Robot Theatre

Intelligent Robotics III

ECE 510/PSU, 06/14/2016

Introduction

The goal of this project was to investigate and implement capabilities necessary for the robot theatre using the HR-OS1 robot.

These capabilities are:

- Remote control of the robot over WiFi.
- Perform all action pages.
- Walk.

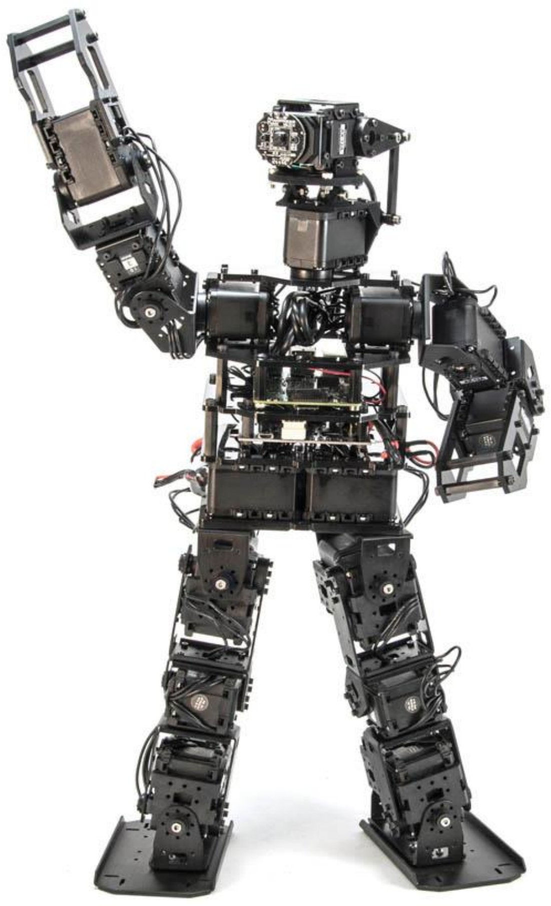


Figure 1. HR-OS1 aka Jimmy.

This was a very ambitious project at the end of which I am able to demonstrate the following:

- Use of Intel® System Studio for programming the HR-OS1 robot.
  This tool greatly improved my productivity however the initial setup wasn't without hassle.
- The use of the HR-OS1 Framework inside Intel® System Studio.
  I was able to port the HR-OS1 Framework and the rme and walk_tuner demos to Intel® System Studio.

- A client-server model for remote controlling the robot over Wi-Fi.
- The use of the ZeroMQ networking library for communication between the robot theatre server and client.
- A server app running on Intel Edison (which powers the HR-OS1 robot).
  This app listens for commands from a central computer – the robot theatre central controller.
- A client app running on a Windows PC.
  The client app sends commands to the robot and can be used to easily script a play for the robot theatre.

Jimmy Crane

The crane helps support the robot between the power cycles and it allows the robot to move freely without actually going anywhere. This makes it very convenient to concentrate only on programming and not worry about the robot as much.

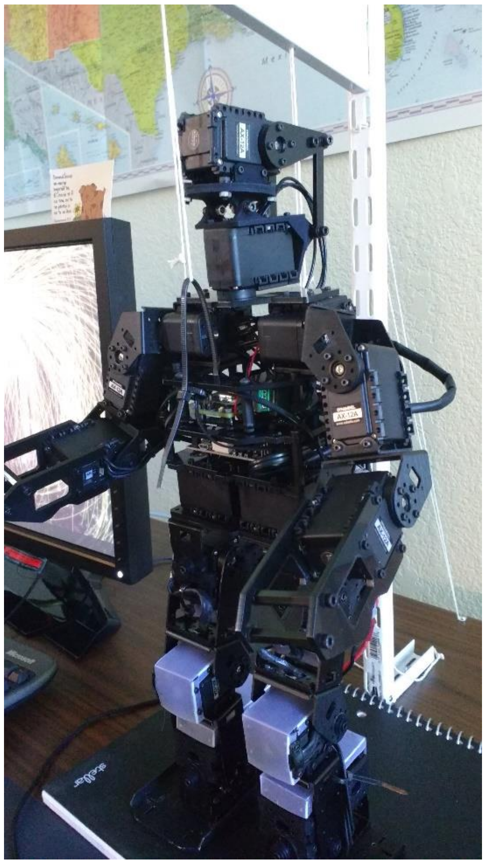


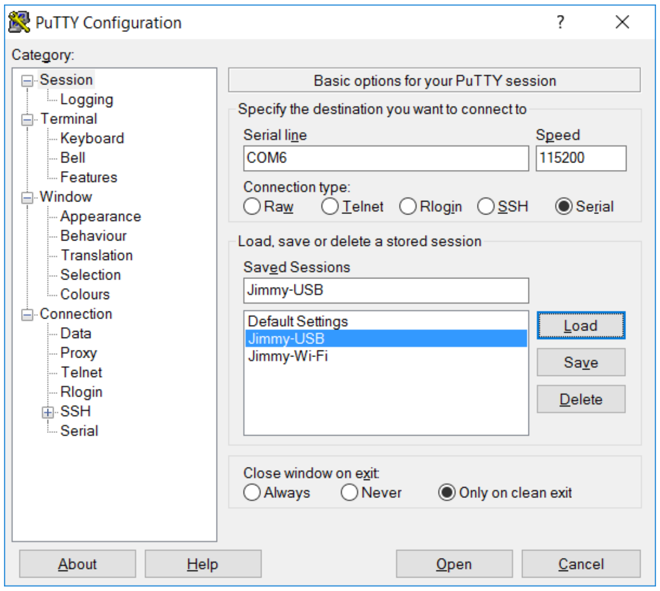
Figure 2. The crane system I used to support the robot during the software development phase.

Intel® System Studio

Intel® System Studio is an Eclipse based integrated development environment (IDE) used for programming Intel Edison remotely. This IDE installs on Windows, Linux and MacOS and provides developers with productivity tools to ease Intel Edison programming. Until now, I did my programming using *putty* and text editing tools like *nano*.

To recap:

- *Putty* is a tool used to establish a remote connection with Intel Edison. I connected to Intel Edison in two ways: over serial (wired USB connection) and over Wi-Fi. https://software.intel.com/en-us/setting-up-serial-terminal-on-system-with-windows



Specify the destination you want to connect to

Serial line: COM6
Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions: Jimmy-USB

Default Settings
Jimmy-USB
Jimmy-Wi-Fi

Close window on exit: Always Never [✓ Only on clean exit]

Figure 3. Use Putty to connect to Intel Edison (Serial over USB or Wi-Fi).

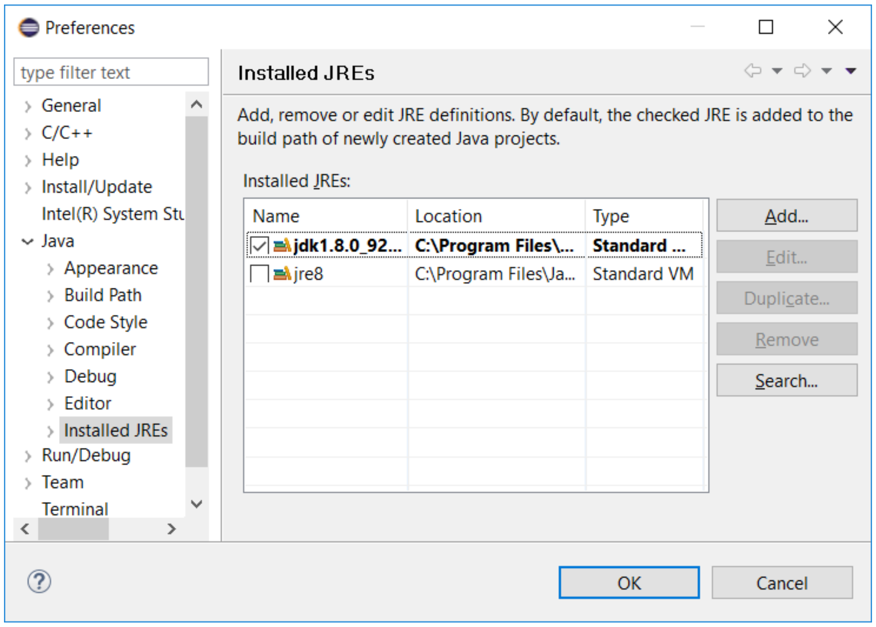
- *Nano* is a very simple text editor for Linux. This is not available on the Intel Edison image that comes out of the HR-OS1 box but there are many tutorials on the web about how to install it. Using *nano,* I was able to write my own simple demo based on the *ps3_demo* but *nano* is just a basic text editor and not suited for projects that are more complicated.

The *Intel® System Studio* on the other hand is a tool designed for Internet of Things (IOT) projects and offers many of the benefits of a modern IDE. Download here: https://software.intel.com/sites/landingpage/iotdk/windows-development-kit.html.

SparkFun has a very good installation tutorial: https://learn.sparkfun.com/tutorials/programming-the-intel-edison-beyond-the-arduino-ide. The tutorial is a bit outdated and I encountered these minor issues:

- No need to update the launcher script – already done in the latest version.
- *Java requirements.*
  At first run, the IDE might complain about the JDK version. It might still complain after installing

this prerequisite. In this case, go to Windows→Preferences→Java→Installed JRE and add the location of the JDK. On my computer this was "C:\Program Files\Java\jdk1.8.0_92".


