 6.2.3.6 enum iRobot::Create::StreamState

Activate or disable sensor streaming.

**Enumerator:**

***STREAM_STATE_OFF*** Disable sensor streaming.

***STREAM_STATE_ON*** Activate sensor streaming.

***STREAM_STATE_OFF*** Disable sensor streaming.

***STREAM_STATE_ON*** Activate sensor streaming.

```
{
  STREAM_STATE_OFF = 0,
  STREAM_STATE_ON = 1
};
```

#### 6.2.3.7 enum iRobot::Create::ChargingState

Enumerate possible charging states.

**Enumerator:**

***CHARGING_STATE_NOT_CHARGING***
***CHARGING_STATE_RECONDITIONING_CHARGING***
***CHARGING_STATE_FULL_CHARGING***
***CHARGING_STATE_TRICKLE_CHARGING***
***CHARGING_STATE_WAITING***
***CHARGING_STATE_CHARGING_FAULT_CONDITION***
***CHARGING_STATE_NOT_CHARGING***
***CHARGING_STATE_RECONDITIONING_CHARGING***
***CHARGING_STATE_FULL_CHARGING***
***CHARGING_STATE_TRICKLE_CHARGING***
***CHARGING_STATE_WAITING***
***CHARGING_STATE_CHARGING_FAULT_CONDITION***

```
    {
      CHARGING_STATE_NOT_CHARGING = 0,
      CHARGING_STATE_RECONDITIONING_CHARGING = 1,
      CHARGING_STATE_FULL_CHARGING = 2,
      CHARGING_STATE_TRICKLE_CHARGING = 3,
      CHARGING_STATE_WAITING = 4,
      CHARGING_STATE_CHARGING_FAULT_CONDITION = 5

    };
```

#### 6.2.3.8 enum iRobot::Create::Event

Enumerate available events.

**Enumerator:**

***EVENT_WHEEL_DROP*** A wheel is dropped.
***EVENT_FRONT_WHEEL_DROP*** Front wheel is dropped.
***EVENT_LEFT_WHEEL_DROP*** Left wheel is dropped.
***EVENT_RIGHT_WHEEL_DROP*** Right wheel is dropped.
***EVENT_BUMP*** Front part has bumped.
***EVENT_LEFT_BUMP*** Left part has bumped.
***EVENT_RIGHT_BUMP*** Right part has bumped.
***EVENT_VIRTUAL_WALL*** A virtual wall has been detected.
***EVENT_WALL*** A wall has been detected.
***EVENT_CLIFF*** A cliff has been detected.
***EVENT_LEFT_CLIFF*** A cliff has been detected (left).
***EVENT_FRONT_LEFT_CLIFF*** A cliff has been detected (front left).
***EVENT_FRONT_RIGHT_CLIFF*** A cliff has been detected (front right).

***EVENT_RIGHT_CLIFF*** A cliff has been detected (right).

***EVENT_HOME_BASE*** Home base has been detected.

***EVENT_ADVANCE_BUTTON*** Advance button has been pushed.

***EVENT_PLAY_BUTTON*** Play button has been pushed.

***EVENT_DIGITAL_INPUT_0*** Digital input 0 has changed.

***EVENT_DIGITAL_INPUT_1*** Digital input 1 has changed.

***EVENT_DIGITAL_INPUT_2*** Digital input 2 has changed.

***EVENT_DIGITAL_INPUT_3*** Digital input 3 has changed.

***EVENT_OI_MODE_PASSIVE*** Robot has switched to passive mode.

***EVENT_WHEEL_DROP*** A wheel is dropped.

***EVENT_FRONT_WHEEL_DROP*** Front wheel is dropped.

***EVENT_LEFT_WHEEL_DROP*** Left wheel is dropped.

***EVENT_RIGHT_WHEEL_DROP*** Right wheel is dropped.

***EVENT_BUMP*** Front part has bumped.

***EVENT_LEFT_BUMP*** Left part has bumped.

***EVENT_RIGHT_BUMP*** Right part has bumped.

***EVENT_VIRTUAL_WALL*** A virtual wall has been detected.

***EVENT_WALL*** A wall has been detected.

***EVENT_CLIFF*** A cliff has been detected.

***EVENT_LEFT_CLIFF*** A cliff has been detected (left).

***EVENT_FRONT_LEFT_CLIFF*** A cliff has been detected (front left).

***EVENT_FRONT_RIGHT_CLIFF*** A cliff has been detected (front right).

***EVENT_RIGHT_CLIFF*** A cliff has been detected (right).

***EVENT_HOME_BASE*** Home base has been detected.

***EVENT_ADVANCE_BUTTON*** Advance button has been pushed.

***EVENT_PLAY_BUTTON*** Play button has been pushed.

***EVENT_DIGITAL_INPUT_0*** Digital input 0 has changed.

***EVENT_DIGITAL_INPUT_1*** Digital input 1 has changed.

***EVENT_DIGITAL_INPUT_2*** Digital input 2 has changed.

***EVENT_DIGITAL_INPUT_3*** Digital input 3 has changed.

***EVENT_OI_MODE_PASSIVE*** Robot has switched to passive mode.

```
    {
      EVENT_WHEEL_DROP = 1,
      EVENT_FRONT_WHEEL_DROP = 2,
      EVENT_LEFT_WHEEL_DROP = 3,
      EVENT_RIGHT_WHEEL_DROP = 4,
      EVENT_BUMP = 5,
      EVENT_LEFT_BUMP = 6,
      EVENT_RIGHT_BUMP = 7,
      EVENT_VIRTUAL_WALL = 8,
      EVENT_WALL = 9,
      EVENT_CLIFF = 10,
      EVENT_LEFT_CLIFF = 11,
      EVENT_FRONT_LEFT_CLIFF = 12,
      EVENT_FRONT_RIGHT_CLIFF = 13,
      EVENT_RIGHT_CLIFF = 14,
```

```
      EVENT_HOME_BASE = 15,
      EVENT_ADVANCE_BUTTON = 16,
      EVENT_PLAY_BUTTON = 17,
      EVENT_DIGITAL_INPUT_0 = 18,
      EVENT_DIGITAL_INPUT_1 = 19,
      EVENT_DIGITAL_INPUT_2 = 20,
      EVENT_DIGITAL_INPUT_3 = 21,
      EVENT_OI_MODE_PASSIVE = 22
    };
```

#### 6.2.3.9 enum iRobot::Create::EventState

Indicate an event state.

**Enumerator:**

- ***EVENT_OCCURRING*** The event is happening currently.
- ***EVENT_NOT_OCCURRING*** The event is not happening currently.
- ***EVENT_OCCURRING*** The event is happening currently.
- ***EVENT_NOT_OCCURRING*** The event is not happening currently.

```
    {
      EVENT_OCCURRING,
      EVENT_NOT_OCCURRING
    };
```

#### 6.2.3.10 enum iRobot::Create::Opcode

Enumerate available opcodes. Opcode are the basics instructions that the protocol support. This class wraps them to avoid using them directly. The only case where you have to use them is when you want to send an Open Interface script.

**Enumerator:**

- ***OPCODE_START*** Start command.
- ***OPCODE_BAUD*** Baud command.
- ***OPCODE_CONTROL*** Control command (equivalent to safe).
- ***OPCODE_SAFE*** Safe command.
- ***OPCODE_FULL*** Full command.
- ***OPCODE_SPOT*** Spot demo command.
- ***OPCODE_COVER*** Cover demo command.
- ***OPCODE_DEMO*** Demo command.
- ***OPCODE_DRIVE*** Driver command.
- ***OPCODE_LOW_SIDE_DRIVERS*** Low side drivers command.
- ***OPCODE_LEDS*** Leds command.
- ***OPCODE_SONG*** Song command.
- ***OPCODE_PLAY*** Play song command.
- ***OPCODE_SENSORS*** Sensors command.
- ***OPCODE_COVER_AND_DOCK*** Cover and dock demo command.

***OPCODE_PWM_LOW_SIDE_DRIVERS*** Pwm low side drivers command.

***OPCODE_DRIVE_DIRECT*** Driver direct command.

***OPCODE_DIGITAL_OUTPUTS*** Digital ouputs command.

***OPCODE_STREAM*** Stream command.

***OPCODE_QUERY_LIST*** Query list command.

***OPCODE_PAUSE_RESUME_STREAM*** Pause/resume stream command.

***OPCODE_SEND_IR*** Send IR command.

***OPCODE_SCRIPT*** Script command.

***OPCODE_PLAY_SCRIPT*** Play script command.

***OPCODE_SHOW_SCRIPT*** Show script command.

***OPCODE_WAIT_TIME*** Wait time command.

***OPCODE_WAIT_DISTANCE*** Wait distance command.

***OPCODE_WAIT_ANGLE*** Wait angle command.

***OPCODE_WAIT_EVENT*** Wait event command.

***OPCODE_START*** Start command.

***OPCODE_BAUD*** Baud command.

***OPCODE_CONTROL*** Control command (equivalent to safe).

***OPCODE_SAFE*** Safe command.

***OPCODE_FULL*** Full command.

***OPCODE_SPOT*** Spot demo command.

***OPCODE_COVER*** Cover demo command.

***OPCODE_DEMO*** Demo command.

***OPCODE_DRIVE*** Driver command.

***OPCODE_LOW_SIDE_DRIVERS*** Low side drivers command.

***OPCODE_LEDS*** Leds command.

***OPCODE_SONG*** Song command.

***OPCODE_PLAY*** Play song command.

***OPCODE_SENSORS*** Sensors command.

***OPCODE_COVER_AND_DOCK*** Cover and dock demo command.

***OPCODE_PWM_LOW_SIDE_DRIVERS*** Pwm low side drivers command.

***OPCODE_DRIVE_DIRECT*** Driver direct command.

***OPCODE_DIGITAL_OUTPUTS*** Digital ouputs command.

***OPCODE_STREAM*** Stream command.

***OPCODE_QUERY_LIST*** Query list command.

***OPCODE_PAUSE_RESUME_STREAM*** Pause/resume stream command.

***OPCODE_SEND_IR*** Send IR command.

***OPCODE_SCRIPT*** Script command.

***OPCODE_PLAY_SCRIPT*** Play script command.

***OPCODE_SHOW_SCRIPT*** Show script command.

***OPCODE_WAIT_TIME*** Wait time command.

***OPCODE_WAIT_DISTANCE*** Wait distance command.

***OPCODE_WAIT_ANGLE*** Wait angle command.

***OPCODE_WAIT_EVENT*** Wait event command.

```
    {
      OPCODE_START = 128,
      OPCODE_BAUD = 129,
      OPCODE_CONTROL = 130,
      OPCODE_SAFE = 131,
      OPCODE_FULL = 132,
      OPCODE_SPOT = 134,
      OPCODE_COVER = 135,
      OPCODE_DEMO = 136,
      OPCODE_DRIVE = 137,
      OPCODE_LOW_SIDE_DRIVERS = 138,
      OPCODE_LEDS = 139,
      OPCODE_SONG = 140,
      OPCODE_PLAY = 141,
      OPCODE_SENSORS = 142,
      OPCODE_COVER_AND_DOCK = 143,
      OPCODE_PWM_LOW_SIDE_DRIVERS = 144,
      OPCODE_DRIVE_DIRECT = 145,
      OPCODE_DIGITAL_OUTPUTS = 147,
      OPCODE_STREAM = 148,
      OPCODE_QUERY_LIST = 149,
      OPCODE_PAUSE_RESUME_STREAM = 150,
      OPCODE_SEND_IR = 151,
      OPCODE_SCRIPT = 152,
      OPCODE_PLAY_SCRIPT = 153,
      OPCODE_SHOW_SCRIPT = 154,
      OPCODE_WAIT_TIME = 155,
      OPCODE_WAIT_DISTANCE = 156,
      OPCODE_WAIT_ANGLE = 157,
      OPCODE_WAIT_EVENT = 158
    };
```

#### 6.2.3.11 enum iRobot::Create::SensorPacket

Enumerate available sensor packets.

