LIQUID ROBOTICS
A Boeing Company

Wave Glider® Integrated Sensor User Guide

Developing for the SMC2 SDK

Revision 02
Item Number: 09161

Trademarks

Liquid Robotics®, Wave Glider®, Wave Glider Management System, WGMS, and SHARC® are worldwide trademarks of Liquid Robotics. Other products and company names mentioned herein may be the trademarks of their respective owners.

Patents

The Wave Glider mechanism is patented under US Patent Number 7,371,136 and other patents pending. All aspects of the Wave Glider product have been developed at private expense by Liquid Robotics.

Warranty

The information contained herein is subject to change without notice. The only warranties for Liquid Robotics, Inc. products and services are set forth in Liquid Robotics Terms and Conditions Articles 11.1 Limited Hardware Warranties, 11.2. Limited Service Warranties, and 11.3 Limitations. Nothing herein should be construed as constituting any additional warranties or representations. Liquid Robotics, Inc. shall not be liable for technical or editorial errors or omissions contained herein.

Contact

LIQUID ROBOTICS
A Boeing Company

Liquid Robotics

1329 Moffett Park Drive

Sunnyvale, CA 94089

Phone: +1 408 636 4200

Customer Support: +1 888-574-4574

Fax: +1 408 747 1923

Revision History

Document Part Number	Revision	SMC Software Version	Changes
09161	1.0.0	2.6.1	Initial Beta SDK Version.
09161	1.0.1	2.6.1	Clarify Oracle JRE Version.
09161	1.0.2	2.6.1	Include instructions for creating a SD card and setting up a SMC installer.
09161	1.0.3	2.8.0	Updated SMC software version.
09161	02	2.8.0	Fixed typos.

Contents

1 INTRODUCTION

This guide explains the resources available on the SMC2 Software Development Kit, `Ubuntu-18.04-LRI-SMC-2.8.0-TX1-09158-03A` VirtualBox virtual machine, designed for cross compiling the Liquid Robotics Sensor Management Computer software to run on the Nvidia Tegra X1 based payloads.

Note

Do not use the VirtualBox OSE distribution since it does not support USB devices well, and external card interfaces and ports which are useful with SMC development.

`SMC2` is used to distinguish this computer from the `SMC1`, which has a Cortex A8 based omap3/dm3730. Please see the `Ubuntu-14.04-LRI-SMC-2.8.0-TX1-07782-14A` VirtualBox virtual machine and the Developing for the SMC (Liquid Robotics P/N 04340-03) to compile for that target.

The Sensor Management Computer (SMC) is a Linux-based embedded ARM computer tailored to act as a Wave Glider payload and to implement algorithms and interfaces to sensors and other devices. This computer includes:

- Nvidia Tegra X1 with a quad core arm A57 and a Maxwell 256 core GPU.
- Two (2) general-purpose serial ports (J8 and J9) and an optional USB to serial with four (4) ports.
- Dedicated Ethernet communication with the Wave Glider Command and Control Unit (CCU).
- System GPS over the ethernet from the CCU.
- Serial debug port (J5).
- MicroSD interface.
- mSata SSD support.
- Two (2) USB-A ports, each supporting 2.0A and 10W.

Liquid Robotics recommends developing on LR's virtual machine and supports developers in the SDK. The ARM Cortex and Nvidia Maxwell processors implement a variety of features that improve power efficiency when compared with other processors capable of supporting Linux.

The Nvidia Tegra X1 supports a microSD card, currently verified with 64GB but should support any standard microSD card, and a mSata interface, which increases the storage capacity up to at least 1TB. The previous SMC1 based payloads only supported a 16GB microSD card, with a limit of 12GB of storage.

2 LIST OF REQUIRED FILES FROM OPEN OCEANS

The SMC2 SDK and related files can all be found on the Open Oceans portal. All files can be found under Training -> Software Development Kit.

1. Virtual Machine image:
   `Ubuntu-18.04-LRI-SMC-2.8.0-TX1-09158-03A`
2. User Guide:
   `09161-01 Developing for the SMC2 SDK.pdf`
3. Precompiled boost libraries and source headers:
   `boost_1_54_0.TX1.tar.bz2`
4. Precompiled omniORB libraries, binaries and source headers:
   `omniORB.TegraX1.tar.bz2`
5. SMC shareable source folder:
   `smc-2.8.0-shareable-tx1-source.tar.bz2`
6. Prebuilt tar-1.26 required when building the SMC installers:
   `tar-1.26-prebuilt.tar.bz2`
7. SD card golden master image:
   `smc-2.8.0-TX1-golden-master.img.bzip2`
   Also included in the Virtual Machine at:
   `/home/SMCDev/SMC/runtime/smc-2.8.0-TX1-golden-master.img.bzip2`

3 SETTING UP THE DEVELOPMENT ENVIRONMENT

3.1 Operating System and Virtualization

The SMC software runs on an ARM-based Linux environment maintained by Liquid Robotics. We currently recommend using the Eclipse editor and the makepp build system documented below. All users are recommended to use Oracle VirtualBox or VMWare. All examples will be using VirtualBox (tested up to version 6.1.2).

Note

Do not use the VirtualBox OSE distribution since it does not support USB devices well, and external card interfaces and ports which are useful with SMC development.

Oracle VirtualBox and the VirtualBox Extension Pack can be downloaded from this website.

3.2 Recommended Hardware/Software Environment

- A PC or MAC with at least 8GB of RAM and 100GB of free disk space.
- Windows 7 or later, OS X 10 or higher, or a native Ubuntu installation.
- Oracle VirtualBox and Extension Pack.
- USB-based microSD card reader/writer.