Preferences

Installed JREs

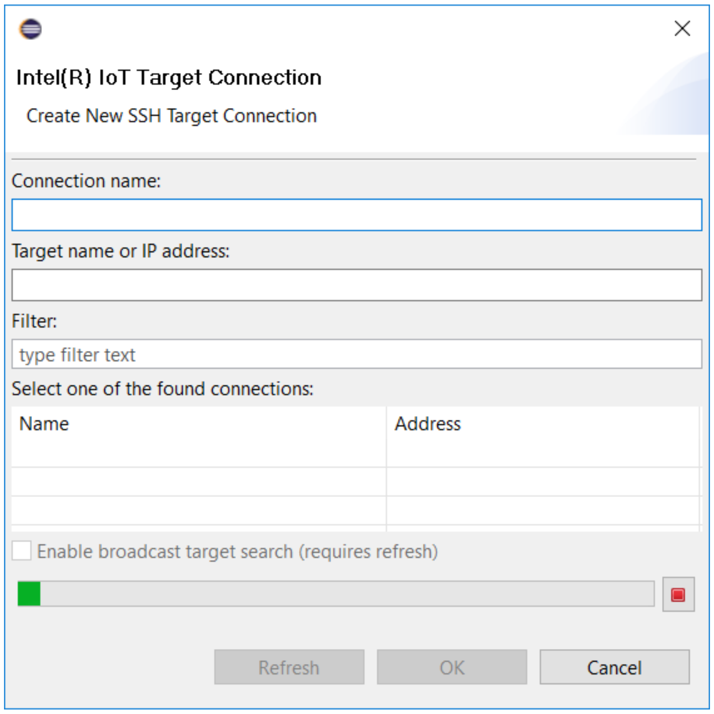
Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

Name	Location	Type
✓ jdk1.8.0_92...	C:\Program Files\...	Standard ...
jre8	C:\Program Files\Ja...	Standard VM

Figure 4. Java JDK 1.8 is necessary for Intel® System Studio to function correctly.

- *Create New SSH Target Connection*
  This dialog is different from the one in the SparkFun tutorial. I will try to auto-detect the Intel Edison settings but it did not work for me. Instead, stop the search and press ok after adding the *Host name* and the *Connection name*.


Intel(R) IoT Target Connection

Create New SSH Target Connection

Connection name:

Target name or IP address:

Filter:

type filter text

Select one of the found connections:

Name	Address

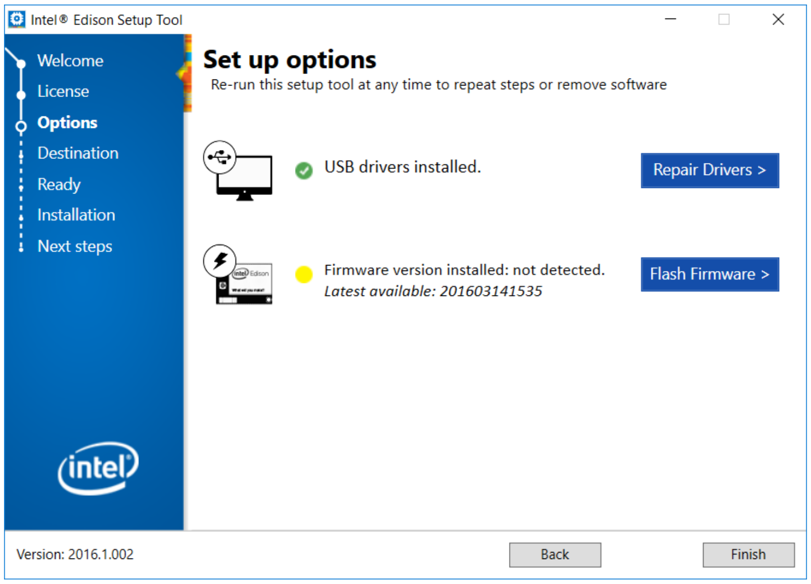
Enable broadcast target search (requires refresh)

Figure 5. Intel(R) System Studio - Create New SSH Target Connection.

After establishing a connection with the Intel Edison board, the IDE will offer to synchronize libraries. Pressing ok might reveal that the operation cannot complete because Intel Edison is running an older Linux version. This is why I decided to update the Intel Edison Linux image.

Intel Edison image update

In order to flash a new Intel Edison image go to: https://software.intel.com/en-us/iot/hardware/edison/downloads. Here you can download the setup tool for Windows, Linux or MacOS. The tool will install the latest FTDI drivers necessary to connect to the board using Serial over USB and will flash the latest Edison image. Intel Edison must be connected to your computer using two USB cables. The flashing operation will take a few minutes and it will erase all the files from Intel Edison so please remember to back up your files.


Intel® Edison Setup Tool

Set up options

Re-run this setup tool at any time to repeat steps or remove software

USB drivers installed.

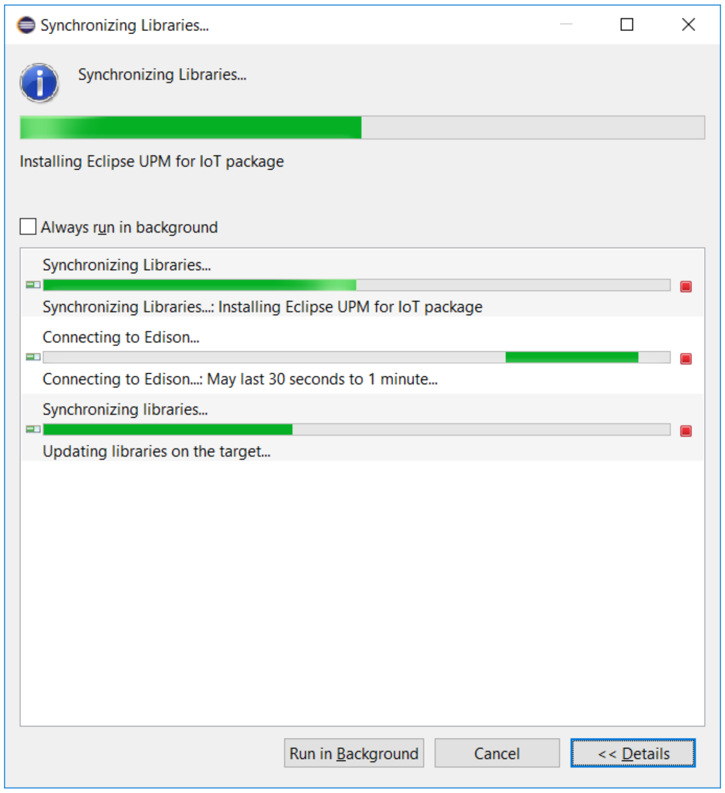
Firmware version installed: not detected.
Latest available: 201603141535

Version: 2016.1.002

Figure 6. Intel Edison Setup Tool. Please backup your files before flashing the new image.

The new image (160314) brings a few improvements like increased root partition size, *nano* is preinstalled, *and vim* is preinstalled as well (before only *vi* was available out of the box). In addition, the *putty* connection over Wi-Fi seems to have significantly improved. This procedure also resets the password.

After this the IDE was able to synchronize the libraries.


Synchronizing Libraries...

Synchronizing Libraries...

Installing Eclipse UPM for IoT package

Always run in background

Synchronizing Libraries...

Synchronizing Libraries...: Installing Eclipse UPM for IoT package

Connecting to Edison...

Connecting to Edison...: May last 30 seconds to 1 minute...

Synchronizing libraries...

Updating libraries on the target...

Figure 7. Synchronizing libraries between the Intel Edison and the Eclipse based Intel® System Studio IDE.

Install/Build the HR-OS1 Framework

The flashing operation erased all files so we need to go back to the HR-OS1 Intel Edison setup: https://github.com/Interbotix/HROS1-Framework/wiki/HR-OS1-Electronics-and-Framework-Setup-%28Intel-Edison%29.

Setup Intel Edison password and Wi-Fi using the following command:

configure_edison --setup

Download the HR-OS1 framework:

git clone https://github.com/Interbotix/HROS1-Framework.git

Build the framework:

cd ~/HROS1-Framework/Linux/build
make clean && make all

The build fails – it is missing a libjpeg.h file. This is because the *libjpeg-dev* component is not part of the latest Intel Edison flash image. In order to install it follow these steps:
https://communities.intel.com/message/265911#265911
http://alextgalileo.altervista.org/edison-package-repo-configuration-instructions.html
https://communities.intel.com/thread/55692
http://intel1413.rssing.com/chan-22403226/all_p310.html

Edit */etc/opkg/base-feeds.conf* and replace its contents:

```
src/gz all http://repo.opkg.net/edison/repo/all
src/gz edison http://repo.opkg.net/edison/repo/edison
src/gz core2-32 http://repo.opkg.net/edison/repo/core2-32
```

```
opkg install libjpeg-dev
```

Repeat the build steps – the build succeeds.

Build the rme project:

```
cd ~/HROS1-Framework/Linux/project/rme
make
```

Run the rme project:

```
./rme
```

RME (Robot Motion Editor) fails to run with the error message:

```
Fail to open port
Arbotix Pro is used by another program or do not have root privileges.
```

This is because Intel Edison needs FTDI drivers to connect to the Arbotix Pro servo controller. Use this command to install them:

```
opkg install kernel-module-ftdi-sio
```

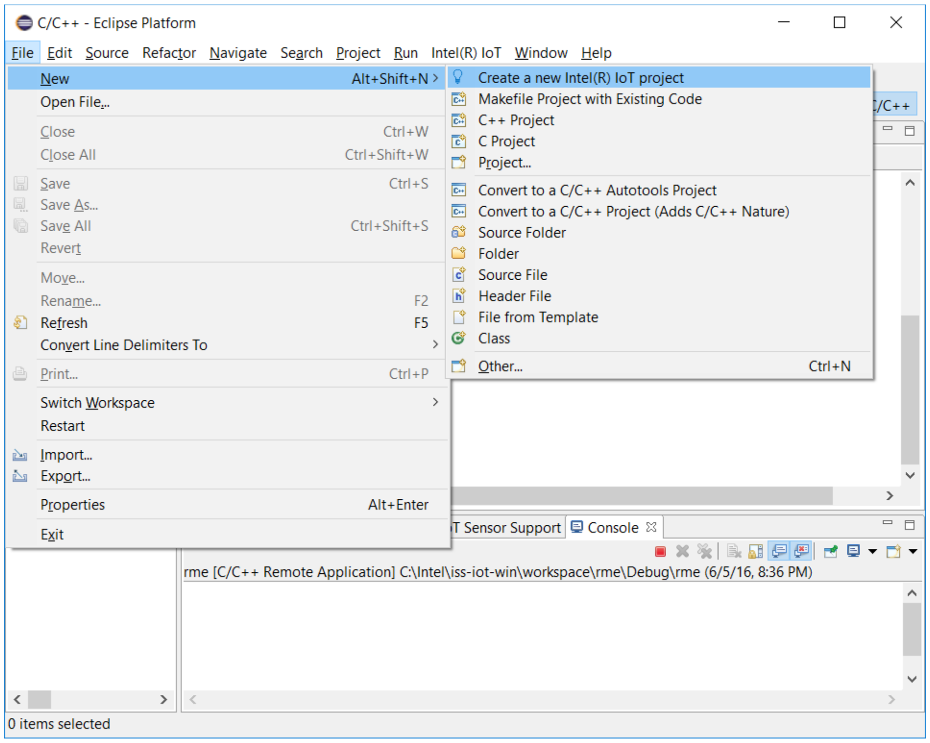
RME ran successfully after this.

