# iRobot Create library communication library

Thomas Moulard, the University of
Southern California (USC), and iLab at USC

# Copying this document

# Contents

# Chapter 1

# Introduction

The "iRobot Create communication library" (or `libirobot-create`) provides a complete implementation of the `Open Interface` version 2 [2] in `C++`.

The iRobot Create [3] targets developers, educators and hobbyist. It eases the work by providing a pre-assembled plat-form and a set of accessories. However, no software is provided by iRobot to control the robot remotely.

This library supports:

- Direct connection through a serial cable.
- Bluetooth connection using a Bluetooth Adapter Module (BAM).

It allows complex software running on a remote computer to drive the robot.

The main drawback of the Bluetooth Adapter Module is the time lag introduced by the wireless connection, the bluetooth range can also be a problem. If a cable is used, there is no lag but the robots' movements are limited.

If the robot *has* to be totally autonomous, a good solution may be to setup a small laptop on the robot and use a direct serial connection.

*Important:* iRobot also sell a "command module" [1] which is a 8-bit, 18MHz microcontroller, this library does *not* support it.

The library, the documentation and all associated files are available on the project's homepage: `https://savannah.nongnu.org/projects/libirobot-create/`.

| *Connection type?* | Time lag? | Range? |
|---|---|---|
| *Direct connection* | none | cable length |
| *Bluetooth* | yes | 100 meters (class 1 devices) |

Figure 1.1: Advantages and drawbacks of each conncetion.

# Chapter 2

# Installation

## 2.1 Specific instructions for the Bluetooth Adapter Module (BAM)

Bluetooth has to be working on your computer, it includes kernel support and the installation of the `bluez` package which provides useful command line tools. You may also want to install a graphic user interface such as `kdebluetooth`.

## 2.2 Versions and dependencies

This library has an (optional but *strongly* recommended) dependency on LibSerial[1], please make sure that this library is installed on your system before trying to use this library using serial port communication.

## 2.3 Installation from a package

### 2.3.1 Ebuild (Gentoo)

An ebuild is provided in the download area, however it is not included in Portage which means you can not directly type `emerge libirobot-create`.

You first have to setup a local Portage overlay. The process is explained on Gentoo-wiki.com[2].

When your local overlay is ready, download the ebuild and copy it in your overlay. You can then install the package normally.

[1] More information on LibSerial website `http://libserial.sourceforge.net/`.

[2] A complete tutorial is available here: `http://gentoo-wiki.com/HOWTO_Installing_3rd_Party_Ebuilds`

### 2.3.2 Deb (Debian)

A deb file is provided in the download area, to install it follow the instructions of the Listing 2.1.

```
dpkg -i filename.deb
```

Listing 2.1: Installing a debian package (root privileges required)

### 2.3.3 RPM

RPMs are available for the LibSerial library. A 32 bits [3] version can be found on internet, a 64 bits is available in the download area [4] of the project.

```
rpm -i libserial-0.5.2-FC5.i386.rpm # For 32-bits.
# *OR*
rpm -i libserial-0.5.2-0.x86_64.rpm # For 64-bits.
```

Listing 2.2: Installing LibSerial from an RPM

However, this RPM does not provide the libtool library, so you'll have to execute the following command to indicate that you don't want to check this dependency (libtool libraries are optional so it is not a real problem).

```
echo "libtool(/usr/lib/libserial.la)" >> /etc/rpm/sysinfo
```

Listing 2.3: Remove `libserial.la` dependency

You can then download a rpm for the `libirobot-create` from the project homepage.

```
rpm -i libirobot-create-0.1-1.i386.rpm # For 32-bits.
# *OR*
rpm -i libirobot-create-0.1-1.x86_64.rpm # For 64-bits.
```

Listing 2.4: Remove `libserial.la` dependency

## 2.4 Installation from a tarball

To compile from a tarball and install the library, follow the instructions of Listing 2.7. Invoke `configure` without the `prefix` flag to install the library in `/usr/local`.