3.3 Configuring Ubuntu VM (Using Pre-Existing Image)

After installing VirtualBox,

1. From the VirtualBox `File` menu, select `Import Appliance` and select the appliance file (.ova).
2. On the resulting window, under the `MAC Address Policy` dropdown select `Generate new MAC addresses for all network adapters`. Click `Import`.
3. When the import option completes, start the virtual machine and log in with username `dev` and password `1329moffet`.
4. Be sure to update the packages either through Update Manager or by running the following commands in a terminal session.
   a. `sudo apt-get update`
   b. `sudo apt-get upgrade`
5. Update the Guest Additions of VirtualBox.
   a. Devices -> Insert Guest Additions CD image..
   b. In the resulting window select ‘Run’.
   c. When prompted enter the password `1329moffet`.
   d. A terminal will open and go through the steps of uninstalling the version shipped with the ova and installing the version that matches the version of VirtualBox you have.
   e. When the installation is complete you will see the prompt `Press Return to close this window…`

4 WORKING WITH THE DEVELOPMENT ENVIRONMENT

4.1 Software Development Directories

The following directories have special meanings for the build environment:

- Home directory for the SMC build environment.
  `/home/SMCDev`
- Home directory for the omniORB development files. It includes a link to the Tegra X1 build of the omniORB libraries and binaries.
  `/home/SMCDev/omiORB`
- Home directory for the source code. Each project shown has a sub-directory that contains source code.
  `/home/SMCDev/SMAC`
- Symbolic link to the source code.
  `/home/SMCDev/SMC`
- Home directory for Eclipse projects. Eclipse keeps some extra information in these directories that should not be checked in. The `.cproject` and `.project` files should be checked in.
  `/home/SMCDev/SMC/eclipseWorkspaces`
- Home directory for the makepp command-line build software.
  `/home/SMCDev/SMC/mpp2`
  The top-level directory has instructions and scripts that are used commonly throughout the build. There are sub-directories for each project, and sub-directories under those for each build type (`TegraX1Debug`, `TegraX1Release`). Each project directory has a `rules` file that explains how to build the object files (.o) for the source. The `common` directory contains the rules to build all the executables and libraries. The full set of executables from the build is accumulated under `common/<build-type>`.
- This directory contains scripts and configurations that gather up executable files and make them available for installation.
  `/home/SMCDev/SMC/runtime`

As part of the installation process, some files are installed under `/usr/local`, including:

- TegraX1 build of Boost library files (the version that matches what is installed on the SMC2).
- OmniORB build of Corba library files (the version that matches what is installed on the SMC2).

4.2 Building the Software Using Eclipse

The software can be compiled using one of two recommended build profiles:

- **`Debug`**: un-optimized, includes symbols, best for running with the debugger on Ubuntu (Boost 1.54.0 and omniORB 4.1.7).
- **`Release`**: optimize, best for execution time and size on Ubuntu (Boost 1.54.0 and omniORB 4.1.7).
- **`TegraX1Debug`**: un-optimized, includes symbols, best for running with the debugger on the target (Boost 1.54.0 and omniORB 4.1.7, armhf cross compiler).
- **`TegraX1Release`**: optimized, best for execution time and size on the target (Boost 1.54.0 and omniORB 4.1.7, armhf cross compiler).

An additional 6 build configurations (ZoomDebug, ZoomRelease, zzDebug6.0.0, zzRelease6.0.0, zzZoomDebug6.0.0, zzZoomRelease6.0.0) but they cannot be compiled on the SMC2 SDK. Please see the

`Ubuntu-14.04-LRI-SMC-2.8.0-TX1-07782-14A` VirtualBox virtual machine and the Developing for the SMC (Liquid Robotics P/N 04340-03) to compile for these targets.

To compile, select all projects on the left pane (Project Explorer) and right click on a project, select Build Configurations -> Set Active from the popup menu and choose one of the build profiles. Then select a single project, right click on it, and select Build Project from the popup menu. Eclipse has been configured to use the makepp build system, so it infers build rules from the source code as well as from its configuration files.

Each Eclipse project has been configured to use parallel builds. Click on a project, go to Project -> Preferences, click on C/C++ Build, click on Behavior. Under C/C++ Build Behavior, there is a radio button `Enable Parallel Builds` and a secondary option `Use Optimal Jobs (4)`, where `(4)` would be the number of cores available to the virtual machine.

Then under C/C++ Build, click on Builder Settings. Under Builder Settings, the `Build Command` field has been updated to `makepp -j4`. `-j4` is a command line option to enable 4 cores and this has been automatically updated by Eclipse.

4.3 Creating a Project

There are two ways a user can create a project in Eclipse. For more details refer to `/home/SMCDev/SMC/eclipseWorkspaces/projectSetupNotes/Eclipse.txt`.

4.3.1 Create an Empty Project

- In Eclipse go to File -> New C++ Project.
- Set your project name, denoted as MyProject.
- Project type: Makefile project -> Empty Project
- Toolchains: Linux GCC
- Click Finish.

In the project view, add a folder `src` to the project that is a link to the source directory, typically under `/home/SMCDev/SMC/MyProject`. This folder will be the location of the source files (to edit, compile, debug, etc.).

- Right click on the project -> New -> Folder.
- Enter `src` for the folder name.
- Click `Advanced>>` -> Click `link to alternate location (linked folder).`
- Enter `/home/SMCDev/SMC/MyProject`, where MyProject is the name of the project being created, for the folder path.

4.3.2 Copy an Existing Project to Create a New Project

This process has the benefit of copying the build types, e.g. Debug and TegraX1Release, for building from the Eclipse menu.

- Copy an existing project, e.g. `DemoSensor`, rename it, change where the src folder is pointing to.

In the project view, add a folder called src to the project that is a link to the source directory, typically under /home/SMCDev/SMC/MyProject, this folder will be the location of the source files (to edit, compile, debug, etc.).