I would mention that the IDE also replaces *putty* entirely and I've used it for all of the above operations.

Hello Jimmy

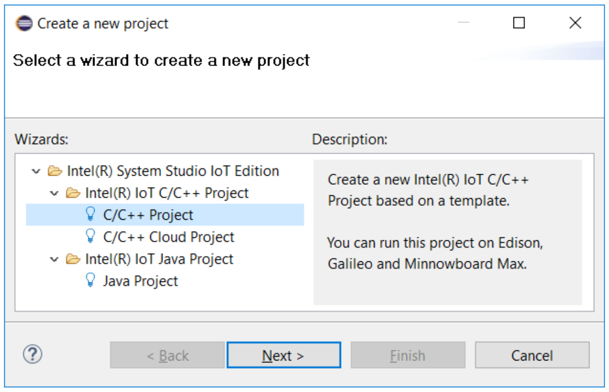
This is the first and the simplest project I created.

1. Create a new project
   File→New→Create a new Intel® IoT project.



```
rme [C/C++ Remote Application] C:\Intel\iss-iot-win\workspace\rme\Debug\rme (6/5/16, 8:36 PM)
```

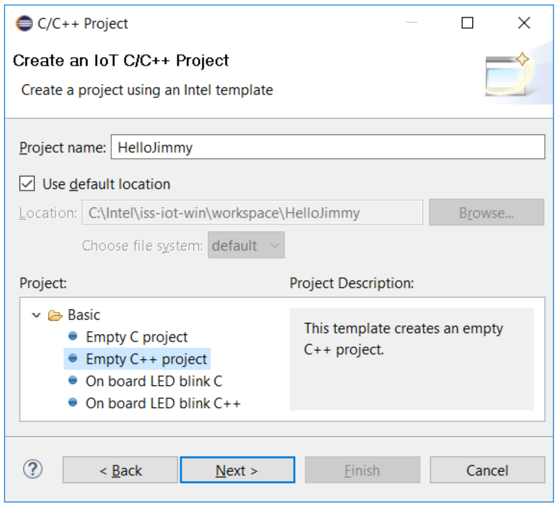
2. Select a new project wizard
   Intel® System Studio IoT Edition → Intel® IoT C/C++ Project → C/C++ Project


Create a new project
Select a wizard to create a new project
Wizards:
Description:
Intel(R) System Studio IoT Edition
Intel(R) IoT C/C++ Project
C/C++ Project
C/C++ Cloud Project
Intel(R) IoT Java Project
Java Project
< Back
Next >
Finish
Cancel

Create a new Intel(R) IoT C/C++ Project based on a template.

You can run this project on Edison, Galileo and Minnowboard Max.

3. Name the project and select type



Create an IoT C/C++ Project

Create a project using an Intel template

Project name: HelloJimmy

☑ Use default location

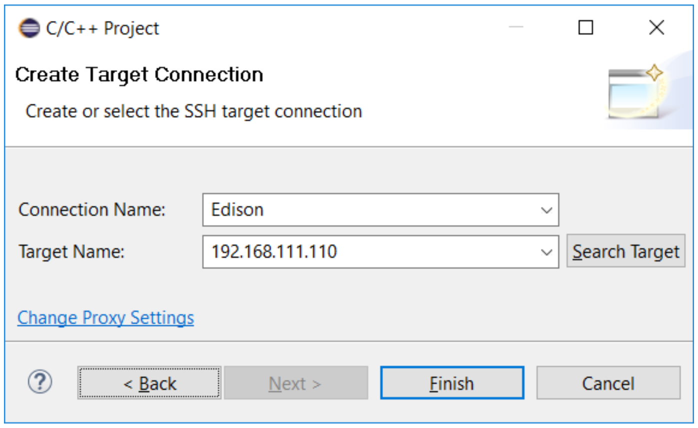
Location: C:\Intel\iss-iot-win\workspace\HelloJimmy

Choose file system: default

Project:
- Basic
  - Empty C project
  - Empty C++ project
  - On board LED blink C
  - On board LED blink C++

Project Description: This template creates an empty C++ project.

4. Select the target connection and press Finish.



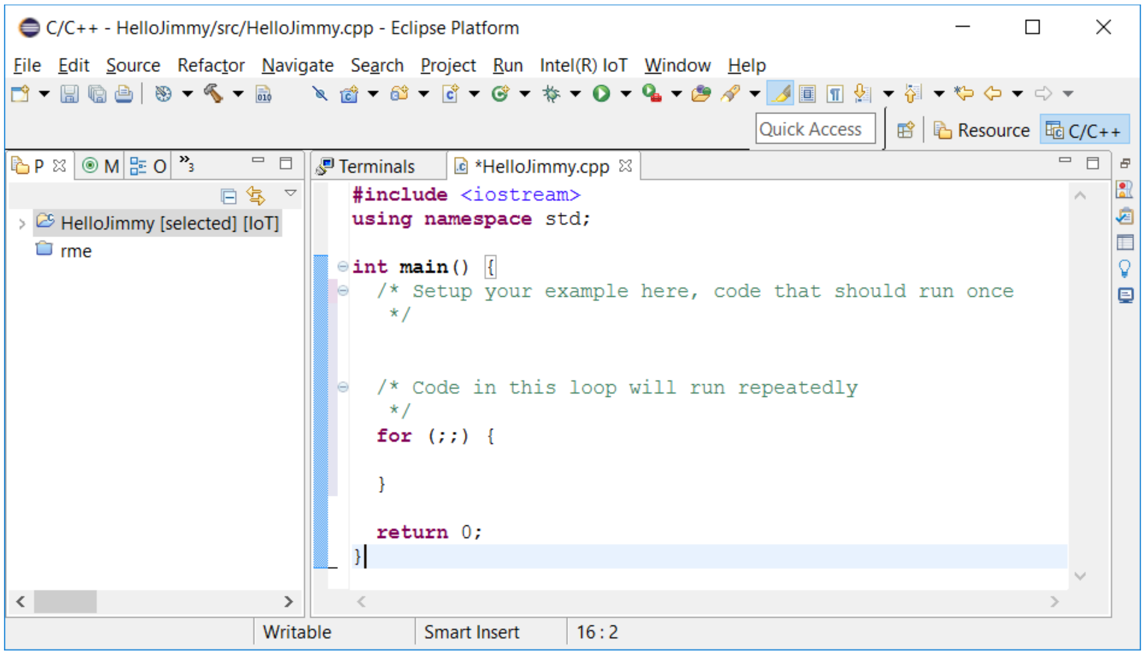
Create Target Connection

Create or select the SSH target connection

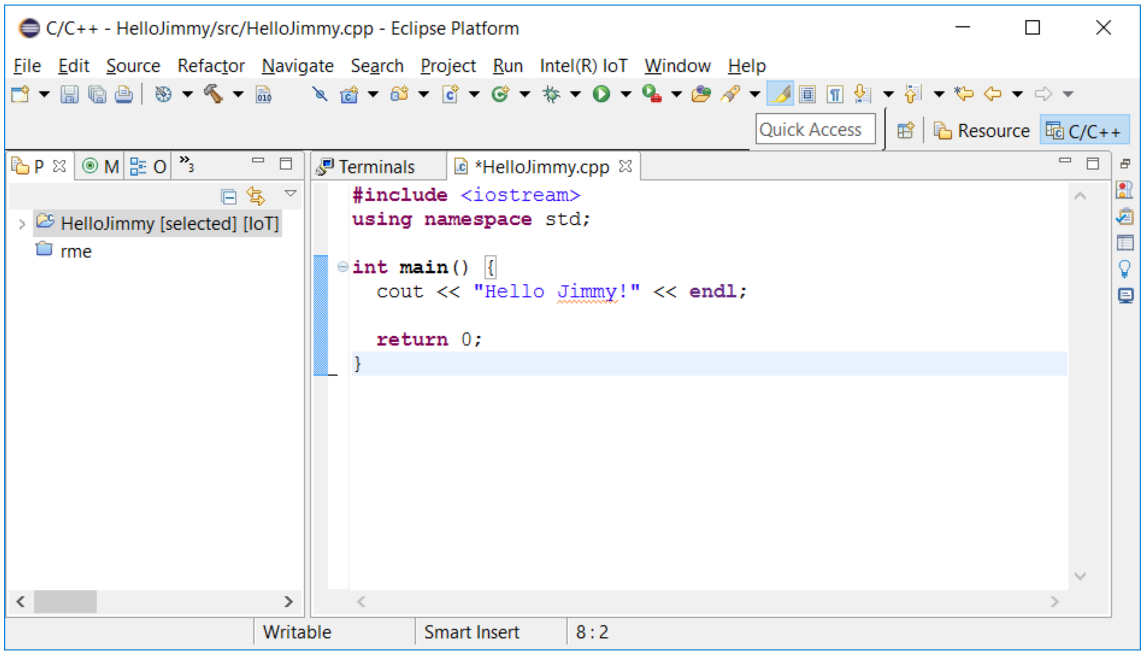
Connection Name: Edison

Target Name: 192.168.111.110

5. Add our simple code.
   Notice that the “Empty C++ Project” template actually comes with a main().

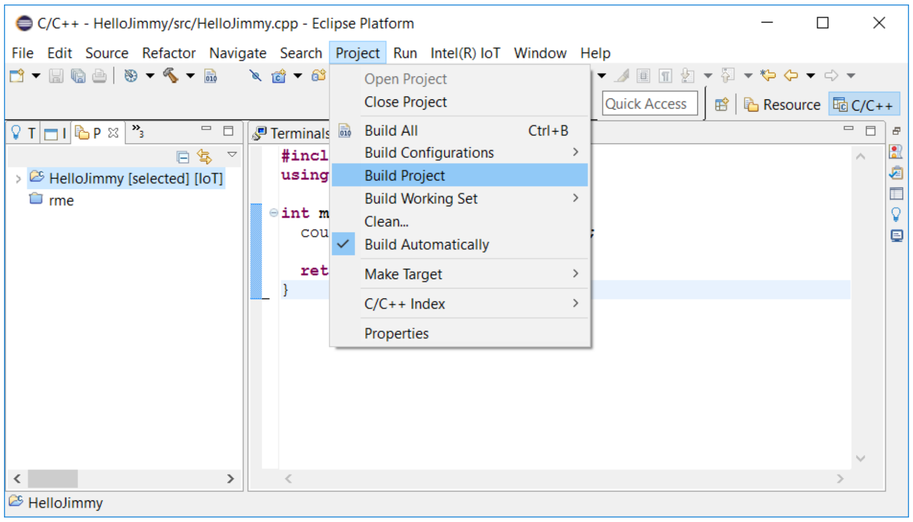

C/C++ - HelloJimmy/src/HelloJimmy.cpp - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Intel(R) IoT Window Help
Quick Access
Resource
C/C++
Terminals
*HelloJimmy.cpp
HelloJimmy [selected] [IoT]
rme
Writable
Smart Insert
16 : 2

```
#include <iostream>
using namespace std;

int main() {
	/* Setup your example here, code that should run once
	 */


	/* Code in this loop will run repeatedly
	 */
	for (;;) {

	}

	return 0;
}
```