**Enumerator:**

***SENSOR_GROUP_0*** Groups packets 7 to 26.

***SENSOR_GROUP_1*** Groups packets 7 to 16.

***SENSOR_GROUP_2*** Groups packets 17 to 20.

***SENSOR_GROUP_3*** Groups packets 21 to 26.

***SENSOR_GROUP_4*** Groups packets 27 to 34.

***SENSOR_GROUP_5*** Groups packets 35 to 42.

***SENSOR_GROUP_6*** Groups packets 7 to 42.

***SENSOR_BUMPS_WHEELS_DROPS*** Wheel and bumper states.

***SENSOR_WALL*** Wall sensor state.

***SENSOR_CLIFF_LEFT*** Left cliff sensor state.

***SENSOR_CLIFF_FRONT_LEFT*** Front left cliff sensor state.

***SENSOR_CLIFF_FRONT_RIGHT*** Front right cliff sensor state.

***SENSOR_CLIFF_RIGHT*** Right cliff sensor state.

***SENSOR_VIRTUAL_WALL*** Virtual wall sensor state.

***SENSOR_OVERCURRENTS*** Overcurrent sensors states.

***SENSOR_IR*** IR bytes received.

***SENSOR_BUTTONS*** Buttons states.

***SENSOR_DISTANCE*** Traveled distance since last read.

***SENSOR_ANGLE*** Turned angle since last read.

***SENSOR_CHARGING_STATE*** Charging state.

***SENSOR_VOLTAGE*** Battery voltage.

***SENSOR_CURRENT*** Batty current.

***SENSOR_BATTERY_TEMPERATURE*** Battery temperature.

***SENSOR_BATTERY_CHARGE*** Battery charge in milliamp-hours (mAh).

***SENSOR_BATTERY_CAPACITY*** Battery charge capacity in milliamp-hours (mAh).

***SENSOR_WALL_SIGNAL*** Wall's sensor signal strength.

***SENSOR_CLIFF_LEFT_SIGNAL*** Left cliff signal strength.

***SENSOR_CLIFF_FRONT_LEFT_SIGNAL*** Front left cliff signal strength.

***SENSOR_CLIFF_FRONT_RIGHT_SIGNAL*** Front right cliff signal strength.

***SENSOR_CLIFF_RIGHT_SIGNAL*** Right cliff signal strength.

***SENSOR_CARGO_BAY_DIGITAL_INPUT*** Cargo Bay digital input strength.

***SENSOR_CARGO_BAY_ANALOG_SIGNAL*** Cargo Bay analog input strength.

***SENSOR_CHARGING_SOURCES_AVAILABLE*** Available charging sources.

***SENSOR_OI_MODE*** Current Open Interface mode.

***SENSOR_SONG_NUMBER*** Current selected song.

***SENSOR_SONG_PLAYING*** Indicates whether or not a song is being played.

***SENSOR_NUMBER_STREAM_PACKETS*** List of streamed packets.

***SENSOR_REQUESTED_VELOCITY*** Requested velocity.

***SENSOR_REQUESTED_RADIUS*** Requested radius.

***SENSOR_REQUESTED_RIGHT_VELOCITY*** Requested right velocity.

***SENSOR_REQUESTED_LEFT_VELOCITY*** Requested left velocity.

***SENSOR_GROUP_0*** Groups packets 7 to 26.

***SENSOR_GROUP_1*** Groups packets 7 to 16.

***SENSOR_GROUP_2*** Groups packets 17 to 20.

***SENSOR_GROUP_3*** Groups packets 21 to 26.

***SENSOR_GROUP_4*** Groups packets 27 to 34.

***SENSOR_GROUP_5*** Groups packets 35 to 42.

***SENSOR_GROUP_6*** Groups packets 7 to 42.

***SENSOR_BUMPS_WHEELS_DROPS*** Wheel and bumper states.

***SENSOR_WALL*** Wall sensor state.

***SENSOR_CLIFF_LEFT*** Left cliff sensor state.

***SENSOR_CLIFF_FRONT_LEFT*** Front left cliff sensor state.

***SENSOR_CLIFF_FRONT_RIGHT*** Front right cliff sensor state.

***SENSOR_CLIFF_RIGHT*** Right cliff sensor state.

***SENSOR_VIRTUAL_WALL*** Virtual wall sensor state.

***SENSOR_OVERCURRENTS*** Overcurrent sensors states.

***SENSOR_IR*** IR bytes received.

***SENSOR_BUTTONS*** Buttons states.

***SENSOR_DISTANCE*** Traveled distance since last read.

***SENSOR_ANGLE*** Turned angle since last read.

***SENSOR_CHARGING_STATE*** Charging state.

***SENSOR_VOLTAGE*** Battery voltage.

***SENSOR_CURRENT*** Batty current.

***SENSOR_BATTERY_TEMPERATURE*** Battery temperature.

***SENSOR_BATTERY_CHARGE*** Battery charge in milliamp-hours (mAh).

***SENSOR_BATTERY_CAPACITY*** Battery charge capacity in milliamp-hours (mAh).

***SENSOR_WALL_SIGNAL*** Wall's sensor signal strength.

***SENSOR_CLIFF_LEFT_SIGNAL*** Left cliff signal strength.

***SENSOR_CLIFF_FRONT_LEFT_SIGNAL*** Front left cliff signal strength.

***SENSOR_CLIFF_FRONT_RIGHT_SIGNAL*** Front right cliff signal strength.

***SENSOR_CLIFF_RIGHT_SIGNAL*** Right cliff signal strength.

***SENSOR_CARGO_BAY_DIGITAL_INPUT*** Cargo Bay digital input strength.

***SENSOR_CARGO_BAY_ANALOG_SIGNAL*** Cargo Bay analog input strength.

***SENSOR_CHARGING_SOURCES_AVAILABLE*** Available charging sources.

***SENSOR_OI_MODE*** Current Open Interface mode.

***SENSOR_SONG_NUMBER*** Current selected song.

***SENSOR_SONG_PLAYING*** Indicates whether or not a song is being played.

***SENSOR_NUMBER_STREAM_PACKETS*** List of streamed packets.

***SENSOR_REQUESTED_VELOCITY*** Requested velocity.

***SENSOR_REQUESTED_RADIUS*** Requested radius.

***SENSOR_REQUESTED_RIGHT_VELOCITY*** Requested right velocity.

***SENSOR_REQUESTED_LEFT_VELOCITY*** Requested left velocity.

```
    {
      SENSOR_GROUP_0 = 0,
      SENSOR_GROUP_1 = 1,
      SENSOR_GROUP_2 = 2,
      SENSOR_GROUP_3 = 3,
      SENSOR_GROUP_4 = 4,
      SENSOR_GROUP_5 = 5,
      SENSOR_GROUP_6 = 6,
      SENSOR_BUMPS_WHEELS_DROPS = 7,
      SENSOR_WALL = 8,
      SENSOR_CLIFF_LEFT = 9,
      SENSOR_CLIFF_FRONT_LEFT = 10,
      SENSOR_CLIFF_FRONT_RIGHT = 11,
      SENSOR_CLIFF_RIGHT = 12,
      SENSOR_VIRTUAL_WALL = 13,
      SENSOR_OVERCURRENTS = 14,
      SENSOR_IR = 17,
      SENSOR_BUTTONS = 18,
      SENSOR_DISTANCE = 19,
```

```
        SENSOR_ANGLE = 20,
        SENSOR_CHARGING_STATE = 21,
        SENSOR_VOLTAGE = 22,
        SENSOR_CURRENT = 23,
        SENSOR_BATTERY_TEMPERATURE = 24,
        SENSOR_BATTERY_CHARGE = 25,
        SENSOR_BATTERY_CAPACITY = 26,
        SENSOR_WALL_SIGNAL = 27,
        SENSOR_CLIFF_LEFT_SIGNAL = 28,
        SENSOR_CLIFF_FRONT_LEFT_SIGNAL = 29,
        SENSOR_CLIFF_FRONT_RIGHT_SIGNAL = 30,
        SENSOR_CLIFF_RIGHT_SIGNAL = 31,
        SENSOR_CARGO_BAY_DIGITAL_INPUT = 32,
        SENSOR_CARGO_BAY_ANALOG_SIGNAL = 33,
        SENSOR_CHARGING_SOURCES_AVAILABLE = 34,
        SENSOR_OI_MODE = 35,
        SENSOR_SONG_NUMBER = 36,
        SENSOR_SONG_PLAYING = 37,
        SENSOR_NUMBER_STREAM_PACKETS = 38,
        SENSOR_REQUESTED_VELOCITY = 39,
        SENSOR_REQUESTED_RADIUS = 40,
        SENSOR_REQUESTED_RIGHT_VELOCITY = 41,
        SENSOR_REQUESTED_LEFT_VELOCITY = 42
      };
```

#### 6.2.3.12 enum iRobot::Create::Mode

**iRobot** (p. 13) **Create** (p. 29) modes The mode defines which commands are available and how the robot behaves when some events happens like wheel drop or if a cliff is detected.

**Enumerator:**

***IROBOT_CREATE_OFF*** Start-up mode, no command can be sent except start.

  **See also:**

    **Create::sendStartCommand** (p. 89)

***IROBOT_CREATE_PASSIVE*** Passive mode, actuators can not be controlled.

***IROBOT_CREATE_SAFE*** Safe mode, all commands are available. **iRobot** (p. 13) **Create** (p. 29) will stop if the wheels are dropped, if a cliff is detected or if the charger is plugged in and powered.

***IROBOT_CREATE_FULL*** Full mode, all commands are available. No safety check is done in this mode. The robot may fall from cliffs and will keep running if the wheels are dropped.

***IROBOT_CREATE_OFF*** Start-up mode, no command can be sent except start.

  **See also:**

    **Create::sendStartCommand** (p. 89)

***IROBOT_CREATE_PASSIVE*** Passive mode, actuators can not be controlled.

***IROBOT_CREATE_SAFE*** Safe mode, all commands are available. **iRobot** (p. 13) **Create** (p. 29) will stop if the wheels are dropped, if a cliff is detected or if the charger is plugged in and powered.

***IROBOT_CREATE_FULL*** Full mode, all commands are available. No safety check is done in this mode. The robot may fall from cliffs and will keep running if the wheels are dropped.

```
{
  IROBOT_CREATE_OFF,
  IROBOT_CREATE_PASSIVE,
  IROBOT_CREATE_SAFE,
  IROBOT_CREATE_FULL
};
```

#### 6.2.3.13 enum iRobot::Create::Baud

Enumerate all possible communication speeds.

**Enumerator:**

***BAUD_300*** Communicate at 300 bauds per second.

***BAUD_600*** Communicate at 600 bauds per second.

***BAUD_1200*** Communicate at 1200 bauds per second.

***BAUD_2400*** Communicate at 2400 bauds per second.

***BAUD_4800*** Communicate at 4800 bauds per second.

***BAUD_9600*** Communicate at 9600 bauds per second.

***BAUD_14400*** Communicate at 14400 bauds per second.

***BAUD_19200*** Communicate at 19200 bauds per second.

***BAUD_28800*** Communicate at 28800 bauds per second.

***BAUD_38400*** Communicate at 38400 bauds per second.

***BAUD_57600*** Communicate at 57600 bauds per second.

***BAUD_115200*** Communicate at 115200 bauds per second. In this mode, be sure to keep at least 15 µs between each command.

***BAUD_300*** Communicate at 300 bauds per second.

***BAUD_600*** Communicate at 600 bauds per second.

***BAUD_1200*** Communicate at 1200 bauds per second.

***BAUD_2400*** Communicate at 2400 bauds per second.

***BAUD_4800*** Communicate at 4800 bauds per second.

***BAUD_9600*** Communicate at 9600 bauds per second.

***BAUD_14400*** Communicate at 14400 bauds per second.

***BAUD_19200*** Communicate at 19200 bauds per second.

***BAUD_28800*** Communicate at 28800 bauds per second.

***BAUD_38400*** Communicate at 38400 bauds per second.

***BAUD_57600*** Communicate at 57600 bauds per second.

***BAUD_115200*** Communicate at 115200 bauds per second. In this mode, be sure to keep at least 15 µs between each command.

```
{
  BAUD_300 = 0,
  BAUD_600 = 1,
  BAUD_1200 = 2,
  BAUD_2400 = 3,
  BAUD_4800 = 4,
  BAUD_9600 = 5,
  BAUD_14400 = 6,
```

```
        BAUD_19200 = 7,
        BAUD_28800 = 8,
        BAUD_38400 = 9,
        BAUD_57600 = 10,
        BAUD_115200 = 11
    };
```

#### 6.2.3.14 enum iRobot::Create::Demo

Enumerate built-in demos.

**Enumerator:**

- ***DEMO_COVER*** **iRobot** (p. 13) **Create** (p. 29) covers an entire room using a combination of behaviors,such as bouncing off of walls, following walls, and spiraling.
- ***DEMO_COVER_AND_DOCK*** Identical to the Cover demo, with one exception; if **iRobot** (p. 13) **Create** (p. 29) sees the Home Base's∗ infrared signals, it uses these to move towards the Home Base and dock with it.
- ***DEMO_SPOT_COVER*** **iRobot** (p. 13) **Create** (p. 29) spirals outward, then inward, to cover an area around its starting position.
- ***DEMO_MOUSE*** **iRobot** (p. 13) **Create** (p. 29) tries to follow around the edges of a room using its wall sensor and bumper.
- ***DEMO_DRIVE_FIGURE_EIGHT*** **iRobot** (p. 13) **Create** (p. 29) continuously drives in a figure 8 pattern.
- ***DEMO_WIMP*** **iRobot** (p. 13) **Create** (p. 29) drives forward when pushed from behind. If **iRobot** (p. 13) **Create** (p. 29) hits an obstacle while driving, it drives away from the obstacle.
- ***DEMO_HOME*** **iRobot** (p. 13) **Create** (p. 29) drives toward a Virtual Wall when the back and sides of its Omnidirectional IR Receiver are covered with black electrical tape. When it touches the Virtual Wall or another obstacle, it stops.
- ***DEMO_TAG*** Identical to the Home demo, except **iRobot** (p. 13) **Create** (p. 29) goes back and forth between multiple Virtual Walls by bumping into one, turning around, driving to the next Virtual Wall, bumping into it and turning around to bump into the next Virtual Wall.
- ***DEMO_PACHELBEL*** **iRobot** (p. 13) **Create** (p. 29) plays the notes of Pachelbel's Canon in sequence when its cliff sensors are activated.
- ***DEMO_BANJO*** **iRobot** (p. 13) Create's four cliff sensors play the notes of a chord, depending on how the bumper is pressed: No bumper: G major Right or left bumper: D major7 Both bumpers (center): C major
- ***DEMO_ABORT*** Abort current running demo.
- ***DEMO_COVER*** **iRobot** (p. 13) **Create** (p. 29) covers an entire room using a combination of behaviors,such as bouncing off of walls, following walls, and spiraling.
- ***DEMO_COVER_AND_DOCK*** Identical to the Cover demo, with one exception; if **iRobot** (p. 13) **Create** (p. 29) sees the Home Base's∗ infrared signals, it uses these to move towards the Home Base and dock with it.
- ***DEMO_SPOT_COVER*** **iRobot** (p. 13) **Create** (p. 29) spirals outward, then inward, to cover an area around its starting position.
- ***DEMO_MOUSE*** **iRobot** (p. 13) **Create** (p. 29) tries to follow around the edges of a room using its wall sensor and bumper.

***DEMO_DRIVE_FIGURE_EIGHT*** **iRobot** (p. 13) **Create** (p. 29) continuously drives in a figure 8 pattern.

***DEMO_WIMP*** **iRobot** (p. 13) **Create** (p. 29) drives forward when pushed from behind. If **iRobot** (p. 13) **Create** (p. 29) hits an obstacle while driving, it drives away from the obstacle.

***DEMO_HOME*** **iRobot** (p. 13) **Create** (p. 29) drives toward a Virtual Wall when the back and sides of its Omnidirectional IR Receiver are covered with black electrical tape. When it touches the Virtual Wall or another obstacle, it stops.

***DEMO_TAG*** Identical to the Home demo, except **iRobot** (p. 13) **Create** (p. 29) goes back and forth between multiple Virtual Walls by bumping into one, turning around, driving to the next Virtual Wall, bumping into it and turning around to bump into the next Virtual Wall.

***DEMO_PACHELBEL*** **iRobot** (p. 13) **Create** (p. 29) plays the notes of Pachelbel's Canon in sequence when its cliff sensors are activated.

***DEMO_BANJO*** **iRobot** (p. 13) Create's four cliff sensors play the notes of a chord, depending on how the bumper is pressed: No bumper: G major Right or left bumper: D major7 Both bumpers (center): C major

***DEMO_ABORT*** Abort current running demo.

```
{
  DEMO_COVER = 0,

  DEMO_COVER_AND_DOCK = 1,

  DEMO_SPOT_COVER = 2,

  DEMO_MOUSE = 3,

  DEMO_DRIVE_FIGURE_EIGHT = 4,

  DEMO_WIMP = 5,

  DEMO_HOME = 6,

  DEMO_TAG = 7,

  DEMO_PACHELBEL = 8,

  DEMO_BANJO = 9,

  DEMO_ABORT = 255
};
```

#### 6.2.3.15 enum iRobot::Create::DriveCommand

Enumerates special driving commands.