- Right click on the project -> New -> Folder.
- Enter `src` for the folder name.
- Click `Advanced>>` -> Click `link to alternate location (linked folder)`.
- Enter `/home/SMCDev/SMC/MyProject`, where MyProject is the name of the project being created, for the folder path.

4.4 Building the Software Using makepp (Command Line)

makepp is a kind of make utility that performs source file scanning to understand dependencies and does not require as much specification to perform a build. Note that makepp drops hidden directories named as `.makepp` in source code subdirectories where it stores the results of its dependency scans.

4.4.1 Maintaining the makepp Build

The `rules` file in each project directory (i.e. `mpp2/<project>`) contains a list of modules to be included. The advantage of this is a C or C++ files in the source directory (i.e. `/home/SMCDev/SMC/<project>`) can exist but are not included in the build. This also means that the list of files must be maintained. When new module files are added to the project or when modules are removed, the list should be updated. For example, the highlighted list should be updated (`mpp2/GPSListenerLib/rules`):

```
SRCDIR := $(SRCPARENT)/GPSListenerLib
DEP0 := SMACFramework EventLib
DEP := SMACIdl $(DEP0)
CXXFLAGS := -I$(SRCDIR) -I$(SRCPARENT)/$(DEP0) $(IDLCFLAGS) -
I$(CORBA_INSTALL)/include -I$(CORBA_INSTALL)/include/COS $(CXXFLAGSSTD)
-c

MODULES := \
GPSEventSource \
GPSListenerPersistentSettings \
GPSListenerProgram \
GPSListenerSettings \
GPSListenerTestProgram \
NMEAParser

$(phony all): $(MODULES).o

$(phony clean):
      rm -rf $(MODULES).o

$ (phony cleanall): $(TOP)/$(DEP)/$(BUILDDIR)/cleanall clean

%.o : $(SRCDIR)/%.cpp
    $(CXX) $(CXXFLAGS) $(input) -o $(output)
```

Presently, the same list appears in the section related to the artifact under `mpp2/common/rules` and must be maintained.

```
############################################################
################## GPSListenerLib Library ##################
############################################################
```

```
GPSLISTENERLIBMODULES := \
GPSEventSource \
GPSListenerPersistentSettings \
GPSListenerProgram \
GPSListenerSettings \
GPSListenerTestProgram\
NMEAParser

GPSLISTENERLIBDEP := SMACIdl SMACFramework EventLib

libGPSListenerLib.so:
$(TOP)/GPSListenerLib/$(BUILDDIR)/$(GPSLISTENERLIBMODULES).o
lib$(GPSLISTENERLIBDEP).so
      $(LINKER) $(LFLAGSLIB) -o $(output)
$(TOP)/GPSListenerLib/$(BUILDDIR)/$(GPSLISTENERLIBMODULES).o -L. -
l$(GPSLISTENERLIBDEP) $(BOOSTLFLAGS)
```

4.4.2 Building Everything

```
cd /home/SMCDev/SMC/mpp2
makepp common/<build-type>/all
```

For example, `makepp common/TegraX1Release/all`.

4.4.3 Cleaning Everything (Removes Top Level Build Artifacts)

```
cd /home/SMCDev/SMC/mpp2
makepp common/<build-type>/clean
```

For example, `makepp common/TegraX1Debug/clean`.

4.4.4 Cleaning Everything Recursively (Removes All Build Artifacts)

```
cd /home/SMCDev/SMC/mpp2
makepp common/<build-type>/cleanall
```

For example, `makepp common/TegraX1Release/cleanall`.

4.4.5 Force Rebuild of Everything

```
cd /home/SMCDev/SMC/mpp2
makepp common/<build-type>/cleanall
```

For example, `makepp common/TegraX1Release/cleanall`.

```
cd /home/SMCDev/SMC/mpp2
makepp common/<build-type>/all
```

For example, `makepp common/TegraX1Release/all`.

4.4.6 Build a Single Project

```
cd /home/SMCDev/SMC/mpp2
makepp common/<build-type>/artifact
```

For example, `makepp common/TegraX1Debug/libSMACFramework.so`.

4.4.7 Force Rebuild a Single Project

```
cd /home/SMCDev/SMC/mpp2
rm -f common/<build-type>/artifact
```

For example, `rm -f common/TegraX1Debug/libSMACFramework.so`.

```
makepp <project>/<artifact>/clean
```

For example, `makepp SMACFramework/TegraX1Debug/clean`.

```
makepp common/<build-type>/artifact
```

For example, `makepp common/TegraX1Debug/libSMACFramework.so`.

4.5 Deploying and Integrating a Project to an Existing SMC2

Once you've built your project with the TegraX1Debug and TegraX1Release config, you can copy the release files via sftp to `/home/smc/bin` on your SMC2 payload and they will be available to other smc services.

To integrate your project into the SMC framework on the target SMC2, you also need the following:

- Create a directory under /home/smc/roots to hold your startup scripts.
  `mkdir /home/smc/roots/200_MyProject`
- Create a startup script that will be executed by the SMC framework on startup or reboot.
  `echo "/home/smc/bin/MyProject 2>&1 | Logger --work-dir=$HOME/log --prefix="MyProject" --extension=log" >> /home/smc/roots/200_MyProject/startup`
- Make the startup script executable.
  `chmod +x /home/smc/roots/200_MyProject/startup`
- If you don't want your project to start on reboot of the SMC2, create a file `.disabled` in the project directory.
  `touch /home/smc/roots/200_MyProject/.disabled`
- Shut down the SMC framework.
  `killsmc`
- Restart the SMC framework.
  `/home/smc/startup`
- The SMC framework will go through each directory under `/home/smc/roots` and in turn, if there is no `.disabled` file, will attempt to execute the script called `startup` there.
- Further integration into the SMC framework, for example ship-to-shore messaging, is beyond the scope of this document.