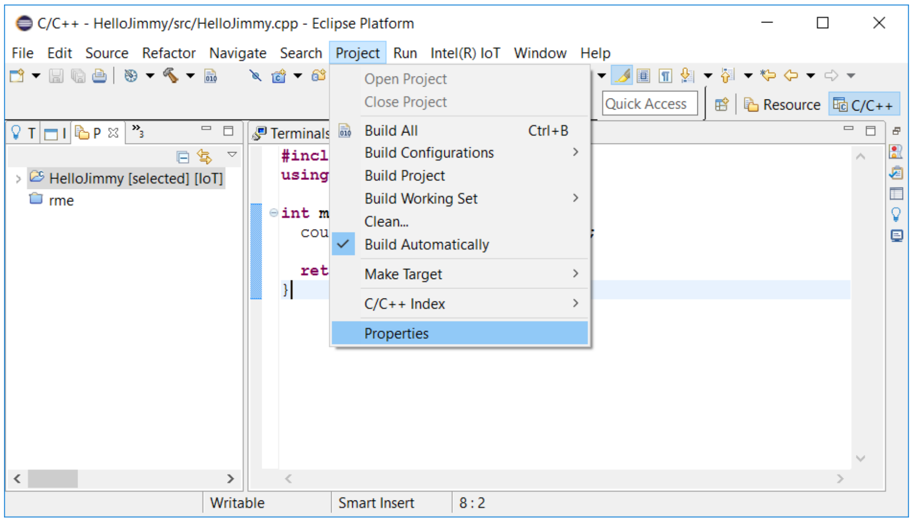

C/C++ - HelloJimmy/src/HelloJimmy.cpp - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Intel(R) IoT Window Help
Quick Access
Resource
C/C++
HelloJimmy [selected] [IoT]
rme
Terminals
*HelloJimmy.cpp
Writable
Smart Insert
8 : 2

```
#include <iostream>
using namespace std;

int main() {
  cout << "Hello Jimmy!" << endl;

  return 0;
}
```

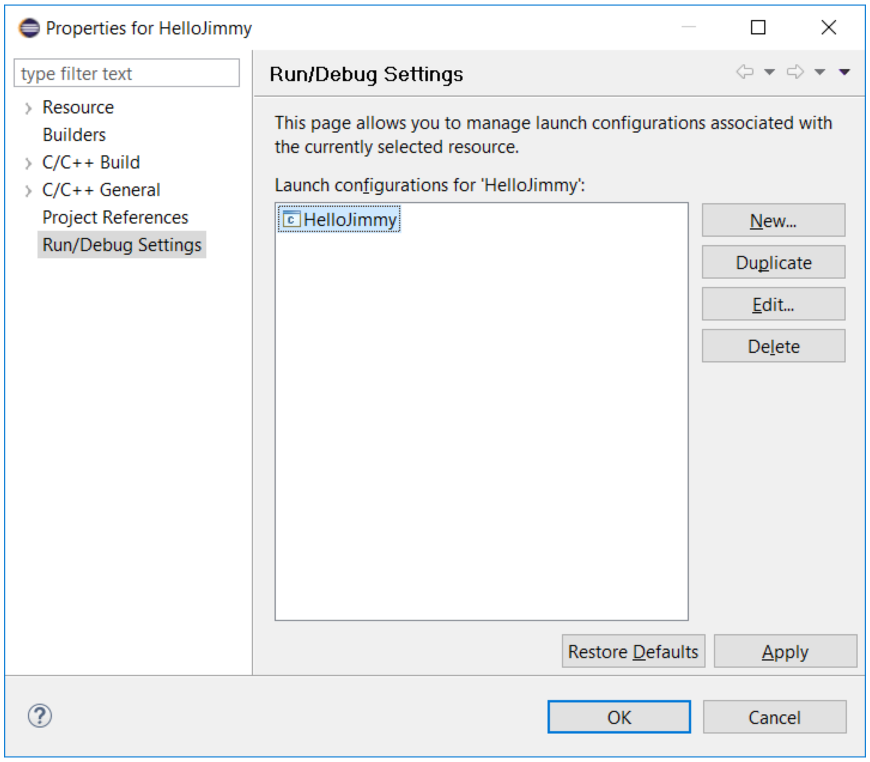
6. Build & run the project
   Project → Build Project
   Run → Run
   Run → Debug
   Selecting Run or Debug will cause the executable to be copied to Jimmy to the default location which is /tmp/.



7. Change the default destination of the HelloJimmy executable.
   Project → Properties → Run/Debug Settings → Edit
   Change /tmp/HelloJimmy to /home/root/HelloJimmy for example and Run/Debug again (in order to copy the executable to Jimmy).



```
#incl
using

int m
  cou

  ret
}
```

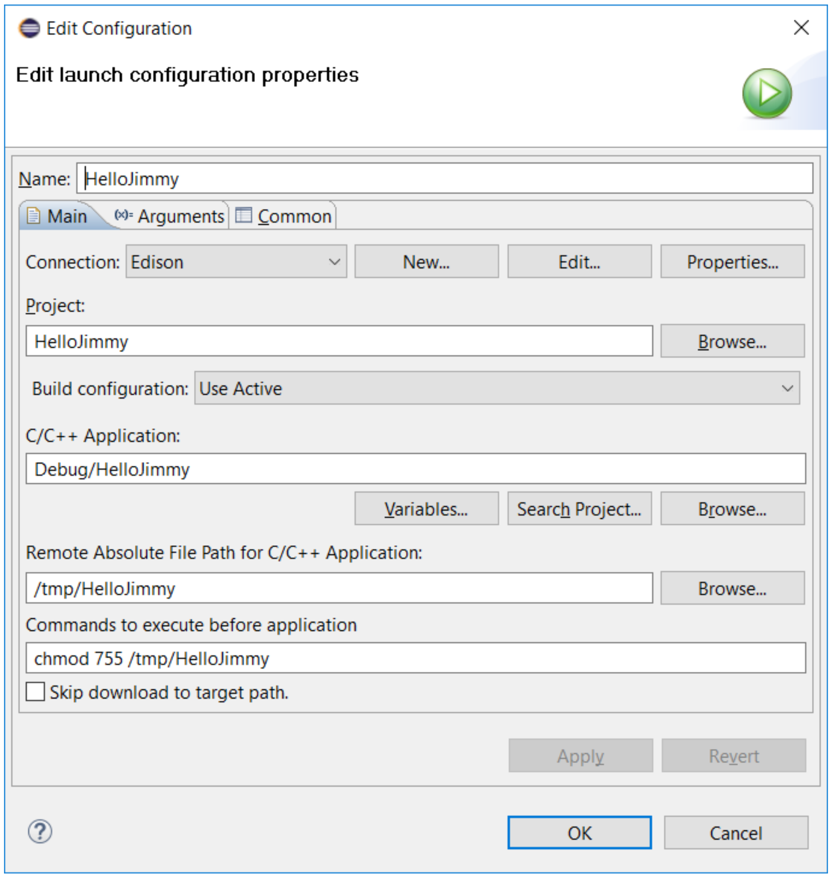


Run/Debug Settings

This page allows you to manage launch configurations associated with the currently selected resource.

Launch configurations for 'HelloJimmy':

HelloJimmy



Name: HelloJimmy

Connection: Edison

Project: HelloJimmy

Build configuration: Use Active

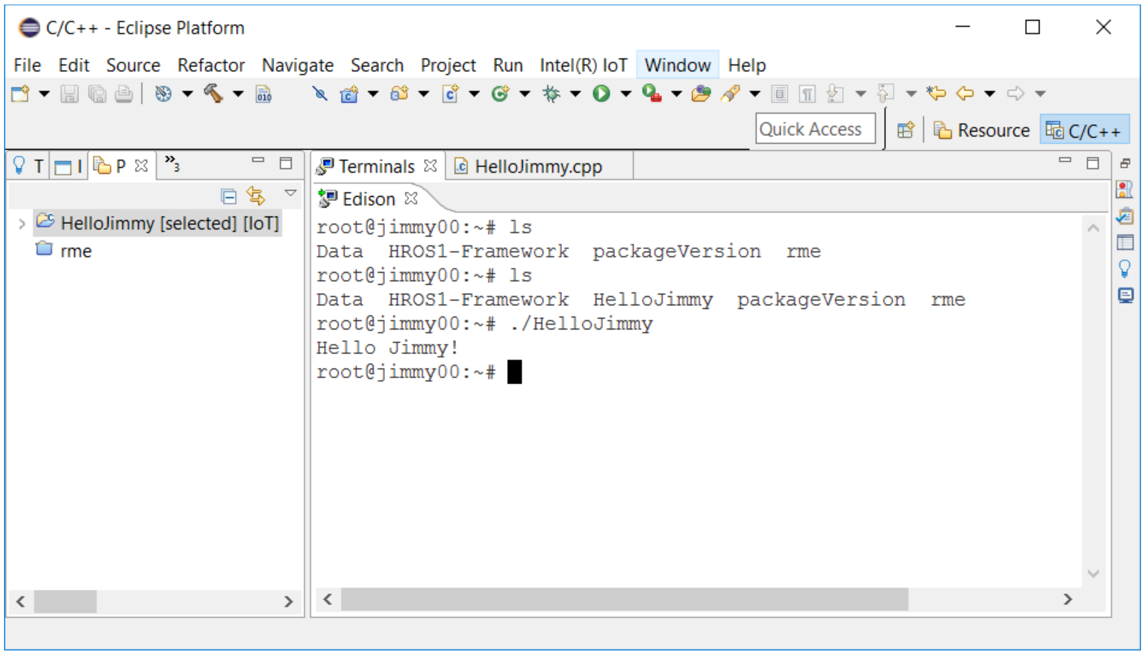
C/C++ Application: Debug/HelloJimmy

Remote Absolute File Path for C/C++ Application: /tmp/HelloJimmy

Commands to execute before application: chmod 755 /tmp/HelloJimmy

☐ Skip download to target path.