**Enumerator:**

***DRIVE_STRAIGHT*** Drive straight.

***DRIVE_INPLACE_CLOCKWISE*** Turn in place clockwise.

***DRIVE_INPLACE_COUNTERCLOCKWISE*** Turn in place counter-clockwise.

***DRIVE_STRAIGHT*** Drive straight.

***DRIVE_INPLACE_CLOCKWISE*** Turn in place clockwise.

***DRIVE_INPLACE_COUNTERCLOCKWISE*** Turn in place counter-clockwise.

```
{
  DRIVE_STRAIGHT,
  DRIVE_INPLACE_CLOCKWISE,
  DRIVE_INPLACE_COUNTERCLOCKWISE
};
```

#### 6.2.3.16 enum iRobot::Create::Led

Enumerate available leds (and combinations). The power led is not in this list because it is handled differently (as its color is not fixed).

**Enumerator:**

***LED_NONE*** No led.

***LED_PLAY*** Play led only.

***LED_ADVANCE*** Advance led only.

***LED_ALL*** All leds.

***LED_NONE*** No led.

***LED_PLAY*** Play led only.

***LED_ADVANCE*** Advance led only.

***LED_ALL*** All leds.

```
{
  LED_NONE = 0,
  LED_PLAY = 2,
  LED_ADVANCE = 8,
  LED_ALL = LED_PLAY | LED_ADVANCE
};
```

#### 6.2.3.17 enum iRobot::Create::StreamState

Activate or disable sensor streaming.

**Enumerator:**

***STREAM_STATE_OFF*** Disable sensor streaming.

***STREAM_STATE_ON*** Activate sensor streaming.

***STREAM_STATE_OFF*** Disable sensor streaming.

***STREAM_STATE_ON*** Activate sensor streaming.

```
{
  STREAM_STATE_OFF = 0,
  STREAM_STATE_ON = 1
};
```

#### 6.2.3.18 enum iRobot::Create::ChargingState

Enumerate possible charging states.

**Enumerator:**

***CHARGING_STATE_NOT_CHARGING***

***CHARGING_STATE_RECONDITIONING_CHARGING***

***CHARGING_STATE_FULL_CHARGING***

***CHARGING_STATE_TRICKLE_CHARGING***

***CHARGING_STATE_WAITING***

***CHARGING_STATE_CHARGING_FAULT_CONDITION***

***CHARGING_STATE_NOT_CHARGING***

***CHARGING_STATE_RECONDITIONING_CHARGING***

***CHARGING_STATE_FULL_CHARGING***

***CHARGING_STATE_TRICKLE_CHARGING***

***CHARGING_STATE_WAITING***

***CHARGING_STATE_CHARGING_FAULT_CONDITION***

```
    {
      CHARGING_STATE_NOT_CHARGING = 0,
      CHARGING_STATE_RECONDITIONING_CHARGING = 1,
      CHARGING_STATE_FULL_CHARGING = 2,
      CHARGING_STATE_TRICKLE_CHARGING = 3,
      CHARGING_STATE_WAITING = 4,
      CHARGING_STATE_CHARGING_FAULT_CONDITION = 5

    };
```

#### 6.2.3.19 enum iRobot::Create::Event

Enumerate available events.

**Enumerator:**

***EVENT_WHEEL_DROP*** A wheel is dropped.

***EVENT_FRONT_WHEEL_DROP*** Front wheel is dropped.

***EVENT_LEFT_WHEEL_DROP*** Left wheel is dropped.

***EVENT_RIGHT_WHEEL_DROP*** Right wheel is dropped.

***EVENT_BUMP*** Front part has bumped.

***EVENT_LEFT_BUMP*** Left part has bumped.

***EVENT_RIGHT_BUMP*** Right part has bumped.

***EVENT_VIRTUAL_WALL*** A virtual wall has been detected.

***EVENT_WALL*** A wall has been detected.

***EVENT_CLIFF*** A cliff has been detected.

***EVENT_LEFT_CLIFF*** A cliff has been detected (left).

***EVENT_FRONT_LEFT_CLIFF*** A cliff has been detected (front left).

***EVENT_FRONT_RIGHT_CLIFF*** A cliff has been detected (front right).

***EVENT_RIGHT_CLIFF*** A cliff has been detected (right).

***EVENT_HOME_BASE*** Home base has been detected.

***EVENT_ADVANCE_BUTTON*** Advance button has been pushed.

***EVENT_PLAY_BUTTON*** Play button has been pushed.

***EVENT_DIGITAL_INPUT_0*** Digital input 0 has changed.

***EVENT_DIGITAL_INPUT_1*** Digital input 1 has changed.

***EVENT_DIGITAL_INPUT_2*** Digital input 2 has changed.

***EVENT_DIGITAL_INPUT_3*** Digital input 3 has changed.

***EVENT_OI_MODE_PASSIVE*** Robot has switched to passive mode.

***EVENT_WHEEL_DROP*** A wheel is dropped.

***EVENT_FRONT_WHEEL_DROP*** Front wheel is dropped.

***EVENT_LEFT_WHEEL_DROP*** Left wheel is dropped.

***EVENT_RIGHT_WHEEL_DROP*** Right wheel is dropped.

***EVENT_BUMP*** Front part has bumped.

***EVENT_LEFT_BUMP*** Left part has bumped.

***EVENT_RIGHT_BUMP*** Right part has bumped.

***EVENT_VIRTUAL_WALL*** A virtual wall has been detected.

***EVENT_WALL*** A wall has been detected.

***EVENT_CLIFF*** A cliff has been detected.

***EVENT_LEFT_CLIFF*** A cliff has been detected (left).

***EVENT_FRONT_LEFT_CLIFF*** A cliff has been detected (front left).

***EVENT_FRONT_RIGHT_CLIFF*** A cliff has been detected (front right).

***EVENT_RIGHT_CLIFF*** A cliff has been detected (right).

***EVENT_HOME_BASE*** Home base has been detected.

***EVENT_ADVANCE_BUTTON*** Advance button has been pushed.

***EVENT_PLAY_BUTTON*** Play button has been pushed.

***EVENT_DIGITAL_INPUT_0*** Digital input 0 has changed.

***EVENT_DIGITAL_INPUT_1*** Digital input 1 has changed.

***EVENT_DIGITAL_INPUT_2*** Digital input 2 has changed.

***EVENT_DIGITAL_INPUT_3*** Digital input 3 has changed.

***EVENT_OI_MODE_PASSIVE*** Robot has switched to passive mode.

```
    {
      EVENT_WHEEL_DROP = 1,
      EVENT_FRONT_WHEEL_DROP = 2,
      EVENT_LEFT_WHEEL_DROP = 3,
      EVENT_RIGHT_WHEEL_DROP = 4,
      EVENT_BUMP = 5,
      EVENT_LEFT_BUMP = 6,
      EVENT_RIGHT_BUMP = 7,
      EVENT_VIRTUAL_WALL = 8,
      EVENT_WALL = 9,
      EVENT_CLIFF = 10,
      EVENT_LEFT_CLIFF = 11,
      EVENT_FRONT_LEFT_CLIFF = 12,
      EVENT_FRONT_RIGHT_CLIFF = 13,
      EVENT_RIGHT_CLIFF = 14,
```

```
    EVENT_HOME_BASE = 15,
    EVENT_ADVANCE_BUTTON = 16,
    EVENT_PLAY_BUTTON = 17,
    EVENT_DIGITAL_INPUT_0 = 18,
    EVENT_DIGITAL_INPUT_1 = 19,
    EVENT_DIGITAL_INPUT_2 = 20,
    EVENT_DIGITAL_INPUT_3 = 21,
    EVENT_OI_MODE_PASSIVE = 22
  };
```

#### 6.2.3.20 enum iRobot::Create::EventState

Indicate an event state.

**Enumerator:**

- ***EVENT_OCCURRING*** The event is happening currently.
- ***EVENT_NOT_OCCURRING*** The event is not happening currently.
- ***EVENT_OCCURRING*** The event is happening currently.
- ***EVENT_NOT_OCCURRING*** The event is not happening currently.

```
  {
    EVENT_OCCURRING,
    EVENT_NOT_OCCURRING
  };
```

#### 6.2.3.21 enum iRobot::Create::Opcode

Enumerate available opcodes. Opcode are the basics instructions that the protocol support. This class wraps them to avoid using them directly. The only case where you have to use them is when you want to send an Open Interface script.

**Enumerator:**

- ***OPCODE_START*** Start command.
- ***OPCODE_BAUD*** Baud command.
- ***OPCODE_CONTROL*** Control command (equivalent to safe).
- ***OPCODE_SAFE*** Safe command.
- ***OPCODE_FULL*** Full command.
- ***OPCODE_SPOT*** Spot demo command.
- ***OPCODE_COVER*** Cover demo command.
- ***OPCODE_DEMO*** Demo command.
- ***OPCODE_DRIVE*** Driver command.
- ***OPCODE_LOW_SIDE_DRIVERS*** Low side drivers command.
- ***OPCODE_LEDS*** Leds command.
- ***OPCODE_SONG*** Song command.
- ***OPCODE_PLAY*** Play song command.
- ***OPCODE_SENSORS*** Sensors command.
- ***OPCODE_COVER_AND_DOCK*** Cover and dock demo command.

***OPCODE_PWM_LOW_SIDE_DRIVERS*** Pwm low side drivers command.

***OPCODE_DRIVE_DIRECT*** Driver direct command.

***OPCODE_DIGITAL_OUTPUTS*** Digital ouputs command.

***OPCODE_STREAM*** Stream command.

***OPCODE_QUERY_LIST*** Query list command.

***OPCODE_PAUSE_RESUME_STREAM*** Pause/resume stream command.

***OPCODE_SEND_IR*** Send IR command.

***OPCODE_SCRIPT*** Script command.

***OPCODE_PLAY_SCRIPT*** Play script command.

***OPCODE_SHOW_SCRIPT*** Show script command.

***OPCODE_WAIT_TIME*** Wait time command.

***OPCODE_WAIT_DISTANCE*** Wait distance command.

***OPCODE_WAIT_ANGLE*** Wait angle command.

***OPCODE_WAIT_EVENT*** Wait event command.

***OPCODE_START*** Start command.

***OPCODE_BAUD*** Baud command.

***OPCODE_CONTROL*** Control command (equivalent to safe).

***OPCODE_SAFE*** Safe command.

***OPCODE_FULL*** Full command.

***OPCODE_SPOT*** Spot demo command.

***OPCODE_COVER*** Cover demo command.

***OPCODE_DEMO*** Demo command.

***OPCODE_DRIVE*** Driver command.

***OPCODE_LOW_SIDE_DRIVERS*** Low side drivers command.

***OPCODE_LEDS*** Leds command.

***OPCODE_SONG*** Song command.

***OPCODE_PLAY*** Play song command.

***OPCODE_SENSORS*** Sensors command.

***OPCODE_COVER_AND_DOCK*** Cover and dock demo command.

***OPCODE_PWM_LOW_SIDE_DRIVERS*** Pwm low side drivers command.

***OPCODE_DRIVE_DIRECT*** Driver direct command.

***OPCODE_DIGITAL_OUTPUTS*** Digital ouputs command.

***OPCODE_STREAM*** Stream command.

***OPCODE_QUERY_LIST*** Query list command.

***OPCODE_PAUSE_RESUME_STREAM*** Pause/resume stream command.

***OPCODE_SEND_IR*** Send IR command.

***OPCODE_SCRIPT*** Script command.

***OPCODE_PLAY_SCRIPT*** Play script command.

***OPCODE_SHOW_SCRIPT*** Show script command.

***OPCODE_WAIT_TIME*** Wait time command.

***OPCODE_WAIT_DISTANCE*** Wait distance command.

***OPCODE_WAIT_ANGLE*** Wait angle command.

***OPCODE_WAIT_EVENT*** Wait event command.

```
{
  OPCODE_START = 128,
  OPCODE_BAUD = 129,
  OPCODE_CONTROL = 130,
  OPCODE_SAFE = 131,
  OPCODE_FULL = 132,
  OPCODE_SPOT = 134,
  OPCODE_COVER = 135,
  OPCODE_DEMO = 136,
  OPCODE_DRIVE = 137,
  OPCODE_LOW_SIDE_DRIVERS = 138,
  OPCODE_LEDS = 139,
  OPCODE_SONG = 140,
  OPCODE_PLAY = 141,
  OPCODE_SENSORS = 142,
  OPCODE_COVER_AND_DOCK = 143,
  OPCODE_PWM_LOW_SIDE_DRIVERS = 144,
  OPCODE_DRIVE_DIRECT = 145,
  OPCODE_DIGITAL_OUTPUTS = 147,
  OPCODE_STREAM = 148,
  OPCODE_QUERY_LIST = 149,
  OPCODE_PAUSE_RESUME_STREAM = 150,
  OPCODE_SEND_IR = 151,
  OPCODE_SCRIPT = 152,
  OPCODE_PLAY_SCRIPT = 153,
  OPCODE_SHOW_SCRIPT = 154,
  OPCODE_WAIT_TIME = 155,
  OPCODE_WAIT_DISTANCE = 156,
  OPCODE_WAIT_ANGLE = 157,
  OPCODE_WAIT_EVENT = 158
};
```

#### 6.2.3.22 enum iRobot::Create::SensorPacket

Enumerate available sensor packets.