[3] `http://rpm.pbone.net/index.php3/stat/4/idpl/2893519/com/libserial-0.5.2-FC5.i386.rpm.html`

[4] `http://download.savannah.gnu.org/releases/libirobot-create/libserial-0.5.2-0.x86_64.rpm`

## 2.5 Installation from the Git repository

If you do not have libserial, install it (from a package or from source)[5].

### 2.5.1 Checking out

```
git clone git://git.savannah.nongnu.org/libirobot-create.git
```

Listing 2.5: Anonymous check-out of the project repository

### 2.5.2 Building the library

To compile, follow the instructions of the Listing 2.6. If you want to install definitively the package on your computer, follow the instructions of Listing 2.7.

```
./bootstrap
mkdir _build
cd _build
../configure
make
# To install *temporarily* (optional):
make install DESTDIR=/my/tmp/install/directory
```

Listing 2.6: Compiling the sources

```
./bootstrap
mkdir _build
cd _build
../configure --prefix=$HOME/.local
mkdir $HOME/.local
make install
# Check if it has been installed successfully:
ls -R $HOME/.local # You should see all the installed files.
```

Listing 2.7: Compiling the sources and definitive installation

*Note:* these instructions are working for any other Autotools package.

You can change the prefix to whatever you want, however it is *usually a bad idea* to install a source package in usual prefixes such as `/usr` or `/usr/local`.

## 2.6 Documentation

The installation process install the Doxygen documentation on your computer. If you are installing from the tarball and you have kept the default installation paths, you can find it here: `/usr/share/doc/libirobot-create`.

[5] More information on LibSerial website `http://libserial.sourceforge.net/`.

For packages, it is usually in `/usr/doc/libirobot-create`.

If you are compiling from source, you can compile the Doxygen documentation yourself using the commands of the Listing 2.8 for html version or of Listing 2.9 to generate a PDF.

```
cd _build
make doc
```

Listing 2.8: Building Doxygen documentation

```
cd _build/doc/latex
make pdf
```

Listing 2.9: Building Doxygen documentation

# Chapter 3

# Tutorial

iRobot 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) 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[1] library (it library has to be installed on your computer).

## 3.1 Serial port communication tutorial

The Listing 3.1 provides the entire code source of the tutorial.

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

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


  if (argc < 2)
```

[1] `http://libserial.sourceforge.net`

```
  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;
    }
}
```

Listing 3.1: Tutorial sources

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.

## 3.2 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.

The Listing 3.2 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 ();
}
```

Listing 3.2: Logging tutorial

# Chapter 4

# Frequently Asked Questions

## 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 support, the constructor will fail and throw a LibSerialNotAvailable exception. It means that when the configure script has not detected the LibSerial library on your computer. Check your config.log, it should contain something like that:

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

Listing 4.1: `config.log` sample

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
```

Listing 4.2: Invoking `configure` when using non-standard paths

*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 support.*

## 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 library (or another 3rd party serial library). Failing

to do so may lead to random errors, wrong values for the sensors, etc.

## 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 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.
```

Listing 4.3: Sensor reading sample

### 4.3.1 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
```

Listing 4.4: Scanning for Bluetooth devices in command line

You should see something that looks like that:

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

Listing 4.5: Bluetooth scan result sample

The *Element Serial* is the iRobot 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'
```

Listing 4.6: Connecting to the robot

You should then see:

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

Listing 4.7: Result of a Bluetooth robot connection

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.

### 4.3.2 How to compile my project with libirobot-create?

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
```

Listing 4.8: Compiling a project using the library

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
```

Listing 4.9: Compiling a project using the library with non-standard paths

The `-Wl,-R/my/library/path` 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.