8. List files and confirm that HelloJimmy was copied to the right location. Run HelloJimmy.


C/C++ - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Intel(R) IoT Window Help
Quick Access
Resource
C/C++
HelloJimmy [selected] [IoT]
rme
Terminals
HelloJimmy.cpp
Edison

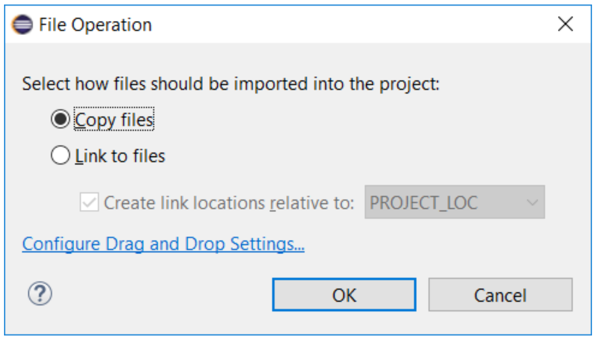
```
root@jimmy00:~# ls
Data  HROS1-Framework  packageVersion  rme
root@jimmy00:~# ls
Data  HROS1-Framework  HelloJimmy  packageVersion  rme
root@jimmy00:~# ./HelloJimmy
Hello Jimmy!
root@jimmy00:~#
```

RME

RME (the Robot Motion Editor) is one of the first demos to try on Jimmy once the assembly is complete. It is also a great example of how to program Jimmy so it was a good next step to try to build it using the Intel® System Studio.

This is a much more complicated task. First steps are described above in the creation of our simplest project, HelloJimmy so let's assume you've already created a project called rme.

1. Delete rme.cpp. The C++ project template is responsible for the creation of this file.
    a. Assuming you cloned the HR-OS1 code from GitHub:
       Go to <GitHub clone path>/HROS1-Framework/Linux/project/rme and select the following files: main.cpp, cmd_process.h and cmd_process.cpp. Now drag and drop the files over the 'src' folder in the rme project. Select Copy Files to add copies to the rme project.



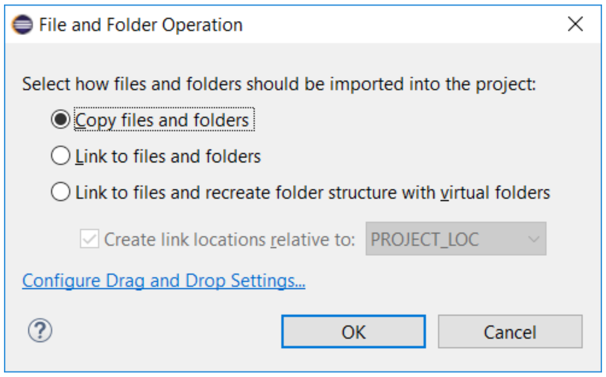
File Operation

Select how files should be imported into the project:

- (●) Copy files
- ( ) Link to files

☑ Create link locations relative to: PROJECT_LOC

2. Build the project.
   An error message will show up:
   ..\src\cmd_process.h:5:25: fatal error: LinuxDARwIn.h: No such file or directory
3. Got to <GitHub clone path>/HROS1-Framework/Linux/project/rme and select these folders: Framework and Linux. Drag and drop over the rme project name. Select Copy files and folder to have the framework code copied to the rme project.



File and Folder Operation

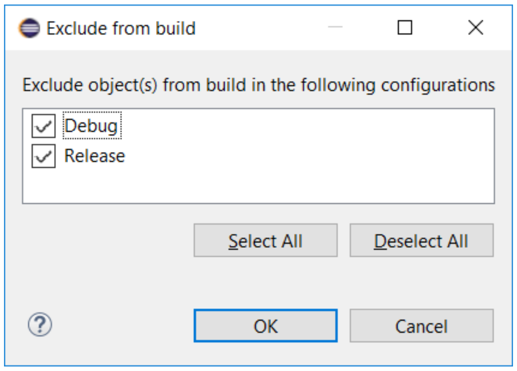
Select how files and folders should be imported into the project:

- (●) Copy files and folders
- ( ) Link to files and folders
- ( ) Link to files and recreate folder structure with virtual folders

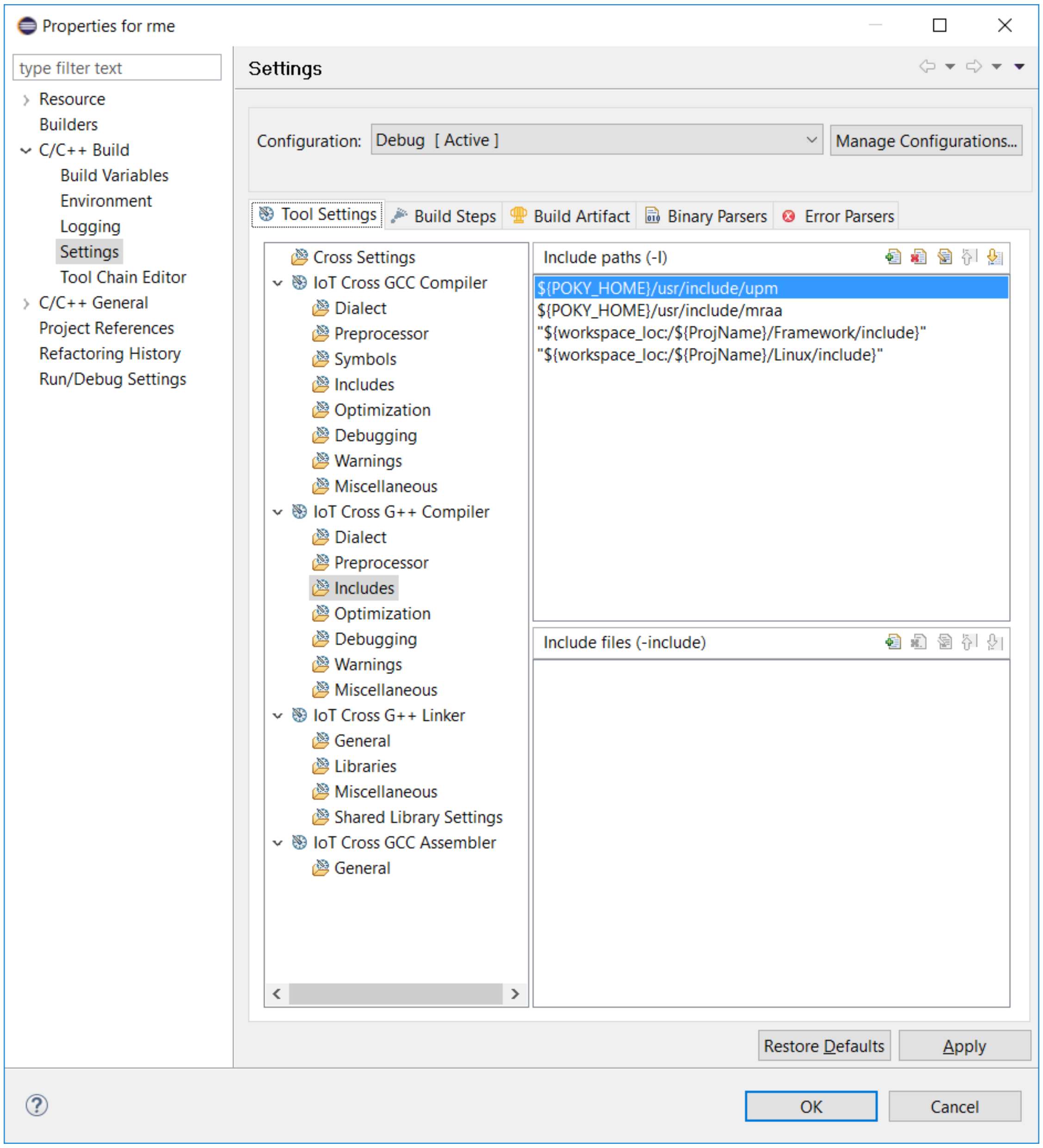
☑ Create link locations relative to: PROJECT_LOC

4. In the rme project select Linux/project → right click → Delete. This is done in order to remove unnecessary files for this project.
5. Build the project again in order to identify the next set of conflicts that need addressing. You will notice that nothing new actually happens. This is because the files in the recently added folders (Framework and Linux) are not yet part of the build process.

On each folder: Right click → Resource Configurations → Exclude from build → Deselect both Debug and Release and press Ok. Build the project again.


Exclude from build
Exclude object(s) from build in the following configurations
Debug
Release
Select All
Deselect All
OK
Cancel

6. Still nothing happens.
   At this point we have to set the include directories. These are:
   <rme_path>/Framework/include/
   <rme_path>/Linux/include/
   In order to do this go to the project properties → C/C++ Build → Settings → Tool Settings → IoT Cross G++ Compiler → Includes.
   Add the above two paths relative to the workspace.
   Build the project.