**Enumerator:**

***SENSOR_GROUP_0*** Groups packets 7 to 26.

***SENSOR_GROUP_1*** Groups packets 7 to 16.

***SENSOR_GROUP_2*** Groups packets 17 to 20.

***SENSOR_GROUP_3*** Groups packets 21 to 26.

***SENSOR_GROUP_4*** Groups packets 27 to 34.

***SENSOR_GROUP_5*** Groups packets 35 to 42.

***SENSOR_GROUP_6*** Groups packets 7 to 42.

***SENSOR_BUMPS_WHEELS_DROPS*** Wheel and bumper states.

***SENSOR_WALL*** Wall sensor state.

***SENSOR_CLIFF_LEFT*** Left cliff sensor state.

***SENSOR_CLIFF_FRONT_LEFT*** Front left cliff sensor state.

***SENSOR_CLIFF_FRONT_RIGHT*** Front right cliff sensor state.

***SENSOR_CLIFF_RIGHT*** Right cliff sensor state.

***SENSOR_VIRTUAL_WALL*** Virtual wall sensor state.

***SENSOR_OVERCURRENTS*** Overcurrent sensors states.

***SENSOR_IR*** IR bytes received.

***SENSOR_BUTTONS*** Buttons states.

***SENSOR_DISTANCE*** Traveled distance since last read.

***SENSOR_ANGLE*** Turned angle since last read.

***SENSOR_CHARGING_STATE*** Charging state.

***SENSOR_VOLTAGE*** Battery voltage.

***SENSOR_CURRENT*** Batty current.

***SENSOR_BATTERY_TEMPERATURE*** Battery temperature.

***SENSOR_BATTERY_CHARGE*** Battery charge in milliamp-hours (mAh).

***SENSOR_BATTERY_CAPACITY*** Battery charge capacity in milliamp-hours (mAh).

***SENSOR_WALL_SIGNAL*** Wall's sensor signal strength.

***SENSOR_CLIFF_LEFT_SIGNAL*** Left cliff signal strength.

***SENSOR_CLIFF_FRONT_LEFT_SIGNAL*** Front left cliff signal strength.

***SENSOR_CLIFF_FRONT_RIGHT_SIGNAL*** Front right cliff signal strength.

***SENSOR_CLIFF_RIGHT_SIGNAL*** Right cliff signal strength.

***SENSOR_CARGO_BAY_DIGITAL_INPUT*** Cargo Bay digital input strength.

***SENSOR_CARGO_BAY_ANALOG_SIGNAL*** Cargo Bay analog input strength.

***SENSOR_CHARGING_SOURCES_AVAILABLE*** Available charging sources.

***SENSOR_OI_MODE*** Current Open Interface mode.

***SENSOR_SONG_NUMBER*** Current selected song.

***SENSOR_SONG_PLAYING*** Indicates whether or not a song is being played.

***SENSOR_NUMBER_STREAM_PACKETS*** List of streamed packets.

***SENSOR_REQUESTED_VELOCITY*** Requested velocity.

***SENSOR_REQUESTED_RADIUS*** Requested radius.

***SENSOR_REQUESTED_RIGHT_VELOCITY*** Requested right velocity.

***SENSOR_REQUESTED_LEFT_VELOCITY*** Requested left velocity.

***SENSOR_GROUP_0*** Groups packets 7 to 26.

***SENSOR_GROUP_1*** Groups packets 7 to 16.

***SENSOR_GROUP_2*** Groups packets 17 to 20.

***SENSOR_GROUP_3*** Groups packets 21 to 26.

***SENSOR_GROUP_4*** Groups packets 27 to 34.

***SENSOR_GROUP_5*** Groups packets 35 to 42.

***SENSOR_GROUP_6*** Groups packets 7 to 42.

***SENSOR_BUMPS_WHEELS_DROPS*** Wheel and bumper states.

***SENSOR_WALL*** Wall sensor state.

***SENSOR_CLIFF_LEFT*** Left cliff sensor state.

***SENSOR_CLIFF_FRONT_LEFT*** Front left cliff sensor state.

***SENSOR_CLIFF_FRONT_RIGHT*** Front right cliff sensor state.

***SENSOR_CLIFF_RIGHT*** Right cliff sensor state.

***SENSOR_VIRTUAL_WALL*** Virtual wall sensor state.

***SENSOR_OVERCURRENTS*** Overcurrent sensors states.

***SENSOR_IR*** IR bytes received.

***SENSOR_BUTTONS*** Buttons states.

***SENSOR_DISTANCE*** Traveled distance since last read.

***SENSOR_ANGLE*** Turned angle since last read.

***SENSOR_CHARGING_STATE*** Charging state.

***SENSOR_VOLTAGE*** Battery voltage.

***SENSOR_CURRENT*** Batty current.

***SENSOR_BATTERY_TEMPERATURE*** Battery temperature.

***SENSOR_BATTERY_CHARGE*** Battery charge in milliamp-hours (mAh).

***SENSOR_BATTERY_CAPACITY*** Battery charge capacity in milliamp-hours (mAh).

***SENSOR_WALL_SIGNAL*** Wall's sensor signal strength.

***SENSOR_CLIFF_LEFT_SIGNAL*** Left cliff signal strength.

***SENSOR_CLIFF_FRONT_LEFT_SIGNAL*** Front left cliff signal strength.

***SENSOR_CLIFF_FRONT_RIGHT_SIGNAL*** Front right cliff signal strength.

***SENSOR_CLIFF_RIGHT_SIGNAL*** Right cliff signal strength.

***SENSOR_CARGO_BAY_DIGITAL_INPUT*** Cargo Bay digital input strength.

***SENSOR_CARGO_BAY_ANALOG_SIGNAL*** Cargo Bay analog input strength.

***SENSOR_CHARGING_SOURCES_AVAILABLE*** Available charging sources.

***SENSOR_OI_MODE*** Current Open Interface mode.

***SENSOR_SONG_NUMBER*** Current selected song.

***SENSOR_SONG_PLAYING*** Indicates whether or not a song is being played.

***SENSOR_NUMBER_STREAM_PACKETS*** List of streamed packets.

***SENSOR_REQUESTED_VELOCITY*** Requested velocity.

***SENSOR_REQUESTED_RADIUS*** Requested radius.

***SENSOR_REQUESTED_RIGHT_VELOCITY*** Requested right velocity.

***SENSOR_REQUESTED_LEFT_VELOCITY*** Requested left velocity.

```
    {
      SENSOR_GROUP_0 = 0,
      SENSOR_GROUP_1 = 1,
      SENSOR_GROUP_2 = 2,
      SENSOR_GROUP_3 = 3,
      SENSOR_GROUP_4 = 4,
      SENSOR_GROUP_5 = 5,
      SENSOR_GROUP_6 = 6,
      SENSOR_BUMPS_WHEELS_DROPS = 7,
      SENSOR_WALL = 8,
      SENSOR_CLIFF_LEFT = 9,
      SENSOR_CLIFF_FRONT_LEFT = 10,
      SENSOR_CLIFF_FRONT_RIGHT = 11,
      SENSOR_CLIFF_RIGHT = 12,
      SENSOR_VIRTUAL_WALL = 13,
      SENSOR_OVERCURRENTS = 14,
      SENSOR_IR = 17,
      SENSOR_BUTTONS = 18,
      SENSOR_DISTANCE = 19,
```

```
      SENSOR_ANGLE = 20,
      SENSOR_CHARGING_STATE = 21,
      SENSOR_VOLTAGE = 22,
      SENSOR_CURRENT = 23,
      SENSOR_BATTERY_TEMPERATURE = 24,
      SENSOR_BATTERY_CHARGE = 25,
      SENSOR_BATTERY_CAPACITY = 26,
      SENSOR_WALL_SIGNAL = 27,
      SENSOR_CLIFF_LEFT_SIGNAL = 28,
      SENSOR_CLIFF_FRONT_LEFT_SIGNAL = 29,
      SENSOR_CLIFF_FRONT_RIGHT_SIGNAL = 30,
      SENSOR_CLIFF_RIGHT_SIGNAL = 31,
      SENSOR_CARGO_BAY_DIGITAL_INPUT = 32,
      SENSOR_CARGO_BAY_ANALOG_SIGNAL = 33,
      SENSOR_CHARGING_SOURCES_AVAILABLE = 34,
      SENSOR_OI_MODE = 35,
      SENSOR_SONG_NUMBER = 36,
      SENSOR_SONG_PLAYING = 37,
      SENSOR_NUMBER_STREAM_PACKETS = 38,
      SENSOR_REQUESTED_VELOCITY = 39,
      SENSOR_REQUESTED_RADIUS = 40,
      SENSOR_REQUESTED_RIGHT_VELOCITY = 41,
      SENSOR_REQUESTED_LEFT_VELOCITY = 42
    };
```

### 6.2.4 Constructor & Destructor Documentation

#### 6.2.4.1 iRobot::Create::Create (std::iostream & *stream*) throw (InvalidArgument) [explicit]

Construct an instance of **Create** (p. 29).

**Parameters:**

*stream* Stream used for communication.

```
    : currentMode_ (IROBOT_CREATE_OFF),
      stream_ (stream),
      INITIALIZE_SENSORS ()
  {
    init ();
  }
```

#### 6.2.4.2 iRobot::Create::Create (LibSerial::SerialStream & *stream*) throw (InvalidArgument, LibSerialNotAvailable) [explicit]

Construct an instance of **Create** (p. 29) using serial port communication.

**Parameters:**

*stream* Stream used for communication.

```
  : stream_ ((std::iostream&)stream)
  {
    throw LibSerialNotAvailable ();
    assert (0);
  }
```

#### 6.2.4.3 iRobot::Create::∼Create () throw ()

Destroy an instance of **Create** (p. 29).

```
{}
```

#### 6.2.4.4 iRobot::Create::Create (std::iostream & *stream*) throw (InvalidArgument) [explicit]

Construct an instance of **Create** (p. 29).

**Parameters:**

*stream* Stream used for communication.

#### 6.2.4.5 iRobot::Create::Create (LibSerial::SerialStream & *stream*) throw (InvalidArgument, LibSerialNotAvailable) [explicit]

Construct an instance of **Create** (p. 29) using serial port communication.

**Parameters:**

*stream* Stream used for communication.

#### 6.2.4.6 iRobot::Create::∼Create () throw ()

Destroy an instance of **Create** (p. 29).

### 6.2.5 Member Function Documentation

#### 6.2.5.1 void iRobot::Create::sendBaudCommand (Baud *speed*) throw (CommandNotAvailable, InvalidArgument)

Change the communication speed. After changing the communication speed, it is required to wait at least 100ms.

**Parameters:**

*speed* New communication speed.

HAVE_LIBSERIAL

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();
  if (baud < 0 || baud > BAUD_115200)
    throw InvalidArgument ();
  const unsigned char op  = OPCODE_BAUD;
  const unsigned char b = baud;
  stream_ << op << b;
  stream_.flush ();
```

```

#ifdef HAVE_LIBSERIAL
    {
      LibSerial::SerialStream* stream =
        dynamic_cast<LibSerial::SerialStream*> (&stream_);
      if (!!stream)
        stream->SetBaudRate (toLibSerialBaud (baud));
    }
#endif

    usleep (100 * 1000);
  }
```

#### 6.2.5.2 void iRobot::Create::sendStartCommand ()

Switch to passive mode. Automatically done when the class is instanciated.

```
  {
    if (currentMode_ != IROBOT_CREATE_OFF)
      return;
    const unsigned char op  = OPCODE_START;

    stream_ << op;
    stream_.flush ();
    currentMode_ = IROBOT_CREATE_PASSIVE;
  }
```

#### 6.2.5.3 void iRobot::Create::sendSafeCommand () throw (CommandNotAvailable)

Switch to safe mode.

```
  {
    if (currentMode_ < IROBOT_CREATE_PASSIVE)
      throw CommandNotAvailable ();
    const unsigned char op  = OPCODE_SAFE;
    stream_ << op;
    stream_.flush ();
    currentMode_ = IROBOT_CREATE_SAFE;
  }
```

#### 6.2.5.4 void iRobot::Create::sendFullCommand () throw (CommandNotAvailable)

Switch to full mode.

```
  {
    if (currentMode_ < IROBOT_CREATE_PASSIVE)
      throw CommandNotAvailable ();
    const unsigned char op  = OPCODE_FULL;
    stream_ << op;
    stream_.flush ();
    currentMode_ = IROBOT_CREATE_FULL;
  }
```

#### 6.2.5.5 void iRobot::Create::sendDemoCommand (Demo *demo*) throw (CommandNotAvailable, InvalidArgument)

Run a built-in demo.

**Parameters:**

***demo*** Demo id.

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();
  if (demo != DEMO_ABORT && (demo < 0 || demo > DEMO_BANJO))
    throw InvalidArgument ();
  const unsigned char op  = OPCODE_DEMO;
  const unsigned char d = demo;
  stream_ << op << d;
  stream_.flush ();
  currentMode_ = IROBOT_CREATE_PASSIVE;
}
```

#### 6.2.5.6 void iRobot::Create::sendDriveCommand (short *v*, short *r*) throw (CommandNotAvailable, InvalidArgument)

Drive the robot.

**Parameters:**

***v*** Robot's velocity.

***r*** Robot's turning radius.

**See also:**

**sendDriveDirectCommand** (p. 90)

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();
  if (velocity < VELOCITY_MIN || velocity > VELOCITY_MAX)
    throw InvalidArgument ();
  if (radius < RADIUS_MIN || radius > RADIUS_MAX)
    throw InvalidArgument ();

  const unsigned char op  = OPCODE_DRIVE;

  stream_ << op;
  sendInt16 (stream_, velocity);
  sendInt16 (stream_, radius);
  stream_.flush ();
}
```

#### 6.2.5.7 void iRobot::Create::sendDriveCommand (short *v*, DriveCommand *dc*) throw (CommandNotAvailable, InvalidArgument)

Drive the robot using a special drigin mode.

**Parameters:**

***v*** Robot's velocity.

***dc*** Special driving mode.

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();
  if (velocity < VELOCITY_MIN || velocity > VELOCITY_MAX)
    throw InvalidArgument ();

  int arg = 0;
  switch (cmd)
    {
    case DRIVE_STRAIGHT:
      arg = 0x8000;
      break;
    case DRIVE_INPLACE_CLOCKWISE:
      arg = 0xFFFF;
      break;
    case DRIVE_INPLACE_COUNTERCLOCKWISE:
      arg = 0x0001;
      break;
    default:
      throw InvalidArgument ();
    }

  const unsigned char op  = OPCODE_DRIVE;
  stream_ << op;
  sendInt16 (stream_, velocity);
  sendInt16 (stream_, arg);
  stream_.flush ();
}
```

#### 6.2.5.8 void iRobot::Create::sendDriveDirectCommand (short *vr*, short *vl*) throw (CommandNotAvailable, InvalidArgument)

Drive the two wheels separately.

**Parameters:**

***vr*** Right wheel's velocity.

***vl*** Left wheel's velocity.

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();
  if (velocityL < VELOCITY_MIN || velocityL > VELOCITY_MAX)
    throw InvalidArgument ();
  if (velocityR < VELOCITY_MIN || velocityR > VELOCITY_MAX)
    throw InvalidArgument ();

  const unsigned char op  = OPCODE_DRIVE_DIRECT;
  stream_ << op;
  sendInt16 (stream_, velocityR);
  sendInt16 (stream_, velocityL);
  stream_.flush ();
}
```

#### 6.2.5.9 void iRobot::Create::sendLedCommand (Led *l*, unsigned char *c*, unsigned char *i*) throw (CommandNotAvailable, InvalidArgument)

Change led status.

**Parameters:**

***l*** Switch on/off advance or play leds.

***c*** Power led color.

***i*** Power led light intensity.

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();
  if (ledBits < LED_NONE || ledBits > LED_ALL)
    throw InvalidArgument ();

  const unsigned char op  = OPCODE_LEDS;
  const unsigned char b = ledBits;
  stream_ << op << b << powerColor << powerIntensity;
  stream_.flush ();
}
```

#### 6.2.5.10 void iRobot::Create::sendDigitalOutputsCommand (bool *d1*, bool *d2*, bool *d3*) throw (CommandNotAvailable)

Change the digital output values.