4.6 Building a SMC Configuration

After compiling the TegraX1Release or TegraX1Debug, there is a folder named `runtime` where one may build a SMC installer. The SMC installer may be used to install the SMC software onto a LRI Linux microSD card image.

Note

SD card image is included at `/home/SMCDev/SMC/runtime/smc-2.8.0-TX1-golden-master.img.bzip2`.

This is the same process used for generating an image for the existing `SMC1` hardware, however some details in the root file system "template" are different.

1. Populate the `runtime/rfs` directory with program files from the `runtime/template` directory and with the SMC binaries and libraries (in this case TegraX1Release).
    a. Building your own:
    ```
    cd /home/SMCDev/SMC/runtime
    ./populate_smc_mpp TegraX1Release
    ```
    b. Using the prebuilt binaries supplied with the SDK.
    ```
    cd /home/SMCDev/SMC/runtime
    ./populate_smc_mpp TegraX1Release/prebuilt
    ```
2. Create `smc.tar.bzip2` using the content of the `runtime/rfs` directory.
    ```
    ./maketar
    ```
3. Rename the tarball according to the release number. The SMC release version number for this guide is 2.8.0.
    ```
    mv smc.tar.bzip2 smc-2.8.0A.tar.bzip2
    ```
4. Split the full build into the core set of SMC libraries and binaries.
    ```
    ./splitreleases smc-2.8.0A.tar.bzip2 ../../tar-1.26/bin/tar
    ```
5. This generates a tar file containing only the core SMC components.
    ```
    smc-2.8.0A-core.tar.bzip2
    ```
6. `runtime/config` is where different SMC configurations can be stored in separate directories. Each directory should contain a runme script that references the SMC build artifacts and creates a resulting installation file for use on the target.
    a. See `config/0000000000-2.8.0/runmefirst` for an example.
    b. Each config captures a specified payload wiring diagram.
    c. The `runmefirst` file modifies the SMC configuration files as stored in the provided smc tar file.
    d. `runmefirst` has a variable `INFILE` initialized in the script pointing to a specific SMC version, e.g. smc-2.8.0A-core.tar.bzip2.
    e. `runmefirst` has a variable `PARTNUM` which specifies the part number, so we can check later when the SMC is running, e.g. 0000000000-2.8.0.
    f. The config files it modifies are usually `<sensor_name>-config.xml` stored under sensor folders in `/home/smc/roots` on the target, which are taken from `runtime/template/home/smc/roots` and copied into the `rfs` folder during `populate_smc_mpp` script. You may directly modify the `template` and start over at step 1 as well.
    g. The `runmefirst` files extracts some files from the existing `smc-2.8.0A-core` tar file, modifies them, and creates a new tar file with the modified configuration.
7. Invoke the buildconfig script for an individual directory in runtime/config.
    ```
    cd /home/SMCDev/SMC/runtime
    ./buildconfig 0000000000-2.8.0
    ```
8. This results in two outputs, each with an associated `.sha512` file.
    a. This is the tarball with the SMC rfs and desired wiring configuration, however untarring this onto an existing SMC SD card will not result in desired configuration of networks and it will not communicate with Regulus anymore.
    ```
    0000000000-2.8.0.tar.bzip2
    ```
    b. This is the tarball with a self-extracting header that can be executed on the SMC to install and configure for use on LRI payloads with the stand alone `SMC2`.
    ```
    0000000000-2.8.0.run
    ```

4.6.1 Notes on SMC SD Card

The SD card image comes from LR with a LR controlled Linux image and a SMC configuration preinstalled. The SD card image is based on RegOS version 1.4.3 and the SD card image golden master included in the virtual machine at `/home/SMCDev/SMC/runtime/smc-2.8.0-TX1-golden-master.img.bzip2`.

By convention, the SMC install files for a release, such as 0000000000.run and 0000000000.run.sha512 are found at `/root/` on a target SD card.

The golden master has not had a SMC software package installed onto it and thus will not communicate or register itself with Regulus. The idea is the golden master is a fresh SD card image where one can install a specific SMC configuration using a `.run` file.

The SD card image comes with Oracle Java 8 runtime environment:

```
smc@smc-tegrax1:~$ java -version
openjdk version "1.8.0_162"
OpenJDK Runtime Environment (Zulu Embedded 8.27.0.91-linux-aarch32hf)
(build 1.8.0_162-b91)
OpenJDK Server VM (Zulu Embedded 8.27.0.91-linux-aarch32hf) (build
25.162-b91, mixed mode)
```

4.7 Create a SD Card from the SMC Golden Master

4.7.1 Using a Physical SD Card Duplicator

It is recommended to create a SD card from the golden master image and then use a SD card duplicator because copying from a disk image can be a very slow process.

The easiest way for most customers to back up and restore their microSD cards is using a microSD card duplicator. For example, the U-Reach SD Duplicator with MicroSD Adapters is a standalone SD/microSD card duplicator that can be used with a plug-in power supply or with batteries.

Customers using a duplicator to store backups on physical microSD cards may have the advantage of simplicity, but flash cards are not meant for long-term storage (~5 years) and can be damaged by handling. Also, you must have physical access to the backup card to use it which may be problematic. Thus, it is recommended to back up mission data by plugging the SD card into a computer and storing the data somewhere with redundancies.

To save time while copying, duplicators may copy only some sections of the microSD card, leaving traces of the previous content in place. Be sure to use the secure erase function of your duplicator to prepare the target media that has been used before if the microSD card will change hands.

4.7.2 Using the SMC Golden Master

A fresh SD card can be created from the golden master disk image using the below process.