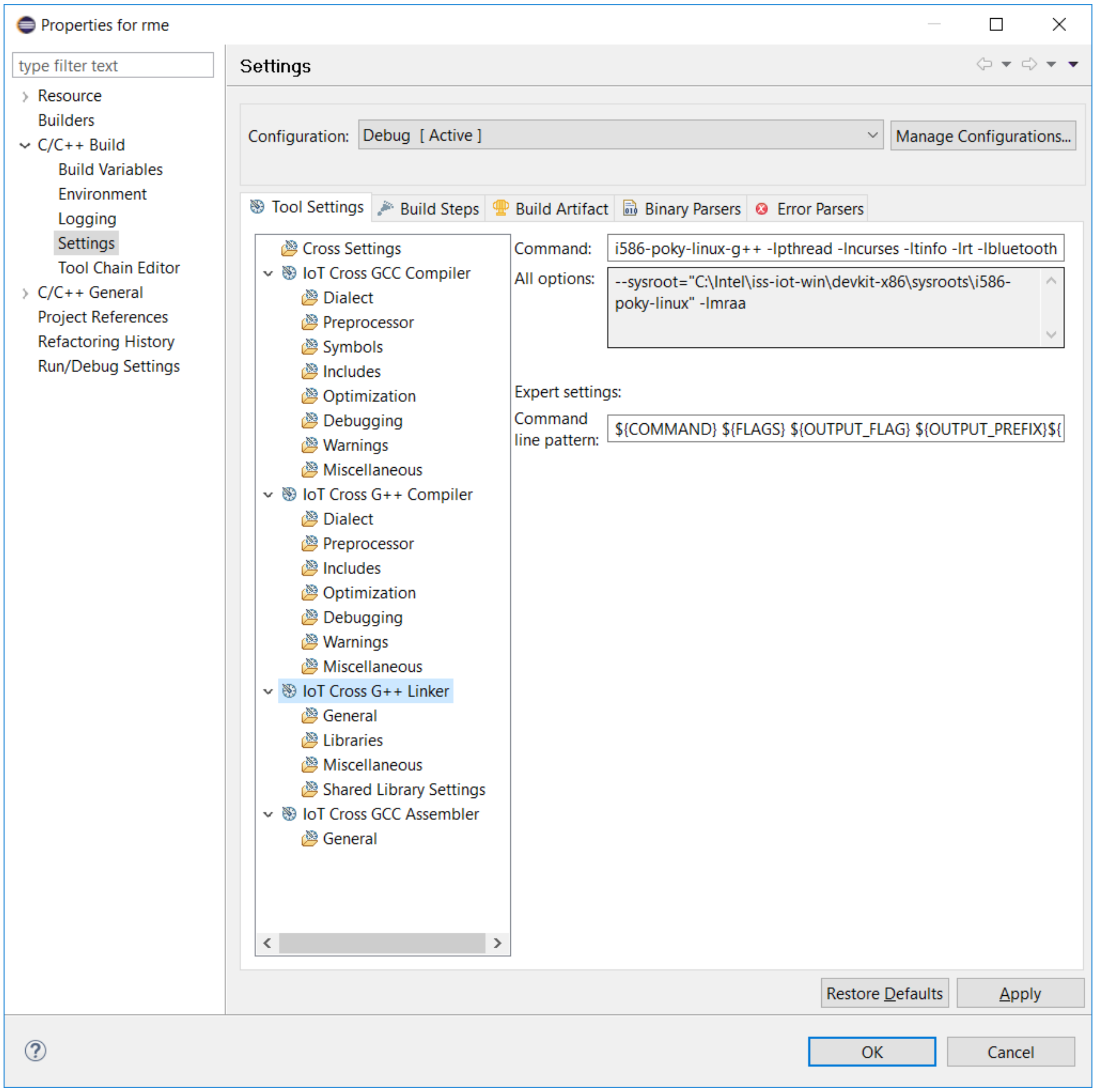
Configuration: Debug [Active] Manage Configurations...

Include paths (-I)

```
${POKY_HOME}/usr/include/upm
${POKY_HOME}/usr/include/mraa
"${workspace_loc:/${ProjName}/Framework/include}"
"${workspace_loc:/${ProjName}/Linux/include}"
```

Include files (-include)

7. We get a new error message:
   Linux\build\streamer\mjpg_streamer.o: undefined reference to symbol 'pthread_create@@GLIBC_2.1'
   This means we need to update the linker settings.
   Go to project properties → C/C++ Build → Settings → Tools Settings → IoT G++ Linker and append the following to the Command field:
   -lpthread -lncurses -ltinfo -lrt -lbluetooth
   Build the project.



Configuration: Debug [Active]

Command: i586-poky-linux-g++ -lpthread -lncurses -ltinfo -lrt -lbluetooth

All options: --sysroot="C:\Intel\iss-iot-win\devkit-x86\sysroots\i586-poky-linux" -lmraa

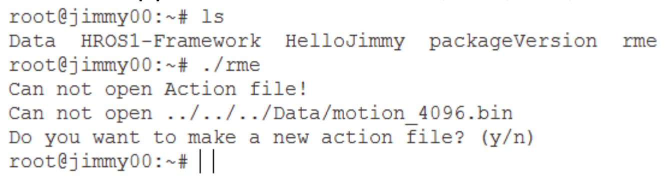
Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${

8. The build process generates multiple errors concerning Jimmy vision support. We don't need it for this project so it is safe to exclude from the build (or delete) the following folders/files from the project:
   Linux/build/streamer folder
   Linux/build/LinuxCamera.cpp
   Linux/include/LinuxCamera.h
   Framework/include/RegionProps.h
   Framework/src/vision/ConnectRegions.cpp
   Framework/include/ConnectRegions.h

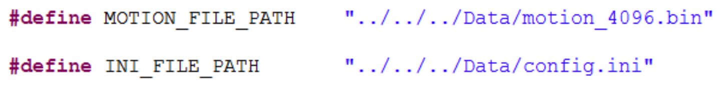
   The project builds successfully.

9. Run the project. This will copy the rme executable to /host/root/rme.



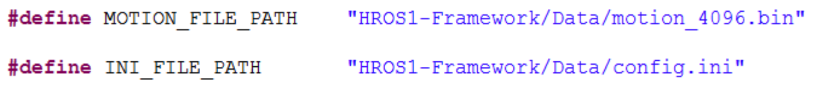
```
root@jimmy00:~# ls
Data  HROS1-Framework  HelloJimmy  packageVersion  rme
root@jimmy00:~# ./rme
Can not open Action file!
Can not open ../../../Data/motion_4096.bin
Do you want to make a new action file? (y/n)
root@jimmy00:~# ||
```

10. The error message displayed tells us that now we need to tell rme where to find the configurations files.



```
#define MOTION_FILE_PATH    "../../../Data/motion_4096.bin"

#define INI_FILE_PATH       "../../../Data/config.ini"
```

Modify main.cpp. This will tell rme where to look for the configuration files.



```
#define MOTION_FILE_PATH    "HROS1-Framework/Data/motion_4096.bin"

#define INI_FILE_PATH       "HROS1-Framework/Data/config.ini"
```

11. Done.

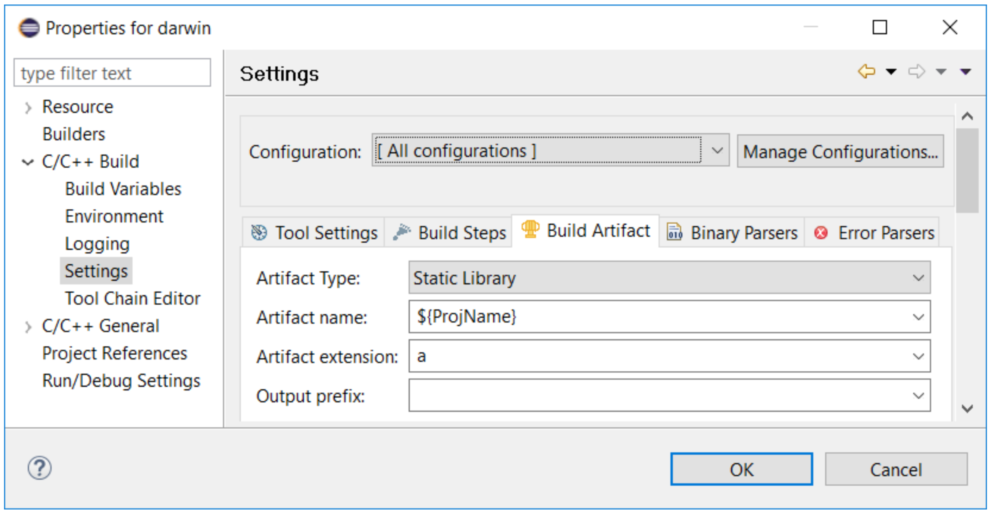
A better way to program in Intel® System Studio

The previous rme project for Intel® System Studio included the HR-OS1 Framework source files directly in the project. This is fine for one project but it is counterproductive when dealing with multiple projects. It is desirable to group common files in a library.

Darwin

The original setup of the HR-OS1 Framework produces a 'darwin.a' static library. This library is linked into all of the demo projects, like rme and walk_tuner. We can produce the darwin library in Intel® System Studio following these steps:

1. Create a new project called darwin in a similar manner as described for the previous rme project.
2. Delete the src folder and contents. There's no need for a main() in a static library.
3. Add the HR-OS1 Framework files to the project. (folders Framework and Linux from GitHub).
4. Delete the Linux/project files (the demo projects provided with the framework).
5. Exclude the following files/folders from the build. Make sure to exclude them both from Debug and Release.
    a. Linux/build/streamer folder
    b. Linux/build/LinuxCamera.cpp
    c. Linux/include/LinuxCamera.h
    d. Framework/include/RegionProps.h
    e. Framework/src/vision/ConnectRegions.cpp
    f. Framework/include/ConnectRegions.h
6. Set the Framework and Linux include folders relative to the workspace.
   Project Properties → C/C++ Build → Settings → Tool Settings → IoT Cross G++ Compiler → Includes.
7. Set Artifact Type to Static Library.
   Project properties → C/C++ Build → Settings → Build Artifact → Artifact Type → Static Library.
   Delete the output prefix (default is lib).