**Parameters:**

***d1*** Switch on/off first digital output.

***d2*** Switch on/off second digital output.

***d3*** Switch on/off third digital output.

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();

  const unsigned char op  = OPCODE_DIGITAL_OUTPUTS;
  const unsigned char v = d0 + d1 * 2 + d2 * 4;
  stream_ << op << v;
  stream_.flush ();
}
```

#### 6.2.5.11 void iRobot::Create::sendPwmLowSideDriversCommand (unsigned char *lsd1*, unsigned char *lsd2*, unsigned char *lsd3*) throw (CommandNotAvailable, InvalidArgument)

Drive low side drivers with variable power.

**Parameters:**

***lsd1*** Low side driver 1 power.

***lsd2*** Low side driver 2 power.

***lsd3*** Low side driver 3 power.

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();
  if (d1 > LOW_SIDE_VELOCITY_MAX
```

```
      || d2 > LOW_SIDE_VELOCITY_MAX
      || d3 > LOW_SIDE_VELOCITY_MAX)
    throw InvalidArgument ();

  const unsigned char op  = OPCODE_PWM_LOW_SIDE_DRIVERS;
  stream_ << op << d1 << d2 << d3;
  stream_.flush ();
}
```

#### 6.2.5.12 void iRobot::Create::sendLowSideDriversCommand (bool *lsd1*, bool *lsd2*, bool *lsd3*) throw (CommandNotAvailable)

Switch on/off low side drivers.

**Parameters:**

***lsd1*** Switch on/off low side driver 1.

***lsd2*** Switch on/off low side driver 2.

***lsd3*** Switch on/off low side driver 3.

**See also:**

**sendPwmLowSideDriversCommand** (p. 91)

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();

  const unsigned char op  = OPCODE_LOW_SIDE_DRIVERS;
  const unsigned char v = ld0 + ld1 * 2 + sd * 4;
  stream_ << op << v;
  stream_.flush ();
}
```

#### 6.2.5.13 void iRobot::Create::sendIrCommand (unsigned char *v*) throw (CommandNotAvailable)

Send an IR command on low side driver 1. The data will be encoded using the iRobot's **Create** (p. 29) receiver format.

**Parameters:**

***v*** Sent value.

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();

  const unsigned char op  = OPCODE_SEND_IR;
  stream_ << op << v;
  stream_.flush ();
}
```

#### 6.2.5.14 void iRobot::Create::sendSongCommand (unsigned char *sid*, const song_t & *song*) throw (CommandNotAvailable, InvalidArgument)

Define a song.

**Parameters:**

***sid*** Song id.

***song*** Song to send.

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();
  if (songN > SONG_MAX)
    throw InvalidArgument ();
  if (song.size () > SONG_MAX_SIZE)
    throw InvalidArgument ();

  std::stringstream ss;
  for (song_t::const_iterator it = song.begin (); it != song.end (); ++it)
    {
      if (it->first < NO_NOTE || it->first > NOTE_MAX)
        throw InvalidArgument ();
      ss << it->first << it->second;
    }
  const unsigned char op  = OPCODE_SONG;
  const unsigned char size = song.size ();
  stream_ << op << songN << size << ss.str ();
  stream_.flush ();
}
```

#### 6.2.5.15 void iRobot::Create::sendPlaySongCommand (unsigned char *sid*) throw (CommandNotAvailable, InvalidArgument)

Play a song.

**Parameters:**

***sid*** Sond id to play.

```
{
  if (currentMode_ < IROBOT_CREATE_SAFE)
    throw CommandNotAvailable ();
  if (v > SONG_MAX)
    throw InvalidArgument ();

  const unsigned char op  = OPCODE_PLAY;
  stream_ << op << v;
  stream_.flush ();
}
```

#### 6.2.5.16 void iRobot::Create::sendSensorsCommand (SensorPacket *sp*) throw (CommandNotAvailable, InvalidArgument)

Request the robot to send a particular sensor packet.

**Parameters:**

***sp*** Sensor packet to send.

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();
  if (packet < SENSOR_GROUP_0
      || packet > SENSOR_REQUESTED_LEFT_VELOCITY)
    throw InvalidArgument ();

  const unsigned char op  = OPCODE_SENSORS;
  const unsigned char p = packet;
  stream_ << op << p;

  queriedSensors_.push (packet);
  stream_.flush ();
}
```

#### 6.2.5.17 void iRobot::Create::sendQueryListCommand (const sensorPackets_t & *lsp*) throw (CommandNotAvailable, InvalidArgument)

Request the robot to send a list of sensor packets.

**Parameters:**

***lsp*** List of sensor packets to send

```
{
  MAKE_SENSOR_CMD (OPCODE_QUERY_LIST,
                   queriedSensors_.push (*it));
}
```

#### 6.2.5.18 void iRobot::Create::sendStreamCommand (const sensorPackets_t & *lsp*) throw (CommandNotAvailable, InvalidArgument)

Request the robot to start streaming some sensor packets. After the streaming is started, the requested values will be sent every 15ms.

**Parameters:**

***lsp*** List of sensor packets to stream.

```
{
  MAKE_SENSOR_CMD (OPCODE_STREAM,
                   streamedSensors_.push_back (*it));
}
```

#### 6.2.5.19 void iRobot::Create::sendPauseStreamCommand (StreamState *st*) throw (CommandNotAvailable, InvalidArgument)

Pause or resume sensor streaming.

**Parameters:**

***st*** Stream state to set.

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();
  if (state < 0 || state > STREAM_STATE_ON)
    throw InvalidArgument ();

  const unsigned char op  = OPCODE_PAUSE_RESUME_STREAM;
  const unsigned char st = state;
  stream_ << op << st;
  stream_.flush ();
}
```

#### 6.2.5.20 void iRobot::Create::sendScriptCommand (const opcodes_t & *script*) throw (CommandNotAvailable, InvalidArgument)

Send a script.

**Parameters:**

***script*** Script to send.

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();
  if (opcodes.size () > SCRIPT_MAX_SIZE)
    throw InvalidArgument ();

  std::stringstream ss;
  for (opcodes_t::const_iterator it = opcodes.begin ();
       it != opcodes.end (); ++it)
    {
      if (*it < OPCODE_START
          || *it > OPCODE_WAIT_EVENT)
        throw InvalidArgument ();
      const unsigned char opcode = *it;
      ss << opcode;
    }

  const unsigned char op = OPCODE_SCRIPT;
  const unsigned char size = opcodes.size ();
  stream_ << op << size << ss.str ();
  stream_.flush ();
}
```

#### 6.2.5.21 void iRobot::Create::sendPlayScriptCommand () throw (CommandNotAvailable)

Play the current stored script.

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();

  const unsigned char op  = OPCODE_PLAY_SCRIPT;
  stream_ << op;
  stream_.flush ();
}
```

#### 6.2.5.22 void iRobot::Create::sendShowScriptCommand () throw (CommandNotAvailable)

Show the stored script.

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();

  const unsigned char op  = OPCODE_SHOW_SCRIPT;
  stream_ << op;
  stream_.flush ();
}
```

#### 6.2.5.23 void iRobot::Create::sendWaitTimeCommand (unsigned char *t*) throw (CommandNotAvailable)

Make the robot wait for a specific amount of time.

**Parameters:**

***t*** Time to wait in tenth of a second (resolution of 15 ms).

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();

  const unsigned char op  = OPCODE_WAIT_TIME;
  stream_ << op << t;
  stream_.flush ();
}
```

#### 6.2.5.24 void iRobot::Create::sendWaitDistanceCommand (short *d*) throw (CommandNotAvailable)

Make the robot wait until it travelled a certain distance.

**Parameters:**

***d*** Distance to travel

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();

  const unsigned char op = OPCODE_WAIT_DISTANCE;
  stream_ << op;
  sendInt16 (stream_, d);
  stream_.flush ();
}
```

#### 6.2.5.25 void iRobot::Create::sendWaitAngleCommand (short *a*) throw (CommandNotAvailable)

Make the robot wait until it has rotated enough.

**Parameters:**

*a* Angle (in degrees)

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();

  const unsigned char op  = OPCODE_WAIT_ANGLE;
  stream_ << op;
  sendInt16 (stream_, a);
  stream_.flush ();
}
```

#### 6.2.5.26 void iRobot::Create::sendWaitEventCommand (Event *e*, EventState *es* = EVENT_OCCURRING) throw (CommandNotAvailable, InvalidArgument)

Make the robot wait for a specific event.

**Parameters:**

*e* Wait for this event.

*es* Wait for this event state.

```
{
  if (currentMode_ < IROBOT_CREATE_PASSIVE)
    throw CommandNotAvailable ();
  if (event < EVENT_WHEEL_DROP || event > EVENT_OI_MODE_PASSIVE)
    throw InvalidArgument ();
  if (state < EVENT_OCCURRING || state > EVENT_NOT_OCCURRING)
    throw InvalidArgument ();

  const unsigned char op  = OPCODE_WAIT_EVENT;
  const char e = (state == EVENT_OCCURRING) ? event : -event;
  stream_ << op << e;
  stream_.flush ();
}
```

#### 6.2.5.27 void iRobot::Create::updateSensors ()

Read the stream to update sensors values. This handles both streamed data and specific queries through querylist or sensors commands.

```
{
  while (!!stream_.good () && !queriedSensors_.empty ())
    {
      if (!!stream_.good () && safePeek (stream_) == STREAM_HEADER)
        readStream ();
      if (!readSensorPacket ())
        if (stream_.rdbuf ()->in_avail () > 0)
          stream_.ignore ();
        else
          break;
    }

  while (!!stream_.good () && stream_.rdbuf ()->in_avail () > 0)
    {
      if (safePeek (stream_) == STREAM_HEADER)
        readStream ();
```

```
      else
        stream_.ignore ();
    }
}
```

#### 6.2.5.28 void iRobot::Create::readStream ()

Read streamed sensors and update sensors values.

```
{
  if (!stream_.good ())
    return;
  if (safePeek (stream_) != STREAM_HEADER)
    return;

  // Ignore header.
  stream_.ignore ();

  std::stringstream ss;
  unsigned char checksum = STREAM_HEADER;
  unsigned char size = 0;

  stream_ >> size;
  checksum += size;
  for (unsigned i = 0; i < size; ++i)
    {
      unsigned char c;
      stream_ >> c;
      ss << c;
      checksum += c;
    }

  unsigned char sentChecksum = 0;
  stream_ >> sentChecksum;

  if (checksum + sentChecksum != 256)
    return;

  for (sensorPackets_t::const_iterator it = streamedSensors_.begin ();
       it != streamedSensors_.end (); ++it)
    {
      unsigned char packet = 0;
      ss >> packet;
      if (packet != *it)
        return;
      if (!readSensorPacket (*it, ss))
        return;
    }
}
```

#### 6.2.5.29 Create::Mode iRobot::Create::mode () throw ()

```
{
  return currentMode_;
}
```

#### 6.2.5.30 bool iRobot::Create::wheeldropCaster ()

#### 6.2.5.31 bool iRobot::Create::wheeldropLeft ()

#### 6.2.5.32 bool iRobot::Create::wheeldropRight ()

#### 6.2.5.33 bool iRobot::Create::bumpLeft ()

#### 6.2.5.34 bool iRobot::Create::bumpRight ()

#### 6.2.5.35 bool iRobot::Create::wall ()

Get wall sensor value.

#### 6.2.5.36 bool iRobot::Create::cliffLeft ()

#### 6.2.5.37 bool iRobot::Create::cliffFrontLeft ()

#### 6.2.5.38 bool iRobot::Create::cliffFrontRight ()

#### 6.2.5.39 bool iRobot::Create::cliffRight ()

#### 6.2.5.40 bool iRobot::Create::deviceDetect ()

#### 6.2.5.41 bool iRobot::Create::digitalInput3 ()

#### 6.2.5.42 bool iRobot::Create::digitalInput2 ()

#### 6.2.5.43 bool iRobot::Create::digitalInput1 ()

#### 6.2.5.44 bool iRobot::Create::digitalInput0 ()

#### 6.2.5.45 short iRobot::Create::analogSignal ()

#### 6.2.5.46 bool iRobot::Create::homeBaseChargerAvailable ()

#### 6.2.5.47 bool iRobot::Create::internalChargerAvailable ()

#### 6.2.5.48 bool iRobot::Create::virtualWall ()

#### 6.2.5.49 bool iRobot::Create::leftWheelOvercurrent ()

#### 6.2.5.50 bool iRobot::Create::rightWheelOvercurrent ()

#### 6.2.5.51 bool iRobot::Create::ld2Overcurrent ()

#### 6.2.5.52 bool iRobot::Create::ld1Overcurrent ()

#### 6.2.5.53 bool iRobot::Create::ld0Overcurrent ()

#### 6.2.5.54 unsigned char iRobot::Create::ir ()

#### 6.2.5.55 bool iRobot::Create::advanceButton ()

#### 6.2.5.56 bool iRobot::Create::playButton ()

#### 6.2.5.57 short iRobot::Create::distance ()

#### 6.2.5.58 short iRobot::Create::angle ()

#### 6.2.5.59 ChargingState iRobot::Create::chargingState ()

#### 6.2.5.60 short iRobot::Create::batteryVoltage ()

#### 6.2.5.61 short iRobot::Create::batteryCurrent ()

#### 6.2.5.62 short iRobot::Create::batteryTemperature ()

#### 6.2.5.63 short iRobot::Create::batteryCharge ()



#### 6.2.5.64 short iRobot::Create::batteryCapacity ()

#### 6.2.5.65 short iRobot::Create::wallSignal ()

#### 6.2.5.66 short iRobot::Create::cliffLeftSignal ()

```
{
  if (!!queriedSensors_.empty ())
    return false;
  if (!!readSensorPacket (queriedSensors_.front (), stream_))
    {
      queriedSensors_.pop ();
      return true;
    }
  return false;
}
```

#### 6.2.5.78 bool iRobot::Create::readSensorPacket (SensorPacket *sensor*, std::istream & *stream*) [protected]

Read sensor packet on a specified stream, update sensor values.