- Insert the SD card into a SD card reader (recommended to use a USB SD card reader). Use `parted` to locate the filename of the SD card. In this case, as highlighted below, the USB 2.0 Card Reader has found a SD card at `/dev/sde`.

```
dev@SMCSDK:~/$ sudo parted -ls
[sudo] password for dev:
Model: ATA VBOX HARDDISK (scsi)
```

```
Disk /dev/sda: 107GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number Start   End    Size   Type     File system    Flags
1      1049kb  104GB  104GB  primary  ext4           boot
2      104GB   107GB  3218MB extended
5      104GB   107GB  3218MB logical  linux-swap(v1)

Model: USB2.0 CardReader SD (scsi)
Disk /dev/sde: 63.9GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number Start  End    Size   Type    File system Flags
1      1049kb 12.0GB 12.0GB primary ext4
2      12.0GB 63.9GB 51.9GB primary btrfs
```

- Copy the golden master image onto the SD card at `/dev/sde`.

```
time ( bzcat smc-2.8.0-TX1-golden-master.img.bzip2 | sudo dd
of=/dev/sde )
124735488+0 records in
124735488+0 records out
63864569856 bytes (64 GB) copied, 19722.6 s, 3.2 MB/s

real 328m43.612s
user 10m24.784s
sys  13m41.596s
```

- This may take a long time, which is why the physical SD card duplicator is strongly recommended and use this process as a backup. It might be this process was slow because it was run inside a Virtual Machine.

4.8 Installing SMC Software onto a Fresh SD Card

- Plug the SMC2 debug cable into the payload and plug the serial port into your computer to gain access to the TX1's console (generally use a USB <-> serial cable) and open up the console using Putty or TeraTerm on Windows or using a command line tool such as screen or minicom on Linux or OS X and a baud rate of 115200.
- Plug in the payload to a CCU expansion port, e.g. G1, and power on the payload using the SV3 command `Turn On Integrator Payload`. For a payload plugged into G1 on WGMS this would be under SV3 Commands -> G1Integrator -> Turn On Integrator Payload.
- Watch the console and wait until it's done booting, log in as `root` user and check that the SMC installer files are there.

```
* Stopping System V runlevel compatibility                          [ OK ]
* Starting Get a getty on ttyS0                                     [ OK ]
* Starting LRI first boot upstart service                           [ OK ]

RegOS 1.4.3 Unconfigured-SVEN ttyS0
Unconfigured-SVEN login: root
Password:
root@Unconfigured-SVEN:~# ls -l
total 25728
-rwxr-xr-x 1 root root     1603 Nov 26 18:09 AptGetSmcLibs
-rw-rw-r-- 1 root root      490 Oct 29  2018 pi.py
-rwx------ 1 root root 52705935 Nov 26 18:54 0000000000.run
-rw-r--r-- 1 root root      151 Nov 26 18:54 0000000000.run.sha512
```

```
root@Unconfigured-SVEN:~#
```

- We can see there is one installer file `0000000000`.
  - `0000000000` is a SMC software that runs SMC 2.8.0 on a Tegra X1 processor. It runs the basic core components of the SMC software.
- We should check the correctness of the files, in case the SD card has been corrupted somehow.

```
root@Unconfigured-SVEN:~# shasum -c *.sha512
0000000000.run: OK
root@Unconfigured-SVEN:~#
```

- Stop the docker daemon.

```
root@Uncofigured-SVEN:~# /etc/init.d/docker stop
```

- Unmount /mnt/XDATA.

```
root@Unconfigured-SVEN:~# umount /mnt/XDATA
```

- Install the desired SMC configuration.

```
root@Unconfigured-SVEN:~# ./0000000000.run
Good SVEN version 1.4.3  >= 1.3.2 from /etc/lri-release. Proceeding with install.
OS is compatible: TegraX1 Ubuntu version 3.10.67-LRI-4+
SD Card Partition devices are present

Install SMC software P/N 0000000000 (y/n) y

mSata SSD Present. Putting XDATA there.
SSD is mounted to /mnt/XDATA
/mnt/ssd -> /mnt/XDATA/ssd
Mounted XDATA
Setting default password for root
Adding user smc
Setting default password for smc
Adding user smc0
Setting default password for smc0
Setup new users
find: `/home/smc': No such file or directory

Discarding unused backup directory

Checked not empty XDATA

Expanding the embedded tarball ...

etc/
etc/hostname
etc/cron.deny
etc/ppp/

…
```

- Once it has untarred all the files onto the Linux filesystem partition, it will open a text menu to set up some install options. Since all SMC2 payloads are currently expansion port payloads, we should select the correct menu items like so
  - type `3` -> to use `REST on Expansion Port` to communicate with Regulus over the network cable.
  - type `9` -> to set SMC static IP address -> Leave blank to use DHCP.

```
No files were backed up from the last SMC install

SMC Software Configuration

 1) Auto Ethernet on startup:          y
 2) Default board ID (typically 32):   32
 3) Host communication:                Payload Protocol on Sensor Port
 4) Host IP address:                   (Determined by script)
    RUDICS enabled:                    n
 5)   RUDICS Host:
```

```
 6)   RUDICS key:
 7)   RUDICS login:
 8)   RUDICS phone:                    (default)
 9) SMC static IP:                     192.168.0.100
    Modules enabled:                   omniNames EventManager PayloadManager
PayloadInterface GPSListener

  select # s=save q=quit >> 3

SMC Software Configuration

 1) Auto Ethernet on startup:          y
 2) Default board ID (typically 32):   32
 3) Host communication:                REST on Expansion Port (SV3 Only)
 4) Host IP address:                   (Determined by script)
    RUDICS enabled:                    n
 5)   RUDICS Host:
 6)   RUDICS key:
 7)   RUDICS login:
 8)   RUDICS phone:                    (default)
 9) SMC static IP:                     192.168.0.100
    Modules enabled:                   omniNames EventManager PayloadManager
PayloadInterface GPSListener

  select # s=save q=quit >> 9

SMC static IP address (blank=use DHCP):

 1) Auto Ethernet on startup:          y
 2) Default board ID (typically 32):   32
 3) Host communication:                REST on Expansion Port (SV3 Only)
 4) Host IP address:                   (Determined by script)
    RUDICS enabled:                    n
 5)   RUDICS Host:
 6)   RUDICS key:
 7)   RUDICS login:
 8)   RUDICS phone:                    (default)
 9) SMC static IP:                     (DHCP)
    Modules enabled:                   omniNames EventManager PayloadManager
PayloadInterface GPSListener

  select # s=save q=quit >> s

Updating PayloadManager-config.xml
- default board ID = 32

Updating PayloadInterface-config.xml
- setting protocol to REST on Expansion Port (SV3 Only)
- setting host IP address to


Do you want to enable autostart of the software (you are finished setting up)?
(y/n) y
If you're finished setting up, do you want to restart the SMC? (y/n) y

Broadcast message from root@Unconfigured-SVEN
        (/dev/ttyS0) at 20:26 ...

The system is going down for reboot NOW!
```