Configuration: [All configurations]

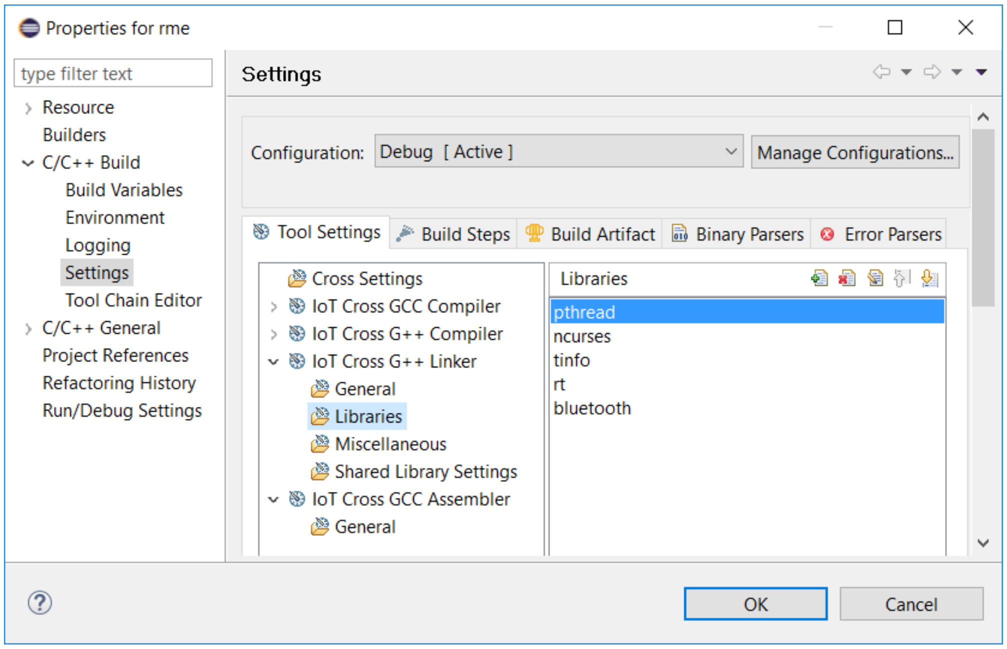
Artifact Type: Static Library
Artifact name: ${ProjName}
Artifact extension: a
Output prefix:

8. Build.

RME v2.0

This describes the steps necessary to create a project that links the darwin.a static library.

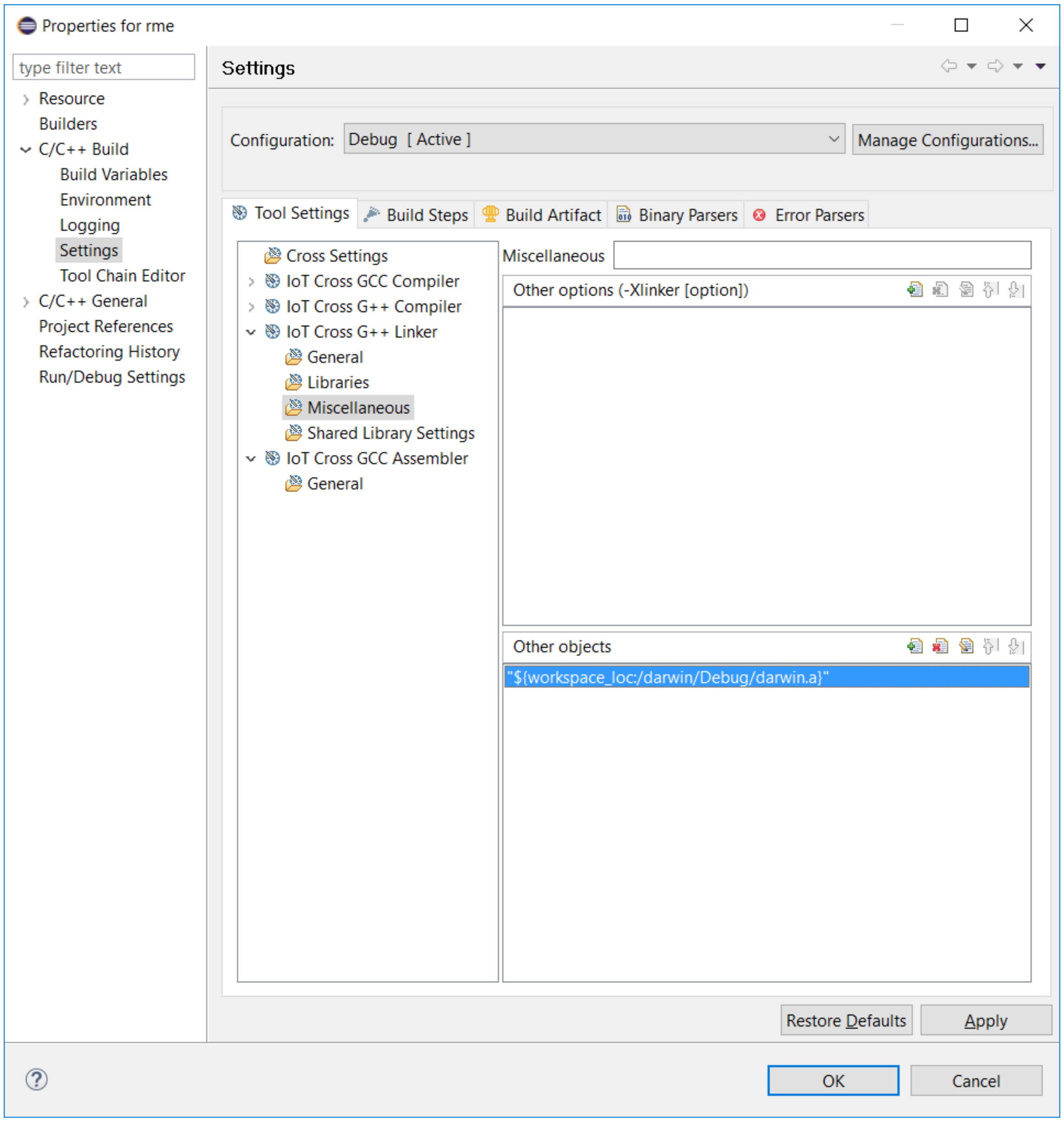
1. Create a new project called rme (delete the previous one with the same name).
2. Delete the rme.cpp file.
3. Add the rme project files:
   a. main.cpp
      Remember to change the paths to the motion and ini files.
   b. cmd_process.h
   c. cmd_process.cpp
4. Set the include paths relative to the workspace:
   a. darwin/Framework/include and
   b. darwin/Linux/include.
5. Set the library dependencies:
   Project Properties → C/C++ General → Paths and Symbols → Libraries →Add
   a. pthread
   b. ncurses
   c. tinfo
   d. rt
   e. bluetooth



Configuration: Debug [Active]

Libraries
pthread
ncurses
tinfo
rt
bluetooth

6. Build project.
   Lots of undefined symbols errors. These symbols reside in the darwin.a static library.
7. Specify the workspace path to the darwin.a static library.
   Project Properties -> C/C++ Build → IoT Cross G++ Linker → Miscellaneous



Configuration: Debug [Active]

Miscellaneous

Other options (-Xlinker [option])

Other objects

"${workspace_loc:/darwin/Debug/darwin.a}"

8. Build the project. Success.

ZeroMQ

ZeroMQ is a highly versatile open source embeddable networking library. In this project, I used it to implement the communication between the robot theatre controller and the HR-OS1 robot.

The robot theatre controller (*rtclient* – robot theatre client) is a Windows based app that instantiates a ZeroMQ client and connects to a ZeroMQ server (*rtserver* – robot theatre server) running on the HR-OS1 robot. The current list of supported command line arguments is:

1. *'-ip' ipAddress*
2. *'-cmd' command*
   where command can be
   - *walk*
   - *action*
   - *exit*
3. *'-q' quanta*
   specifies the action page to execute on the '*action'* command

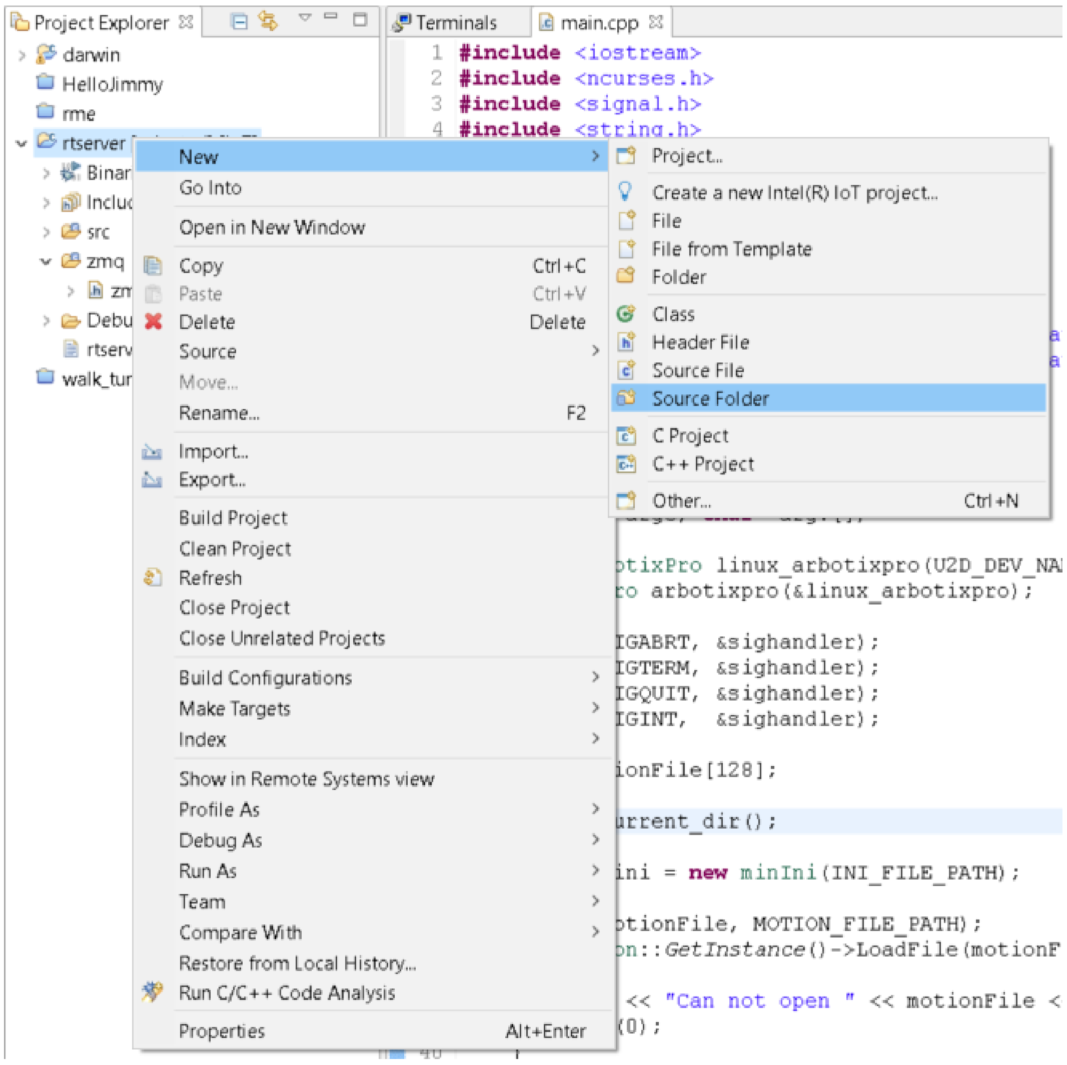
In the current revision, the robot can execute any of the 256 action pages designed using the robot motion editor (rme) and it can walk forward in a straight line. The robot is capable of walking backwards and of turning in any direction – we plan to enable this capability in the future revisions of the scurrent client-server model.