```
{
  if (!stream.good ())
    return false;
  if (safePeek (stream) == STREAM_HEADER)
    return false;

  switch (sensor)
    {
      GROUP_SENSOR(SENSOR_GROUP_0, 7, 26);
      GROUP_SENSOR(SENSOR_GROUP_1, 7, 16);
      GROUP_SENSOR(SENSOR_GROUP_2, 17, 20);
      GROUP_SENSOR(SENSOR_GROUP_3, 21, 26);
      GROUP_SENSOR(SENSOR_GROUP_4, 27, 34);
      GROUP_SENSOR(SENSOR_GROUP_5, 35, 42);
      GROUP_SENSOR(SENSOR_GROUP_6, 7, 42);

    case SENSOR_BUMPS_WHEELS_DROPS:
      {
        std::bitset<8> v = 0;
        unsigned char tmp = 0;
        if (!safeGet (stream, tmp))
          return false;
        v = tmp;
        wheeldropCaster_ = v.test (SENSOR_BIT_WHEELDROP_CASTER);
        wheeldropLeft_ = v.test (SENSOR_BIT_WHEELDROP_LEFT);
        wheeldropRight_ = v.test (SENSOR_BIT_WHEELDROP_RIGHT);
        bumpLeft_ = v.test (SENSOR_BIT_BUMP_LEFT);
        bumpRight_ = v.test (SENSOR_BIT_BUMP_RIGHT);
        break;
      }

      UCHAR_SENSOR (SENSOR_WALL, wall_);
      UCHAR_SENSOR (SENSOR_CLIFF_LEFT, cliffLeft_);
      UCHAR_SENSOR (SENSOR_CLIFF_FRONT_LEFT, cliffFrontLeft_);
      UCHAR_SENSOR (SENSOR_CLIFF_FRONT_RIGHT, cliffFrontRight_);
      UCHAR_SENSOR (SENSOR_CLIFF_RIGHT, cliffRight_);
      UCHAR_SENSOR (SENSOR_VIRTUAL_WALL, virtualWall_);

    case SENSOR_OVERCURRENTS:
      {
        std::bitset<8> v = 0;
        unsigned char tmp = 0;
        if (!safeGet (stream, tmp))
          return false;
        v = tmp;
        leftWheelOvercurrent_ = v.test (SENSOR_BIT_LEFTWHEELOVERCURRENT);
        rightWheelOvercurrent_ = v.test (SENSOR_BIT_RIGHTWHEELOVERCURRENT);
        ld2Overcurrent_ = v.test (SENSOR_BIT_LD2OVERCURRENT);
```

```
        ld1Overcurrent_ = v.test (SENSOR_BIT_LD1OVERCURRENT);
        ld0Overcurrent_ = v.test (SENSOR_BIT_LD0OVERCURRENT);
        break;
      }

      // Unused.
    case 15:
    case 16:
      {
        unsigned char tmp = 0;
        if (!safeGet (stream, tmp))
          return false;
        break;
      }

      UCHAR_SENSOR (SENSOR_IR, ir_);

    case SENSOR_BUTTONS:
      {
        std::bitset<8> v = 0;
        unsigned char tmp = 0;
        if (!safeGet (stream, tmp))
          return false;
        v = tmp;
        advanceButton_ = v.test (SENSOR_BIT_ADVANCEBUTTON);
        playButton_ = v.test (SENSOR_BIT_PLAYBUTTON);
        break;
      }

      INT_SENSOR (SENSOR_DISTANCE, distance_);
      INT_SENSOR (SENSOR_ANGLE, angle_);

      GEN_SENSOR (SENSOR_CHARGING_STATE, chargingState_, ChargingState);

      INT_SENSOR (SENSOR_VOLTAGE, batteryVoltage_);
      INT_SENSOR (SENSOR_CURRENT, batteryCurrent_);
      CHAR_SENSOR (SENSOR_BATTERY_TEMPERATURE, batteryTemperature_);
      INT_SENSOR (SENSOR_BATTERY_CHARGE, batteryCharge_);
      INT_SENSOR (SENSOR_BATTERY_CAPACITY, batteryCapacity_);
      INT_SENSOR (SENSOR_WALL_SIGNAL, wallSignal_);
      INT_SENSOR (SENSOR_CLIFF_LEFT_SIGNAL, cliffLeftSignal_);
      INT_SENSOR (SENSOR_CLIFF_FRONT_LEFT_SIGNAL, cliffFrontLeftSignal_);
      INT_SENSOR (SENSOR_CLIFF_FRONT_RIGHT_SIGNAL, cliffFrontRightSignal_);
      INT_SENSOR (SENSOR_CLIFF_RIGHT_SIGNAL, cliffRightSignal_);

    case SENSOR_CARGO_BAY_DIGITAL_INPUT:
      {
        std::bitset<8> v = 0;
        unsigned char tmp = 0;
        if (!safeGet (stream, tmp))
          return false;
        v = tmp;
        deviceDetect_ = v.test (SENSOR_BIT_DEVICEDETECT);
        digitalInput3_ = v.test (SENSOR_BIT_DIGITALINPUT3);
        digitalInput2_ = v.test (SENSOR_BIT_DIGITALINPUT2);
        digitalInput1_ = v.test (SENSOR_BIT_DIGITALINPUT1);
        digitalInput0_ = v.test (SENSOR_BIT_DIGITALINPUT0);
        break;
      }

      INT_SENSOR(SENSOR_CARGO_BAY_ANALOG_SIGNAL, analogSignal_);

    case SENSOR_CHARGING_SOURCES_AVAILABLE:
      {
        std::bitset<8> v = 0;
        unsigned char tmp = 0;
        if (!safeGet (stream, tmp))
```

```
     return false;
    v = tmp;
    homeBaseChargerAvailable_ =
     v.test (SENSOR_BIT_HOMEBASECHARGERAVAILABLE);
    internalChargerAvailable_ =
     v.test (SENSOR_BIT_INTERNALCHARGERAVAILABLE);
    break;
   }

   //FIXME: trigger a warning if not corresponding to current value?
   GEN_SENSOR (SENSOR_OI_MODE, currentMode_, Mode);

   UCHAR_SENSOR (SENSOR_SONG_NUMBER, songNumber_);
   UCHAR_SENSOR (SENSOR_SONG_PLAYING, songPlaying_);
   UCHAR_SENSOR (SENSOR_NUMBER_STREAM_PACKETS, streamPackets_);

   INT_SENSOR (SENSOR_REQUESTED_VELOCITY, requestedVelocity_);
   INT_SENSOR (SENSOR_REQUESTED_RADIUS, requestedRadius_);
   INT_SENSOR (SENSOR_REQUESTED_LEFT_VELOCITY, requestedLeftVelocity_);
   INT_SENSOR (SENSOR_REQUESTED_RIGHT_VELOCITY, requestedRightVelocity_);

  default:
   break;
  }
 return true;
}
```

#### 6.2.5.79 void iRobot::Create::sendBaudCommand (Baud *speed*) throw (CommandNotAvailable, InvalidArgument)

Change the communication speed. After changing the communication speed, it is required to wait at least 100ms.

**Parameters:**

*speed* New communication speed.

#### 6.2.5.80 void iRobot::Create::sendStartCommand ()

Switch to passive mode. Automatically done when the class is instanciated.

#### 6.2.5.81 void iRobot::Create::sendSafeCommand () throw (CommandNotAvailable)

Switch to safe mode.

#### 6.2.5.82 void iRobot::Create::sendFullCommand () throw (CommandNotAvailable)

Switch to full mode.

#### 6.2.5.83 void iRobot::Create::sendDemoCommand (Demo *demo*) throw (CommandNotAvailable, InvalidArgument)

Run a built-in demo.

**Parameters:**

***demo*** Demo id.

#### 6.2.5.84 void iRobot::Create::sendDriveCommand (short *v*, short *r*) throw (CommandNotAvailable, InvalidArgument)

Drive the robot.

**Parameters:**

***v*** Robot's velocity.

***r*** Robot's turning radius.

**See also:**

**sendDriveDirectCommand** (p. 90)

#### 6.2.5.85 void iRobot::Create::sendDriveCommand (short *v*, DriveCommand *dc*) throw (CommandNotAvailable, InvalidArgument)

Drive the robot using a special drigin mode.

**Parameters:**

***v*** Robot's velocity.

***dc*** Special driving mode.

#### 6.2.5.86 void iRobot::Create::sendDriveDirectCommand (short *vr*, short *vl*) throw (CommandNotAvailable, InvalidArgument)

Drive the two wheels separately.

**Parameters:**

***vr*** Right wheel's velocity.

***vl*** Left wheel's velocity.

#### 6.2.5.87 void iRobot::Create::sendLedCommand (Led *l*, unsigned char *c*, unsigned char *i*) throw (CommandNotAvailable, InvalidArgument)

Change led status.

**Parameters:**

***l*** Switch on/off advance or play leds.

***c*** Power led color.

***i*** Power led light intensity.

#### 6.2.5.88 void iRobot::Create::sendDigitalOutputsCommand (bool *d1*, bool *d2*, bool *d3*) throw (CommandNotAvailable)

Change the digital output values.

**Parameters:**

***d1*** Switch on/off first digital output.

***d2*** Switch on/off second digital output.

***d3*** Switch on/off third digital output.

#### 6.2.5.89 void iRobot::Create::sendPwmLowSideDriversCommand (unsigned char *lsd1*, unsigned char *lsd2*, unsigned char *lsd3*) throw (CommandNotAvailable, InvalidArgument)

Drive low side drivers with variable power.

**Parameters:**

***lsd1*** Low side driver 1 power.

***lsd2*** Low side driver 2 power.

***lsd3*** Low side driver 3 power.

#### 6.2.5.90 void iRobot::Create::sendLowSideDriversCommand (bool *lsd1*, bool *lsd2*, bool *lsd3*) throw (CommandNotAvailable)

Switch on/off low side drivers.

**Parameters:**

***lsd1*** Switch on/off low side driver 1.

***lsd2*** Switch on/off low side driver 2.

***lsd3*** Switch on/off low side driver 3.

**See also:**

**sendPwmLowSideDriversCommand** (p. 91)

#### 6.2.5.91 void iRobot::Create::sendIrCommand (unsigned char *v*) throw (CommandNotAvailable)

Send an IR command on low side driver 1. The data will be encoded using the iRobot's **Create** (p. 29) receiver format.

**Parameters:**

***v*** Sent value.

#### 6.2.5.92 void iRobot::Create::sendSongCommand (unsigned char *sid*, const song_t & *song*) throw (CommandNotAvailable, InvalidArgument)

Define a song.

**Parameters:**

***sid*** Song id.

***song*** Song to send.

#### 6.2.5.93 void iRobot::Create::sendPlaySongCommand (unsigned char *sid*) throw (CommandNotAvailable, InvalidArgument)

Play a song.

**Parameters:**

***sid*** Sond id to play.

#### 6.2.5.94 void iRobot::Create::sendSensorsCommand (SensorPacket *sp*) throw (CommandNotAvailable, InvalidArgument)

Request the robot to send a particular sensor packet.

**Parameters:**

***sp*** Sensor packet to send.

#### 6.2.5.95 void iRobot::Create::sendQueryListCommand (const sensorPackets_t & *lsp*) throw (CommandNotAvailable, InvalidArgument)

Request the robot to send a list of sensor packets.

**Parameters:**

***lsp*** List of sensor packets to send

#### 6.2.5.96 void iRobot::Create::sendStreamCommand (const sensorPackets_t & *lsp*) throw (CommandNotAvailable, InvalidArgument)

Request the robot to start streaming some sensor packets. After the streaming is started, the requested values will be sent every 15ms.

**Parameters:**

***lsp*** List of sensor packets to stream.

#### 6.2.5.97 void iRobot::Create::sendPauseStreamCommand (StreamState *st*) throw (CommandNotAvailable, InvalidArgument)

Pause or resume sensor streaming.

**Parameters:**

***st*** Stream state to set.

#### 6.2.5.98 void iRobot::Create::sendScriptCommand (const opcodes_t & *script*) throw (CommandNotAvailable, InvalidArgument)

Send a script.

**Parameters:**

***script*** Script to send.

#### 6.2.5.99 void iRobot::Create::sendPlayScriptCommand () throw (CommandNotAvailable)

Play the current stored script.

#### 6.2.5.100 void iRobot::Create::sendShowScriptCommand () throw (CommandNotAvailable)

Show the stored script.

#### 6.2.5.101 void iRobot::Create::sendWaitTimeCommand (unsigned char *t*) throw (CommandNotAvailable)

Make the robot wait for a specific amount of time.

**Parameters:**

***t*** Time to wait in tenth of a second (resolution of 15 ms).

#### 6.2.5.102 void iRobot::Create::sendWaitDistanceCommand (short *d*) throw (CommandNotAvailable)

Make the robot wait until it travelled a certain distance.

**Parameters:**

***d*** Distance to travel

#### 6.2.5.103 void iRobot::Create::sendWaitAngleCommand (short *a*) throw (CommandNotAvailable)

Make the robot wait until it has rotated enough.

**Parameters:**

***a*** Angle (in degrees)

#### 6.2.5.104 void iRobot::Create::sendWaitEventCommand (Event *e*, EventState *es* = `EVENT_OCCURRING`) throw (CommandNotAvailable, InvalidArgument)

Make the robot wait for a specific event.

**Parameters:**

***e*** Wait for this event.

***es*** Wait for this event state.

#### 6.2.5.105 void iRobot::Create::updateSensors ()

Read the stream to update sensors values. This handles both streamed data and specific queries through querylist or sensors commands.

#### 6.2.5.106 void iRobot::Create::readStream ()

Read streamed sensors and update sensors values.

#### 6.2.5.107 Mode iRobot::Create::mode () throw ()

#### 6.2.5.108 bool iRobot::Create::wheeldropCaster ()

#### 6.2.5.109 bool iRobot::Create::wheeldropLeft ()

#### 6.2.5.110 bool iRobot::Create::wheeldropRight ()

#### 6.2.5.111 bool iRobot::Create::bumpLeft ()

#### 6.2.5.112 bool iRobot::Create::bumpRight ()

#### 6.2.5.113 bool iRobot::Create::wall ()

Get wall sensor value.

#### 6.2.5.114 bool iRobot::Create::cliffLeft ()

#### 6.2.5.115 bool iRobot::Create::cliffFrontLeft ()

#### 6.2.5.116 bool iRobot::Create::cliffFrontRight ()

#### 6.2.5.117 bool iRobot::Create::cliffRight ()

#### 6.2.5.118 bool iRobot::Create::deviceDetect ()

#### 6.2.5.119 bool iRobot::Create::digitalInput3 ()

#### 6.2.5.120 bool iRobot::Create::digitalInput2 ()

#### 6.2.5.121 bool iRobot::Create::digitalInput1 ()

#### 6.2.5.122 bool iRobot::Create::digitalInput0 ()

#### 6.2.5.123 short iRobot::Create::analogSignal ()

#### 6.2.5.124 bool iRobot::Create::homeBaseChargerAvailable ()

#### 6.2.5.125 bool iRobot::Create::internalChargerAvailable ()

#### 6.2.5.126 bool iRobot::Create::virtualWall ()

#### 6.2.5.127 bool iRobot::Create::leftWheelOvercurrent ()

#### 6.2.5.128 bool iRobot::Create::rightWheelOvercurrent ()

#### 6.2.5.129 bool iRobot::Create::ld2Overcurrent ()

#### 6.2.5.130 bool iRobot::Create::ld1Overcurrent ()

#### 6.2.5.131 bool iRobot::Create::ld0Overcurrent ()

#### 6.2.5.132 unsigned char iRobot::Create::ir ()

#### 6.2.5.133 bool iRobot::Create::advanceButton ()

#### 6.2.5.134 bool iRobot::Create::playButton ()

#### 6.2.5.135 short iRobot::Create::distance ()

#### 6.2.5.136 short iRobot::Create::angle ()

#### 6.2.5.137 ChargingState iRobot::Create::chargingState ()

#### 6.2.5.138 short iRobot::Create::batteryVoltage ()

#### 6.2.5.139 short iRobot::Create::batteryCurrent ()

#### 6.2.5.140 short iRobot::Create::batteryTemperature ()

#### 6.2.5.141 short iRobot::Create::batteryCharge ()



#### 6.2.5.142 short iRobot::Create::batteryCapacity ()

#### 6.2.5.143 short iRobot::Create::wallSignal ()

#### 6.2.5.144 short iRobot::Create::cliffLeftSignal ()

#### 6.2.5.156 bool iRobot::Create::readSensorPacket (SensorPacket, std::istream &) [protected]

Read sensor packet on a specified stream, update sensor values.

### 6.2.6 Member Data Documentation

#### 6.2.6.1 static const unsigned char iRobot::Create::LED_COLOR_GREEN = 0 [static]

Define the green color for power led.

#### 6.2.6.2 static const unsigned char iRobot::Create::LED_COLOR_RED = 255 [static]

Define the red color for power led.

#### 6.2.6.3 static const unsigned char iRobot::Create::LED_INTENSITY_OFF = 0 [static]

Define minimum intensity for power led (off).

#### 6.2.6.4 static const unsigned char iRobot::Create::LED_INTENSITY_FULL = 255 [static]

Define full intensity for power led.

#### 6.2.6.5 static const unsigned char iRobot::Create::LOW_SIDE_VELOCITY_MIN = 0 [static]

Define minimum velocity for low side drivers.

#### 6.2.6.6 static const unsigned char iRobot::Create::LOW_SIDE_VELOCITY_-MAX = 128 [static]

Define maximum velocity for low side drivers.

#### 6.2.6.7 static const int iRobot::Create::VELOCITY_MIN = -500 [static]

Define minimum velocity for robot wheels motors.

#### 6.2.6.8 static const int iRobot::Create::VELOCITY_MAX = 500 [static]

Define maximum velocity for robot wheels motors.

#### 6.2.6.9 static const int iRobot::Create::RADIUS_MIN = -2000 [static]

Define minimum radius turn of the robot.

#### 6.2.6.10 static const int iRobot::Create::RADIUS_MAX = 2000 [static]

Define maximum radius turn of the robot.

#### 6.2.6.11 static const unsigned char iRobot::Create::SONG_MIN = 0 [static]

Define minimum song id.

#### 6.2.6.12 static const unsigned char iRobot::Create::SONG_MAX = 15 [static]

Define maximum song id.

#### 6.2.6.13 static const unsigned char iRobot::Create::SONG_MAX_SIZE = 16 [static]

Define song maximum size.

#### 6.2.6.14 static const unsigned char iRobot::Create::NO_NOTE = 30 [static]

Define the value for a rest node (no sound).

#### 6.2.6.15 static const unsigned char iRobot::Create::NOTE_MIN = 31 [static]

Define the minimum note (G).

#### 6.2.6.16 static const unsigned char iRobot::Create::NOTE_MAX = 127 [static]

Define the maximum note (G).

#### 6.2.6.17 static const unsigned char iRobot::Create::SCRIPT_MAX_SIZE = 100 [static]

Define maximum script size.

#### 6.2.6.18 static const unsigned char iRobot::Create::STREAM_HEADER = 19 [static]

Stream header “magic value”.

#### 6.2.6.19 static const unsigned char iRobot::Create::SENSOR_BIT_-WHEELDROP_CASTER = 4 [static]

Bit used to retrieve the wheeldrop caster status.

#### 6.2.6.20 static const unsigned char iRobot::Create::SENSOR_BIT_-WHEELDROP_LEFT = 3 [static]

Bit used to retrieve the wheeldrop caster status.

#### 6.2.6.21 static const unsigned char iRobot::Create::SENSOR_BIT_-WHEELDROP_RIGHT = 2 [static]

Bit used to retrieve the wheeldrop caster status.

#### 6.2.6.22 static const unsigned char iRobot::Create::SENSOR_BIT_BUMP_LEFT = 1 [static]

Bit used to retrieve the bump left status.

#### 6.2.6.23 static const unsigned char iRobot::Create::SENSOR_BIT_BUMP_-RIGHT = 0 [static]

Bit used to retrieve the bump right status.

#### 6.2.6.24 static const unsigned char iRobot::Create::SENSOR_BIT_-LEFTWHEELOVERCURRENT = 4 [static]

Bit used to retrieve the left wheel overcurrent status.

#### 6.2.6.25 static const unsigned char iRobot::Create::SENSOR_BIT_-RIGHTWHEELOVERCURRENT = 3 [static]

Bit used to retrieve the right wheel overcurrent status.

#### 6.2.6.26 static const unsigned char iRobot::Create::SENSOR_BIT_-LD2OVERCURRENT = 2 [static]

Bit used to retrieve the LD2 overcurrent status.

#### 6.2.6.27 static const unsigned char iRobot::Create::SENSOR_BIT_-LD1OVERCURRENT = 1 [static]

Bit used to retrieve the LD1 overcurrent status.

#### 6.2.6.28 static const unsigned char iRobot::Create::SENSOR_BIT_-LD0OVERCURRENT = 0 [static]

Bit used to retrieve the LD0 overcurrent status.

#### 6.2.6.29 static const unsigned char iRobot::Create::SENSOR_BIT_-ADVANCEBUTTON = 2 [static]

Bit used to retrieve the advance button status.

#### 6.2.6.30 static const unsigned char iRobot::Create::SENSOR_BIT_-PLAYBUTTON = 0 [static]

Bit used to retrieve the play button status.

#### 6.2.6.31 static const unsigned char iRobot::Create::SENSOR_BIT_-DEVICEDETECT = 4 [static]

Bit used to retrieve the device detect pin status.

#### 6.2.6.32 static const unsigned char iRobot::Create::SENSOR_BIT_-DIGITALINPUT3 = 3 [static]

Bit used to retrieve the digital input 3 pin status.

#### 6.2.6.33 static const unsigned char iRobot::Create::SENSOR_BIT_-DIGITALINPUT2 = 2 [static]

Bit used to retrieve the digital input 2 pin status.

#### 6.2.6.34 static const unsigned char iRobot::Create::SENSOR_BIT_-DIGITALINPUT1 = 1 [static]

Bit used to retrieve the digital input 1 pin status.

#### 6.2.6.35 static const unsigned char iRobot::Create::SENSOR_BIT_-DIGITALINPUT0 = 0 [static]

Bit used to retrieve the digital input 0 pin status.

#### 6.2.6.36 static const unsigned char iRobot::Create::SENSOR_BIT_-HOMEBASECHARGERAVAILABLE = 1 [static]

Bit used to retrieve whether the home base charger is available or not.

#### 6.2.6.37 static const unsigned char iRobot::Create::SENSOR_BIT_-INTERNALCHARGERAVAILABLE = 0 [static]

Bit used to retrieve whether the internal charger is available or not.

The documentation for this class was generated from the following files:

- /home/thomas/projects/usc/libirobot-create/_build/libirobot-create-0.1/src/**irobot-create.hh**
- /home/thomas/projects/usc/libirobot-create/src/**irobot-create.hh**
- /home/thomas/projects/usc/libirobot-create/_build/libirobot-create-0.1/src/**irobot-create.cc**
- /home/thomas/projects/usc/libirobot-create/src/**irobot-create.cc**

# Chapter 7

# iRobot Create library File Documentation

## 7.1 /home/thomas/projects/usc/libirobot-create/_-build/libirobot-create-0.1/src/doc.hh File Reference

### Namespaces

- namespace **iRobot**

## 7.2 /home/thomas/projects/usc/libirobot-create/src/doc.hh File Reference

### Namespaces

- namespace **iRobot**

## 7.3 /home/thomas/projects/usc/libirobot-create/_build/libirobot-create-0.1/src/irobot-create-ctl.cc File Reference

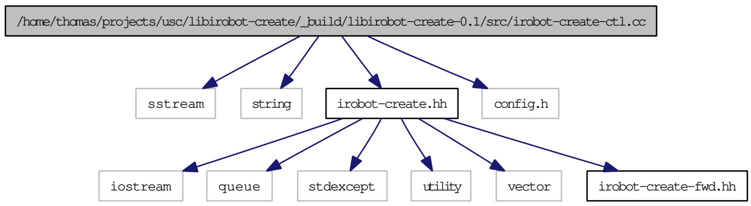
`#include <sstream>`

`#include <string>`

`#include <irobot-create.hh>`

`#include "config.h"`

Include dependency graph for irobot-create-ctl.cc:




### Classes

- struct **Application**

### Defines

- #define **APPLY_ON_OPCODES**(Macro)
- #define **CASTARG_ENUM**(ENUM_TYPE)
- #define **SEND_CMD**(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE)
- #define **STR_TO_OPCODE**(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE)
- #define **PRINT_ACTIONS**(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE) std::cout << "\t" << OPCODE_STRING << std::endl
- #define **SENSOR_CMD**(CALL)

### Functions

- template<typename T>
  T **castArg** (const char ∗)
- **CASTARG_ENUM** (bool)
- **CASTARG_ENUM** (int)
- **CASTARG_ENUM** (unsigned char)
- int **main** (int argc, char ∗∗argv)

## 7.3.1 Define Documentation

#### 7.3.1.1 #define APPLY_ON_OPCODES(Macro)

#### 7.3.1.2 #define CASTARG_ENUM(ENUM_TYPE)

**Value:**