- After rebooting is complete, one may log in as user `smc` and password `1329moffet`.

```
RegOS 1.4.3 smc-tegrax1 ttyS0
smc-tegrax1 login: smc
Password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.10.96-LRI-13+ aarch64)

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
```

```
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

smc@smc-tegrax1:~$ ls
autostart  drivers  licenses  man             omniORB.cfg  sc       XDATA
bin        lib      log       MountDisks.log  roots        startup
```

- We can see these programs are running by the log files (or by looking at the programs running through a command such as `ps -eaf`).

```
smc@smc-tegrax1:~$ ls -l log/
total 24044
-rw------- 1 smc smc 3348404 Mar 26 20:30 EventManager.log
-rw------- 1 smc smc  856034 Mar 26 20:29 GPSListener.log
-rw------- 1 smc smc  311629 Mar 26 20:30 PayloadInterface.log
-rw------- 1 smc smc   80783 Mar 26 20:27 PayloadManager.log
-rw------- 1 smc smc    1778 Mar 26 20:27 ReRun.log
```

- At this point, the SMC software will register itself with Regulus and the SV3 commands will be populated on WGMS.
    - If the SV3 commands don't arrive, please power cycle the integrator payload using (payload plugged into G3 for example) SV3 Commands -> Devices -> G3Integrator ->
        - Turn Off Integrator Payload
        - Unplug Integrator Payload
        - Turn On Integrator Payload
        - It should automatically report SV3 commands to shore under SV3 Commands -> Enter script command. One should see, for example G3_PayloadInterface.
    - Refer to SMC Integrator documentation for details on finding out if the SMC software is up and running.

APPENDIX A: BUILDING A SDK

Recommended to use the Virtual Machine image, rather than generating your own. All steps for creating a SMC2 SDK virtual machine are listed below should the need arrive.

After installing VirtualBox,

1. Create a new Virtual Machine, selecting `Linux` as the type and `Ubuntu` as the version.
2. Allocate as much RAM as would be feasible. Ideally, you would have at least 4GB allocated to Linux (you can change this configuration later if desired). The lowest amount of RAM that has been tested is 3GB (3072MB) for use with the Eclipse indexer.
3. Choose the type of hard drive. The VirtualBox Disk Image (VDI) is recommended unless you need interoperability with another virtualization product.
4. Select fixed size (recommended if you have the free space) or dynamically allocated (if space is a concern).
5. Choose a disk size (at least 50GB is recommended, 100GB would be better for long term use).
6. It's useful to enable bi-directional clipboard access (under Settings -> General -> Advanced).
7. Start the virtual machine and select the `.iso` file downloaded from the Ubuntu website.
8. Follow the prompts to install Ubuntu. Download any updates during the installation, using the entire virtual disk for Ubuntu.
9. Enter your name, username and a strong password (the username `dev` and password `1329moffet` are used for the development image).
10. Wait for installation process to complete and reboot when prompted.
11. From `System Settings` and `User Accounts`, create `smc` and `smc0` users leaving the accounts disabled. These usernames are needed when creating packages for the target (i.e. the SMC).
12. When the Update Manager prompts (usually with hundreds of updates to be performed), open it and download updates. Restart Ubuntu when prompted.
13. At any point during the installation, you can pull down the VirtualBox Devices menu and click Install Guest Additions to address some issues with the size of the display. This will compile some modules and then require a restart. The display size problems will be addressed after restart. Note that later installations may undo this, and it will have to be repeated so it may be better to wait until most of the installations have been made.
14. Open a terminal session (Ctrl-Alt-T).
15. If desired, the password prompt for `sudo` (which allows a single command to be run as the root user) can be bypassed by running `sudo sudoedit /etc/sudoers` and adding the following at the end of the file:
    ```
    dev ALL = (root) NOPASSWD:ALL
    ```
16. Install aptitude by running `sudo apt-get install aptitude`.
17. If desired, restore classic gnome by running `sudo aptitude install gnome-session-fallback` then log out. Pull down the menu presented in the login bar and select `Gnome Classic` or `Gnome Classic (no effects)` then continue to login.
18. Run `sudo usermod -a -G dialout dev` to allow user `dev` to use USB serial ports whenever they are mounted.
19. If desired, install the Chromium web browser by running `sudo aptitude install chromium-browser`.

