Eclipse Configuration

This memo shows how to configure Eclipse Luna for the example applications. The IzoT ShortStack repository does not require that you use Eclipse, but we recommend it.

You may be able to use Eclipse and a C/C++ compiler toolchain directly on your target hardware, for example when using a Raspberry Pi 2 board. For instructions to configure Eclipse for cross-compilation and cross-debugging with a Raspberry Pi, here's a comprehensive step-by-step discussion: http://www.gurucoding.com/en/raspberry_pi_eclipse/index.php

The following discusses aspects of Eclipse configuration and project set-up specific to the example applications included with the IzoT ShortStack repository. We recommend that you follow these procedures at least for one of the example applications in order to verify the correctness of your workbench configuration, and we recommend that you follow the same model for setting-up your own applications.

Repository Layout

The repository includes the following directory structure. Note that only directories relevant to this discussion are shown:

Directory Name			Example File Names	Note
api			ShortStackApi.c ShortStackApi.h ...	3
microserver				
	standard		SS430_FT6050_SYS20000kHz.xif SS430_FT6050_SYS20000kHz.sym ...	
example				
	rpi			1
		driver	ldvTypes.h rpi.c ...	4
		simple	rpi-simple.c	5
			ShortStackDev.c ShortStackDev.h	

			main.xif	
		example#2		
		io		2

Note:

Numbers in the *Notes* column are referenced in the following text.

Eclipse Workspace

We recommend that you create an Eclipse workspace in the same local directory in which you cloned the ShortStack git repository. The remainder of this memo assumes that your Eclipse workspace matches your local repository location.

Eclipse Projects

We recommend that you create one Eclipse project within the workspace for each of your applications. We recommend that you create at least one project for one of the supplied example applications before you add your own.

We recommend calling the main C source file, the file which contains your project's *main()* function, after the project name. The rpi-simple.c file within the rpi-simple project, for example, is this project's main source file. (5)

This convention is not required, but simplifies several integration points between your work, the default Eclipse configuration and the IzoT Interface Interpreter.

Example Application

Open Eclipse with your chosen workspace, then add a new C project (File / New / C Project …). Call it *rpi-simple* and set its location to *example/rpi/simple.*

Because the link layer driver and API code are located in folders outside the Eclipse project folder, you must add those to your project:

Adding The *api* Folder

Select *File / New Folder*, then click the *Advanced* button and select *Link to alternate location (Linked Folder).* Click *Variables*, select *WORKSPACE_LOC*, click *Extend* and select the *api* source folder within your workspace. Click *OK* and *Finish*. (3)

Adding The *driver* Folder

Adding the *driver* folder is very similar to adding the *api* folder, but we recommend that you take a moment to define a new Eclipse variable to reference the *example/rpi* (1) folder at this time. This variable will be used to reference the *driver* folder, and can be used to reference other source folders shared across the examples for Raspberry Pi. For example, more complex example applications share an *io* folder for routines to handle physical application input and output. (2)

Select *File / New Folder*, then click the *Advanced* button and select *Link to alternate location (Linked Folder).* Click *Variables,* then click *New*.

Enter name *RPI_LOC*, then click *Variable,* select *WORKSPACE_LOC*, click *Extend* and select the *example/rpi* directory. Click *OK* to define this variable.

Back in the *Select Path Variable* dialog select the *RPI_LOC* variable, click *Extend* and select the *driver* folder. Click *Finish.* (4)

Other Project Settings

To complete the Eclipse configuration, follow these steps:

Locate the *rpi-simple* project in the Eclipse Project Explorer, right-click it and select *Properties*.

Under *C/C++ Build / Settings*, select the *Tool Settings* tab.

Verify the compiler or cross compiler prefix and path settings.

Select *All Configurations*, then select *Dialect* and chose *ISO C99 (-std=c99).*

For all configurations, add these symbols to the Symbols:

```
ARM_NONE_EABI_GCC
_XOPEN_SOURCE=600
```

If you wish to use the default implementation of a simple driver for persistent data, also add the following symbol for all configurations:

```
LON_NVD_FILEIO=${ProjName}.nvd
```

For the Debug Configuration only, also define this symbol:

```
_DEBUG
```

For all configurations, add the following include paths under *Includes*:

```
${workspace_loc:/${ProjName}}
${workspace_loc:/${ProjName}/api}
${workspace_loc:/${ProjName}/driver}
```

For all configurations, edit the *Other flags* under *Miscellaneous* and add *-pthread.*

For all configurations, add the *pthread* library to *Linker / Libraries.*

Preparing IzoT Interface Interpreter

Last not least, you should integrate IzoT Interface Interpreter with your build steps. For building from Eclipse, the easiest option is to specify the IzoT Interface Interpreter as a pre-build step:

In the project settings, select *C/C++ Build / Settings*, then select the *Build Steps* tab. For all configurations enter the following command:

```
iii "${ProjDirPath}/${ProjName}.c"
```

The IzoT Interface Interpreter Windows Installer installs *iii.exe* for Windows into the *bin* directory within your LonWorks folder, and adds this folder to your search path.

Your application's main source file (*rpi-simple.c* in this example) includes a reference to the IzoT ShortStack Micro Server for use with this application. You will need to update this directive if you wish to use a different Micro Server, or if you deviate from the suggested project layout.

The example applications included with the IzoT ShortStack repository select the Micro Server using this directive:

```
//@IzoT Option server("../../../microserver/standard/SS430_FT6050_SYS20000kHz")
```

See the product documentation for more about IzoT Interface Interpreter's directives in general and *option server* in particular, which supports project relative paths (as shown) and methods of locating a Micro Server along a search path specification.

Now build your application, debug and test either using local operation, or using a cross-debugging configuration.

Your Own Application

We recommend that you configure your own application, to the extent possible, within the same layout as used with the example applications. This enables you to benefit from fixes or improvements to the ShortStack product files as they become available (if you chose to integrate those with *your* GIT branch), and it allows you to push improvements you might have made onto the shared repository, thereby helping others.

Note that this does not require that you make your application public. To keep your application private, simply do not add it to your local git repository, or do not push it to a remote server.

Not all applications are suitable for joining the IzoT ShortStack repository, even a local repository. For example, an existing application may have other requirements regarding the folder layout, or may need to integrate with different tools or another revision control system, or you may simply wish to keep your code and intellectual property outside the IzoT ShortStack.

For those applications, we recommend that you integrate sources from the IzoT ShortStack repository using the Eclipse *linked folder* method discussed under *Example Application* above. This method allows managing your application project and your local branch of the IzoT ShortStack git project separately, but maintains an active link between the two projects such that you do not need to create and manage multiple copies of ShortStack API sources, for example.