```
template <>                                   \
  ENUM_TYPE castArg (const char* arg)           \
  {                                             \
    std::istringstream i (arg);                 \
    int res = 0;                                \
    if (!(i >> res))                            \
      res = 0;                                  \
    return static_cast<ENUM_TYPE> (res);        \
  }
```

#### 7.3.1.3 #define PRINT_ACTIONS(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE) std::cout << "\t" << OPCODE_STRING << std::endl

#### 7.3.1.4 #define SEND_CMD(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE)

**Value:**

```
case OPCODE:                                                    \
  if (argc < (NARGMIN) + 2)                                       \
    usage ();                                                     \
  if (argc > (NARGMAX) + 2)                                       \
    usage ();                                                     \
  CODE;                                                           \
  break
```

#### 7.3.1.5 #define SENSOR_CMD(CALL)

**Value:**

```
Create::sensorPackets_t sensors;                                       \
  for (int i = 2; i < argc; ++i)                                         \
    sensors.push_back (castArg<Create::SensorPacket> (argv[i]));         \
  CALL
```

#### 7.3.1.6 #define STR_TO_OPCODE(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE)

**Value:**

```
if (m == OPCODE_STRING)                                                \
    return static_cast<Create::Opcode> (OPCODE)
```

## 7.3.2 Function Documentation

#### 7.3.2.1 template<typename T> T castArg (const char *) [inline]

```
{
  std::cerr << "Invalid conversion." << std::endl;
  exit (-1);
}
```

#### 7.3.2.2 CASTARG_ENUM (unsigned *char*)

#### 7.3.2.3 CASTARG_ENUM (int)

#### 7.3.2.4 CASTARG_ENUM (bool)

#### 7.3.2.5 int main (int *argc*, char ** *argv*)

```
{
  std::stringstream stream;
  Application app (argc, argv, stream);
  std::cout << stream.str ();
}
```

## 7.4 /home/thomas/projects/usc/libirobot-create/src/irobot-create-ctl.cc File Reference

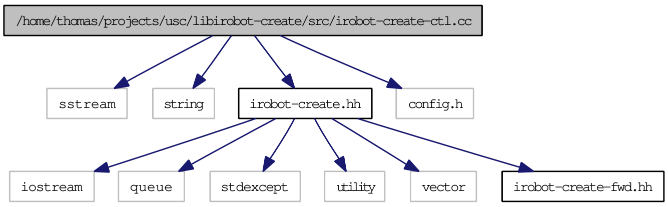
`#include <sstream>`

`#include <string>`

`#include <irobot-create.hh>`

`#include "config.h"`

Include dependency graph for irobot-create-ctl.cc:




### Classes

- struct **Application**

### Defines

- #define **APPLY_ON_OPCODES**(Macro)
- #define **CASTARG_ENUM**(ENUM_TYPE)
- #define **SEND_CMD**(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE)
- #define **STR_TO_OPCODE**(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE)
- #define **PRINT_ACTIONS**(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE) std::cout << "\t" << OPCODE_STRING << std::endl
- #define **SENSOR_CMD**(CALL)

### Functions

- template<typename T>
  T **castArg** (const char ∗)
- **CASTARG_ENUM** (bool)
- **CASTARG_ENUM** (int)
- **CASTARG_ENUM** (unsigned char)
- int **main** (int argc, char ∗∗argv)

### 7.4.1 Define Documentation

#### 7.4.1.1 #define APPLY_ON_OPCODES(Macro)

#### 7.4.1.2 #define CASTARG_ENUM(ENUM_TYPE)

**Value:**

```
template <>                                  \
  ENUM_TYPE castArg (const char* arg)          \
  {                                            \
    std::istringstream i (arg);                \
    int res = 0;                               \
    if (!(i >> res))                           \
      res = 0;                                 \
    return static_cast<ENUM_TYPE> (res);       \
  }
```

#### 7.4.1.3 #define PRINT_ACTIONS(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE) std::cout << "\t" << OPCODE_STRING << std::endl

#### 7.4.1.4 #define SEND_CMD(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE)

**Value:**

```
case OPCODE:                                                    \
  if (argc < (NARGMIN) + 2)                                       \
    usage ();                                                     \
  if (argc > (NARGMAX) + 2)                                       \
    usage ();                                                     \
  CODE;                                                           \
  break
```

#### 7.4.1.5 #define SENSOR_CMD(CALL)

**Value:**

```
Create::sensorPackets_t sensors;                                     \
  for (int i = 2; i < argc; ++i)                                       \
    sensors.push_back (castArg<Create::SensorPacket> (argv[i]));       \
  CALL
```

#### 7.4.1.6 #define STR_TO_OPCODE(OPCODE, OPCODE_STRING, NARGMIN, NARGMAX, CODE)

**Value:**

```
if (m == OPCODE_STRING)                                              \
    return static_cast<Create::Opcode> (OPCODE)
```

### 7.4.2 Function Documentation

#### 7.4.2.1 template<typename T> T castArg (const char *) [inline]

#### 7.4.2.2 CASTARG_ENUM (unsigned *char*)

#### 7.4.2.3 CASTARG_ENUM (int)

#### 7.4.2.4 CASTARG_ENUM (bool)

#### 7.4.2.5 int main (int *argc*, char ** *argv*)