### 4.3.3 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
```

Listing 4.10: Library detection with Autoconf

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

# Bibliography

[1] iRobot. irobot command module owners manual. http://www.irobot.com/filelibrary/create/Command%20Module%20Manual_v2.pdf.

[2] iRobot. irobot create open interface specification. http://www.irobot.com/filelibrary/create/Create%20Open%20Interface_v2.pdf.

[3] iRobot. irobot create owners guide. http://www.irobot.com/filelibrary/create/Create%20Manual_Final.pdf.

# Listings

# Appendix A

# Remote control example

This example implements a simple remote controller for the robot using the arrows keys of the keyboard.

This example *requires* the library `ncurses`[1].

To compile this example, use the command described in the Listing A.1 where `remote-control.cc` is the file described in the Listing A.2.

```
$ g++ remote-control.cc -lirobot-create -lncurses
```

Listing A.1: Compiling the remote control example

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

#include <ncurses.h>

using namespace iRobot;
using namespace LibSerial;

static int speed = 0;
static const int speedStep = 100;

static int radius = 0;
static const int radiusStep = 100;

bool isStopped ()
{
  return (speed < speedStep && speed > -speedStep);
}
```

[1] `http://www.gnu.org/software/ncurses/`

```
void handleRemoteControl (Create& robot)
{
  int ch = getch ();

  bool stop = isStopped ();

  if (ch == KEY_UP)
    {
      speed += speedStep;
      if (!stop && isStopped ())
        radius = 0;
    }
  if (ch == KEY_DOWN)
    {
      speed -= speedStep;
      if (!stop && isStopped ())
        radius = 0;
    }

  if (ch == KEY_LEFT)
    {
      if (radius < 0)
        radius += radiusStep;
      else if (radius == 0)
        radius = Create::RADIUS_MIN;
      else
        radius = 0;
    }
  if (ch == KEY_RIGHT)
    {
      if (radius > 0)
        radius -= radiusStep;
      else if (radius == 0)
        radius = Create::RADIUS_MAX;
      else
        radius = 0;
    }

  if (speed >= Create::VELOCITY_MAX)
    speed = Create::VELOCITY_MAX;
  if (speed <= Create::VELOCITY_MIN)
    speed = Create::VELOCITY_MIN;

  if (radius >= Create::RADIUS_MAX)
    radius = Create::RADIUS_MAX;
```

```
  if (radius <= Create::RADIUS_MIN)
    radius = Create::RADIUS_MIN;

  if (robot.bumpLeft () || robot.bumpRight ())
    speed = 0, radius = 0;

  std::stringstream ss;
  ss << "Speed: " << speed << ", Radius: "
     << radius << std::endl;
  mvprintw(0, 0, ss.str ().c_str ());
  refresh ();

  if (radius < radiusStep && radius > -radiusStep)
    robot.sendDriveCommand (speed,
                            Create::DRIVE_STRAIGHT);
  else if (speed < speedStep && speed > -speedStep)
    {
      int s = 0;
      if (radius > 0)
        s = (Create::RADIUS_MAX - radius) / 4;
      else
        s = (Create::RADIUS_MIN - radius) / 4;

      robot.sendDriveCommand (s,
                      Create::DRIVE_INPLACE_CLOCKWISE);
    }
  else
    robot.sendDriveCommand (speed, -radius);
}

int main (int argc, char** argv)
{
  if (argc < 2)
    return 1;

  initscr ();
  keypad (stdscr, TRUE);
  noecho ();

  SerialStream stream (argv[1]);

  try
    {
      Create robot (stream);
      robot.sendFullCommand ();
```

```
      while (true)
        {
          handleRemoteControl (robot);

          usleep(100 * 1000);
        }

      endwin ();
    }
  catch (InvalidArgument& e)
    {
      std::cerr << e.what () << std::endl;
      return 3;
    }
  catch (CommandNotAvailable& e)
    {
      std::cerr << e.what () << std::endl;
      return 4;
    }
}
```

Listing A.2: Remote control example source