ZeroMQ comes with the latest Intel Edison image so installation is not necessary on HR-OS1. On Windows I downloaded *libzmq* from the GitHub repository (https://github.com/zeromq/libzmq.git) and followed this guide to build it for Visual Studio 2015 (http://zeromq.org/build:_start).

rtserver

Using Intel® System Studio I created the *s* project following these steps:

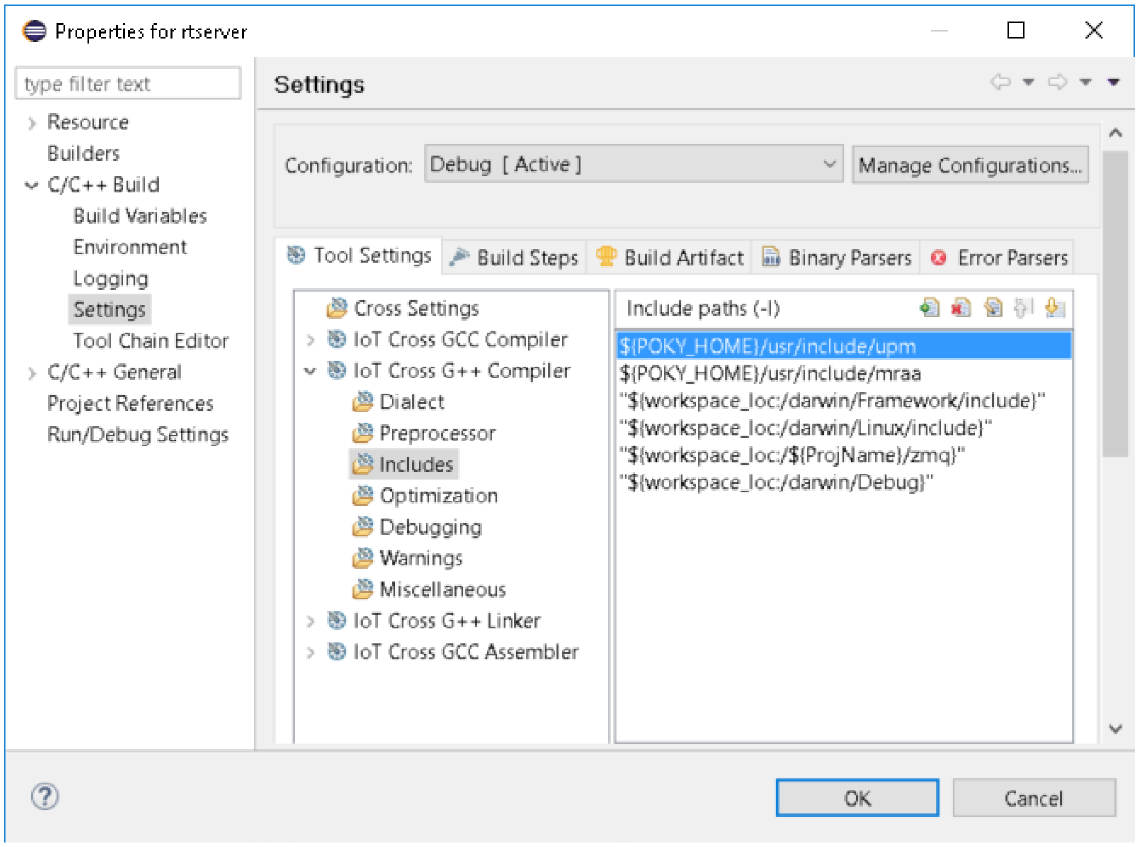
1. Create a new Intel® IoT project named rtserver.
   Follow the RME 2.0 guideline on how to link to the *darwin.a* library.
2. Add a new source folder called zmq.
   Copy here the *zmq.hpp* header file from https://github.com/zeromq/cppzmq.



```
#include <iostream>
#include <ncurses.h>
#include <signal.h>
#include <string.h>
```

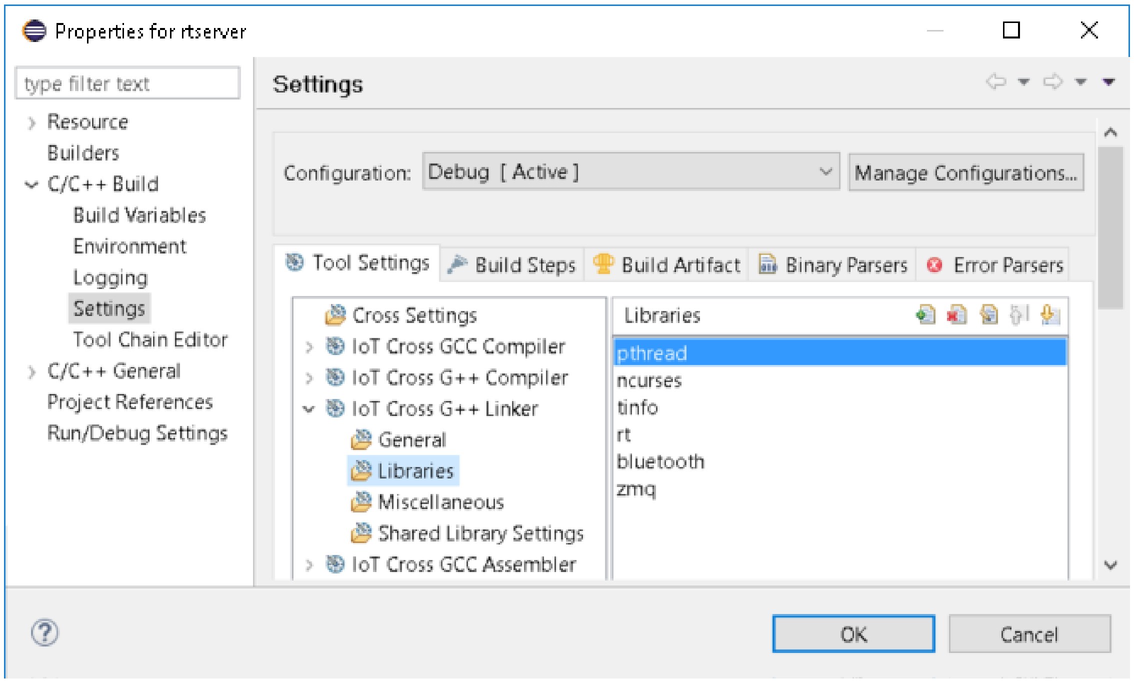
```
ptixPro linux_arbotixpro(U2D_DEV_NA
ro arbotixpro(&linux_arbotixpro);

IGABRT, &sighandler);
IGTERM, &sighandler);
IGQUIT, &sighandler);
IGINT,  &sighandler);

ionFile[128];

urrent_dir();

ini = new minIni(INI_FILE_PATH);

otionFile, MOTION_FILE_PATH);
on::GetInstance()->LoadFile(motionF

 << "Can not open " << motionFile <
(0);
```

3. Set the include directories.



```
${POKY_HOME}/usr/include/upm
${POKY_HOME}/usr/include/mraa
"${workspace_loc:/darwin/Framework/include}"
"${workspace_loc:/darwin/Linux/include}"
"${workspace_loc:/${ProjName}/zmq}"
"${workspace_loc:/darwin/Debug}"
```

4. Specify the library dependencies.



```
pthread
ncurses
tinfo
rt
bluetooth
zmq
```

5. Add code. Build. Run.

I implemented the rtserver code starting from the ZeroMQ C++ server example found here: http://zguide.zeromq.org/cpp:hwserver.

The server runs an infinite loop and listens for commands from an rtclient. The current revision can make the robot walk forward and perform any of the 256 action pages designed with the robot motion editor (rme). On receiving the *exit* command, the server performs the necessary clean-up operations and shuts down the robot servos.

rtclient

I created the rtclient for Windows using Visual Studio 2015. There are two aspects to keep in mind:

1. Use the dynamic *libzmq.lib* when specifying the library dependencies.
   This implies the need to copy the appropriate dlls to the output directory of your project.
2. The *zmq.hpp* include file is not part of the *libzmq* GitHub repository (https://github.com/zeromq/libzmq.git). It is available at this location: https://github.com/zeromq/cppzmq. Copy *zmq.hpp* to the *libzmq* include folder.

I implemented the rtclient code starting from the ZeroMQ C++ client example found here: http://zguide.zeromq.org/cpp:hwclient.

rtclient reads the command line arguments, sends one command to the server and exits. The client app can be used to command multiple robots at the same time.

Conclusion

This was a very challenging project, particularly because, at least to my knowledge, this is the first time all the listed software components where put together in this combination. I was able to re-image Intel Edison, install Intel® System Studio, port the HR-OS1 Framework to Intel® System Studio, use ZeroMQ to implement the client-server model necessary to remote control the HR-OS1 robot over WiFi from a Windows PC.

There are many opportunities for improvement, in particular I'd like to extend the rtclient/rtserver applications to allow commanding the robot to turn and reverse walk. I'd also like to investigate robot vision and speech synthesis and recognition. These additional capabilities would make HR-OS1 even more suitable for the robot theatre scenario.

References

The references are inlined throughout the paper.

Contents