```
{
  std::stringstream stream;
  Application app (argc, argv, stream);
  std::cout << stream.str ();
}
```

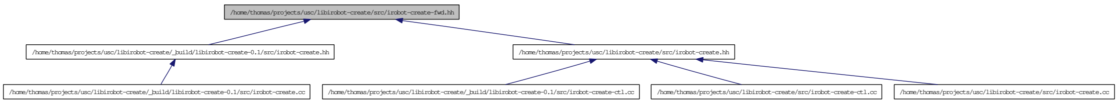
## 7.5 /home/thomas/projects/usc/libirobot-create/_build/libirobot-create-0.1/src/irobot-create-fwd.hh File Reference

### Namespaces

- namespace **iRobot**
- namespace **LibSerial**

## 7.6 /home/thomas/projects/usc/libirobot-create/src/irobot-create-fwd.hh File Reference

This graph shows which files directly or indirectly include this file:




### Namespaces

- namespace **iRobot**
- namespace **LibSerial**

## 7.7 /home/thomas/projects/usc/libirobot-create/_build/libirobot-create-0.1/src/irobot-create.cc File Reference

`#include "config.h"`

`#include <bitset>`

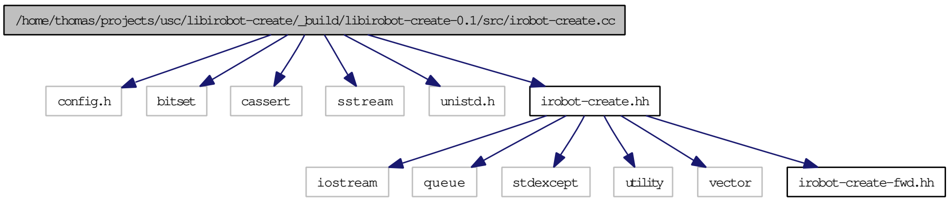
`#include <cassert>`

`#include <sstream>`

`#include <unistd.h>`

`#include "irobot-create.hh"`

Include dependency graph for irobot-create.cc:




### Namespaces

- namespace **iRobot**

### Defines

- #define **INITIALIZE_SENSORS**()
- #define **MAKE_SENSOR_CMD**(OP, CMD)

    *Internal macro.*

- #define **GEN_SENSOR**(PACKET, SENSOR, TYPE)
- #define **CHAR_SENSOR**(PACKET, SENSOR) GEN_SENSOR (PACKET, SENSOR, char)
- #define **UCHAR_SENSOR**(PACKET, SENSOR) GEN_SENSOR (PACKET, SENSOR, unsigned char)
- #define **INT_SENSOR**(PACKET, SENSOR)
- #define **GROUP_SENSOR**(PACKET, MIN, MAX)
- #define **MK_SENSOR_GETTER**(TYPE, NAME)

### 7.7.1 Define Documentation

#### 7.7.1.1 #define CHAR_SENSOR(PACKET, SENSOR) GEN_SENSOR (PACKET, SENSOR, char)

#### 7.7.1.2 #define GEN_SENSOR(PACKET, SENSOR, TYPE)

**Value:**

```
case PACKET:                                   \
  {                                            \
    unsigned char v = 0;                       \
    if (!safeGet (stream, v))                  \
      return false;                            \
    SENSOR = static_cast<TYPE> (v);            \
  }                                            \
  break
```

#### 7.7.1.3 #define GROUP_SENSOR(PACKET, MIN, MAX)

**Value:**

```
{                                                              \
  case PACKET:                                                 \
    bool res = true;                                           \
    for (SensorPacket sensor = static_cast<SensorPacket> (MIN); \
         sensor < static_cast<SensorPacket> (MAX);             \
         sensor = static_cast<SensorPacket> (sensor + 1))      \
      res = res && readSensorPacket (sensor, stream);          \
    if (!res)                                                  \
      return false;                                            \
  }                                                            \
    break
```

#### 7.7.1.4 #define INITIALIZE_SENSORS()

#### 7.7.1.5 #define INT_SENSOR(PACKET, SENSOR)

**Value:**

```
case PACKET:                                   \
  {                                            \
    SENSOR = receiveInt16 (stream);            \
  }                                            \
  break
```

#### 7.7.1.6 #define MAKE_SENSOR_CMD(OP, CMD)

**Value:**

```
if (currentMode_ < IROBOT_CREATE_PASSIVE)                      \
    throw CommandNotAvailable ();                              \
  if (packets.size () > 255)                                   \
    throw InvalidArgument ();                                  \
                                                               \
  std::stringstream ss;                                        \
  for (sensorPackets_t::const_iterator it = packets.begin ();  \
       it != packets.end (); ++it)                             \
    {                                                          \
      if (*it < SENSOR_GROUP_0                                 \
          || *it > SENSOR_REQUESTED_LEFT_VELOCITY)             \
        throw InvalidArgument ();                              \
      const unsigned char p = *it;                             \
      ss << p;                                                 \
      CMD;                                                     \
    }                                                          \
```

```
                                                                    \
  const unsigned char op  = (OP);                                   \
  const unsigned char size = packets.size ();                       \
  stream_ << op << size << ss.str ();                               \
  stream_.flush ()
```

Internal macro.

#### 7.7.1.7 #define MK_SENSOR_GETTER(TYPE, NAME)

**Value:**

```
TYPE                                         \
  Create::NAME ()                              \
  {                                            \
    updateSensors ();                          \
    return NAME##_;                            \
  }
```

#### 7.7.1.8 #define UCHAR_SENSOR(PACKET, SENSOR) GEN_SENSOR (PACKET, SENSOR, unsigned char)

## 7.8 /home/thomas/projects/usc/libirobot-create/src/irobot-create.cc File Reference

`#include "config.h"`

`#include <bitset>`

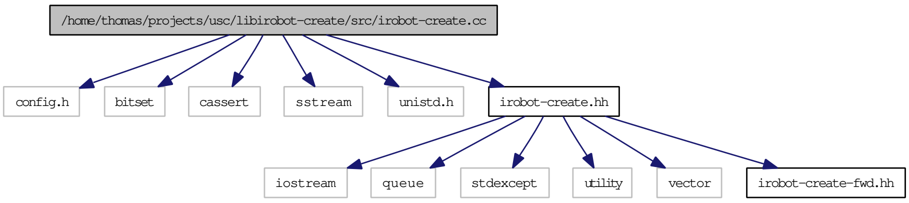
`#include <cassert>`

`#include <sstream>`

`#include <unistd.h>`

`#include "irobot-create.hh"`

Include dependency graph for irobot-create.cc:




### Namespaces

- namespace **iRobot**

### Defines

- #define **INITIALIZE_SENSORS**()
- #define **MAKE_SENSOR_CMD**(OP, CMD)

  *Internal macro.*

- #define **GEN_SENSOR**(PACKET, SENSOR, TYPE)
- #define **CHAR_SENSOR**(PACKET, SENSOR) GEN_SENSOR (PACKET, SENSOR, char)
- #define **UCHAR_SENSOR**(PACKET, SENSOR) GEN_SENSOR (PACKET, SENSOR, unsigned char)
- #define **INT_SENSOR**(PACKET, SENSOR)
- #define **GROUP_SENSOR**(PACKET, MIN, MAX)
- #define **MK_SENSOR_GETTER**(TYPE, NAME)

### 7.8.1 Define Documentation

#### 7.8.1.1 #define CHAR_SENSOR(PACKET, SENSOR) GEN_SENSOR (PACKET, SENSOR, char)

#### 7.8.1.2 #define GEN_SENSOR(PACKET, SENSOR, TYPE)

**Value:**

```
case PACKET:                                  \
  {                                           \
    unsigned char v = 0;                      \
    if (!safeGet (stream, v))                 \
      return false;                           \
    SENSOR = static_cast<TYPE> (v);           \
  }                                           \
  break
```

#### 7.8.1.3 #define GROUP_SENSOR(PACKET, MIN, MAX)

**Value:**

```
{                                                                \
  case PACKET:                                                   \
    bool res = true;                                             \
    for (SensorPacket sensor = static_cast<SensorPacket> (MIN); \
         sensor < static_cast<SensorPacket> (MAX);               \
         sensor = static_cast<SensorPacket> (sensor + 1))        \
      res = res && readSensorPacket (sensor, stream);            \
    if (!res)                                                    \
      return false;                                              \
  }                                                              \
    break
```

#### 7.8.1.4 #define INITIALIZE_SENSORS()

#### 7.8.1.5 #define INT_SENSOR(PACKET, SENSOR)

**Value:**

```
case PACKET:                                  \
  {                                           \
    SENSOR = receiveInt16 (stream);           \
  }                                           \
  break
```

#### 7.8.1.6 #define MAKE_SENSOR_CMD(OP, CMD)

**Value:**

```
if (currentMode_ < IROBOT_CREATE_PASSIVE)                        \
    throw CommandNotAvailable ();                                \
  if (packets.size () > 255)                                     \
    throw InvalidArgument ();                                    \
                                                                 \
  std::stringstream ss;                                          \
  for (sensorPackets_t::const_iterator it = packets.begin ();    \
       it != packets.end (); ++it)                               \
    {                                                            \
      if (*it < SENSOR_GROUP_0                                   \
          || *it > SENSOR_REQUESTED_LEFT_VELOCITY)               \
        throw InvalidArgument ();                                \
      const unsigned char p = *it;                               \
      ss << p;                                                   \
      CMD;                                                       \
    }                                                            \
```

```
                                                                        \
  const unsigned char op  = (OP);                                       \
  const unsigned char size = packets.size ();                           \
  stream_ << op << size << ss.str ();                                   \
  stream_.flush ()
```

Internal macro.

#### 7.8.1.7 #define MK_SENSOR_GETTER(TYPE, NAME)

**Value:**

```
TYPE                                         \
  Create::NAME ()                              \
  {                                            \
    updateSensors ();                          \
    return NAME##_;                            \
  }
```

#### 7.8.1.8 #define UCHAR_SENSOR(PACKET, SENSOR) GEN_SENSOR (PACKET, SENSOR, unsigned char)

## 7.9 /home/thomas/projects/usc/libirobot-create/_build/libirobot-create-0.1/src/irobot-create.hh File Reference

#include <iostream>

#include <queue>

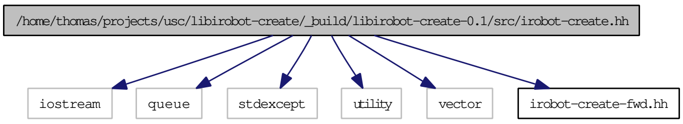
#include <stdexcept>

#include <utility>

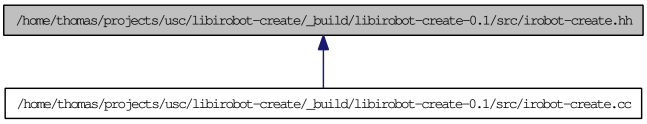
#include <vector>

#include <irobot-create-fwd.hh>

Include dependency graph for irobot-create.hh:




This graph shows which files directly or indirectly include this file:




### Namespaces

- namespace **iRobot**

### Classes

- class **iRobot::Create**

    *Robot communication class, wraps Open Interface version 2.*

### Defines

- #define **IROBOT_MAKE_EXCEPTION**(NAME, DESCRIPTION)

    *Create an exception class which inherits from std::runtime_error.*

### Functions

- **iRobot::IROBOT_MAKE_EXCEPTION** (CommandNotAvailable,"This command is not available in this mode.")

*Error thrown when the command is not valid in the current mode.*

- **iRobot::IROBOT_MAKE_EXCEPTION** (InvalidArgument,"This argument is invalid.")

  *Exception thrown when an argument's value is invalid. Typically, it means that an argument is out of bound.*

- **iRobot::IROBOT_MAKE_EXCEPTION** (LibSerialNotAvailable,"Serial port support has not been compiled.")

  *Exception thrown if no serial port support is available but one tries to use the feature anyway.*

### 7.9.1 Define Documentation

#### 7.9.1.1 #define IROBOT_MAKE_EXCEPTION(NAME, DESCRIPTION)

**Value:**

```
struct NAME : public std::runtime_error                          \
  {                                                               \
    explicit NAME ()                                              \
      : std::runtime_error (DESCRIPTION)                          \
    {}                                                            \
    explicit NAME (const std::string& what)                       \
      : std::runtime_error (what)                                 \
    {}                                                            \
  }
```

Create an exception class which inherits from std::runtime_error.

The class inherits from std::runtime_error and is default constructible. The default error message is set by the DESCRIPTION parameter, a specific error message can also be set by passing a string to the constructor.

**Parameters:**

***NAME*** Name of the generated exception class.

***DESCRIPTION*** Default error message.

# 7.10 /home/thomas/projects/usc/libirobot-create/src/irobot-create.hh File Reference

`#include <iostream>`

`#include <queue>`

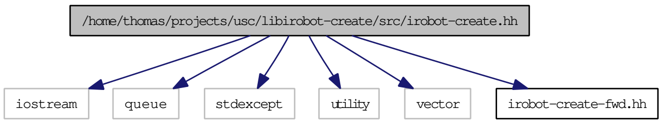
`#include <stdexcept>`

`#include <utility>`

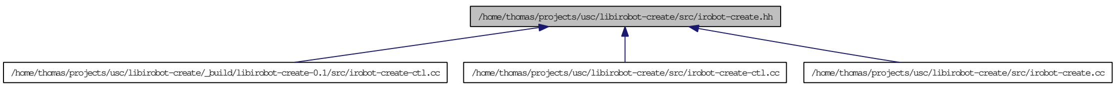
`#include <vector>`

`#include <irobot-create-fwd.hh>`

Include dependency graph for irobot-create.hh:




This graph shows which files directly or indirectly include this file:




## Namespaces

- namespace **iRobot**

## Classes

- class **iRobot::Create**

  *Robot communication class, wraps Open Interface version 2.*

## Defines

- #define **IROBOT_MAKE_EXCEPTION**(NAME, DESCRIPTION)

  *Create an exception class which inherits from std::runtime_error.*

## Functions

- **iRobot::IROBOT_MAKE_EXCEPTION** (CommandNotAvailable,"This command is not available in this mode.")

  *Error thrown when the command is not valid in the current mode.*

- **iRobot::IROBOT_MAKE_EXCEPTION** (InvalidArgument,"This argument is invalid.")

  *Exception thrown when an argument's value is invalid. Typically, it means that an argument is out of bound.*

- **iRobot::IROBOT_MAKE_EXCEPTION** (LibSerialNotAvailable,"Serial port support has not been compiled.")

  *Exception thrown if no serial port support is available but one tries to use the feature anyway.*

### 7.10.1 Define Documentation

#### 7.10.1.1 #define IROBOT_MAKE_EXCEPTION(NAME, DESCRIPTION)

**Value:**

```
struct NAME : public std::runtime_error                          \
  {                                                               \
    explicit NAME ()                                              \
      : std::runtime_error (DESCRIPTION)                          \
    {}                                                            \
    explicit NAME (const std::string& what)                       \
      : std::runtime_error (what)                                 \
    {}                                                            \
  }
```

Create an exception class which inherits from std::runtime_error.

The class inherits from std::runtime_error and is default constructible. The default error message is set by the DESCRIPTION parameter, a specific error message can also be set by passing a string to the constructor.

**Parameters:**

***NAME*** Name of the generated exception class.

***DESCRIPTION*** Default error message.

# Index