20. Install support programs by running `sudo aptitude install g++ meld kdiff3 lua5.1 screen makepp git subversion vlan u-boot-tools cpufrequtils curl i2c-tools speedometer gawk p7zip p7zip-full socat pbzip2 eclipse-cdt vim secure-delete`.
21. For systems running on bare metal or where the system must be contactable via ssh, install the ssh server (`sudo aptitude install openssh-server`).
22. For systems running on bare metal or for additional security, enable the firewall, but allow ssh access:
    ```
    sudo ufw enable
    sudo ufw allow ssh
    ```
23. For systems running in a virtual machine or for enhanced security, install the `security-delete` package using Ubuntu Software Manager. Whenever a virtual machine appliance is exported, it is advisable to clear the free space to reduce the size of the resulting appliance image file. The `security-delete` package also contains utilities that write patterns of random data over files when they delete to reduce the risk of data recovery by a third party.
24. By convention, SMC source code is loaded into directories under `/home/SMCDev/SMC`, and some other files are placed into directories under `/home/SMCDev`. Run the following to create these directories:
    ```
    cd /home
    sudo mkdir SMCDev
    sudo chown dev SMCDev
    sudo chgrp dev SMCDev
    ```
25. Installed the precompiled boost libraries SMCDev:
    ```
    mkdir -p /home/SMCDev/TegraX1
    tar -xf boost_1_54_0.TX1.tar.bz2 -C /home/SMCDev/TegraX1/
    ```
26. Install omniORB support files:
    ```
    mkdir -p /home/SMCDev/omniORB
    tar -xf omniORB.TegraX1.tar.bz2 -C /home/SMCDev/TegraX1/
    ln -s /home/SMCDev/TegraX1/omniORB/TegraX1 /home/SMCDev/omniORB/TegraX1
    ```
27. Install the **`ddd`** front end to gdb for visual debugging:
    ```
    sudo aptitude install ddd
    ```
28. Install the **`tofrodos`** package (todos, fromdos utilities) to adjust how line ends are encoded:
    ```
    sudo aptitude install tofrodos
    ```
29. Install the **`tmux`** window manager if desired:
    ```
    sudo aptitude install tmux
    ```
30. Install the **`zssh`** Z-Modem over SSH utility if desired:
    ```
    sudo aptitude install zssh
    ```
31. Install the source code under `/home/SMCDev/SMC`:
    ```
    tar -xf smc-2.8.0-shareable-tx1-source.tar.bz2 -C /home/SMCDev/SMC
    ln -s /home/SMCDev/SMC /home/SMCDev/SMAC
    ```
32. Install tar-1.26 under `/home/SMCDev`:
    ```
    tar -xf tar-1.26-prebuilt.tar.bz2 -C /home/SMCDev/
    ```
33. Install cross compilers:
    ```
    sudo apt-get install g++-4.8-aarch64-linux-gnu g++-4.8-arm-linux-gnueabihf
    ```
34. Need to be able to install armhf compilers to cross compile. Edit the apt-get source list:
    ```
    sudo vim /etc/apt/sources.list
    deb [arch=armhf] http://ports.ubuntu.com/ubuntu-ports/ bionic main universe
    ```

```
deb-src http://ports.ubuntu.com/ubuntu-ports/ bionic main
deb [arch=armhf] http://ports.ubuntu.com/ubuntu-ports/ bionic-updates main
deb-src http://ports.ubuntu.com/ubuntu-ports/ bionic-updates main
deb [arch=armhf] http://ports.ubuntu.com/ubuntu-ports/ bionic-security main
deb-src http://ports.ubuntu.com/ubuntu-ports/ bionic-security main
```

35. Install required armhf libraries:

```
sudo apt-get update
sudo aptitude install crossbuild-essential-armhf
sudo apt-get install libcurl4-openssl-dev:armhf liblua5.1-dev:armhf libnss3-dev:armhf
```

Note that these are installed and available here for the cross compiler:

```
/usr/lib/arm-linux-gnueabihf/libcurl.so.4.3.0
/usr/lib/arm-linux-gnueabihf/libcurl.a
/usr/lib/arm-linux-gnueabihf/libcurl.la
/usr/lib/arm-linux-gnueabihf/pkgconfig/libcurl.pc
```

36. Note that libcurl and libnss are NOT multiarch compatible. If one installs them for i386/amd64, then they will delete the armhf version:

```
sudo apt-get install libcurl4-openssl-dev
…
The following packages will be REMOVED:
comer-dev:armhf krb5-multidev:armhf libcurl4-openssl-dev:armhf
```

37. If using Eclipse, a few more steps are needed:
    a. Add the following build variables under Window -> Preferences -> C/C++ -> Build -> Build Variables:
        makepp_build_location = /home/SMCDev/SMC/mpp2
        debug_path = common/Debug
        release_path = common/Release
        tx1_release_path = common/TegraX1Release
        tx1_debug_path = common/TegraX1Debug
    b. Add the existing projects to Eclipse from File -> Import…, then selecting General -> Existing Projects Into Workspace, select the workspace directory (`/home/SMCDev/SMC/eclipseWorkspaces`) and then click Finish.

APPENDIX B: RUNNING SMC SAMPLE ON THE SDK

A user can run the sample SMC software on the SDK by following steps.

- Build the Debug config.

```
$ cd /home/SMCDev/SMC/mpp2
$ makepp common/Debug/all
```

- Start the background services from its own terminal.

```
$ cd /home/SMCDev/SMC/mpp2/common/Debug
$ ./run_multiple EventManager PayloadManager
Starting omniNames

Thu Apr 30 16:47:10 2020:

Starting omniNames for the first time.
Wrote initial log file.
Read log file successfully.
Root context is IOR:010000002b00000049444c3a6f6d672e6f72672f436f734e616d696e67
2f4e616d696e67436f6e746578744578743a312e3000000100000000000000088000000001010200
0a0000006c6f63616c686f7374007da90b0000004e616d6553657276696365000400000000000000
0008000000010000000054544101000000 1c00000001000000010001000100000001000105090 1
0100010000000901010003000000140000000010000000a00000031302e302e322e313500f90a03
545441080000007e63ab5e010022c3
Checkpointing Phase 1: Prepare.
Checkpointing Phase 2: Commit.
Checkpointing completed.
Starting EventManager
Starting PayloadManager
2020/04/30 16:47:12.433356 SMCSDK [8910] Initializing EventTransceiver
```

- Then start the program on its own terminal.

```
$ cd /home/SMCDev/SMC/NetBeans/smctools/SMCTools
$ java -classpath dist/SMCTools.jar LiquidRobotics.SMCSamples.SMCBasicSample \
  -ORBInitialHost localhost -ORBInitialPort 2809
Current directory is: /home/SMCDev/SMC/NetBeans/smctools/SMCTools
The default board ID is 33
```

- You will see the following on the infrastructure terminal.

```
2020/04/30 16:48:57.449018 SMCSDK [8903] EventManager::AddSubscriber2(
BasicSample = IOR:000000000000003b49444c3a4c6971756964526f626f746963732f534d41
432f4950696e6761626c65436f6d6d616e6461626c65576974684576656e74733a312e30000000
0000010000000000000082000102000000000a3132372e302e302e310081670000031afabcb00
00000020cd7e4640000000010000000000000001000000008526f6f74504f410000000008000000
0100000000140000000000000200000001000000200000000000010001000000020501000100001
0020000101090000000100010100000000260000000020002) -> 1 connections
2020/04/30 16:48:57.452849 SMCSDK [8903] EventManagerImpl::SetCustomFilter for
 BasicSample filter { enabled=1 aliasToEventName={ } nameList={
"CommandComplete" "GPSSample" "Shutdown" } typeAndDomainList={} }
2020/04/30 16:48:57.467078 SMCSDK [8910] Register payload={ BoardID=0xFF
TaskID=0x88 Description='Basic source sample for an SMC service in Java' IOR=
'IOR:000000000000003b49444c3a4c6971756964526f626f746963732f534d41432f4950696e
6761626c65436f6d6d616e6461626c65576974684576656e74733a312e300000000000010000000
0000000820001020000000a3132372e302e302e310081670000031afabcb0000000020cd
7e4640000000010000000000000001000000008526f6f74504f410000000008000000010000000
0140000000000000200000001000000200000000000010001000000020501000100010020000 1
0109000000010001010000000026000000020002' ReadableName='BasicSample'
DescriptiveJSON='{
2020/04/30 16:48:57.467142 SMCSDK [8910]    "ui" : [
```

```
2020/04/30 16:48:57.467169 SMCSDK [8910]       {
2020/04/30 16:48:57.467235 SMCSDK [8910]          "label" : "Shut Down Server",
2020/04/30 16:48:57.467282 SMCSDK [8910]          "method" : "DoShutdown",
2020/04/30 16:48:57.467318 SMCSDK [8910]          "template" : "%tdoshutdown",
2020/04/30 16:48:57.467361 SMCSDK [8910]          "timeout" : 600,
2020/04/30 16:48:57.467399 SMCSDK [8910]          "tooltip" : "Shut down the
server"
2020/04/30 16:48:57.467435 SMCSDK [8910]       },
2020/04/30 16:48:57.467460 SMCSDK [8910]       {
2020/04/30 16:48:57.467492 SMCSDK [8910]          "label" : "Test",
2020/04/30 16:48:57.467536 SMCSDK [8910]          "method" : "Test",
2020/04/30 16:48:57.467571 SMCSDK [8910]          "template" : "%ttest",
2020/04/30 16:48:57.467613 SMCSDK [8910]          "timeout" : 600,
2020/04/30 16:48:57.467649 SMCSDK [8910]          "tooltip" : "Get back an OK"
2020/04/30 16:48:57.467674 SMCSDK [8910]       },
2020/04/30 16:48:57.467710 SMCSDK [8910]       {
2020/04/30 16:48:57.467743 SMCSDK [8910]          "label" : "Wait10",
2020/04/30 16:48:57.467786 SMCSDK [8910]          "method" : "Wait10",
2020/04/30 16:48:57.467821 SMCSDK [8910]          "template" : "%twait10",
2020/04/30 16:48:57.467862 SMCSDK [8910]          "timeout" : 600,
2020/04/30 16:48:57.467911 SMCSDK [8910]          "tooltip" : "Return the
date/time asynchronously after 10 sec"
2020/04/30 16:48:57.467959 SMCSDK [8910]       },
2020/04/30 16:48:57.467986 SMCSDK [8910]       {
2020/04/30 16:48:57.468017 SMCSDK [8910]          "label" : "ParamTest",
2020/04/30 16:48:57.468061 SMCSDK [8910]          "method" : "ParamTest",
2020/04/30 16:48:57.468109 SMCSDK [8910]          "template" : "%tparamTest
%p",
2020/04/30 16:48:57.468138 SMCSDK [8910]          "timeout" : 600,
2020/04/30 16:48:57.468190 SMCSDK [8910]          "tooltip" : "Echo a parameter
back to the caller"
2020/04/30 16:48:57.468216 SMCSDK [8910]       }
2020/04/30 16:48:57.468239 SMCSDK [8910]    ]
2020/04/30 16:48:57.468273 SMCSDK [8910] }
2020/04/30 16:48:57.468317 SMCSDK [8910] ' } overwrite=1: 1 entries
2020/04/30 16:48:57.482091 SMCSDK [8903] Received event PayloadDeviceAdded
type PayloadRegistrationEvent/LiquidR-SMC and published to 0 subscribers
2020/04/30 16:48:57.483976 SMCSDK [8910] defaultBoardID = 0x21
```

- From another terminal, send commands to BasicSample.

```
$ cd /home/SMCDev/SMC/mpp2/common/Debug
$ ./run_ServerExec BasicSample test
OK
$ ./run_ServerExec BasicSample wait10
Thu Apr 30 16:50:33 PDT 2020
```

- On the BasicSample terminal, you see:

```
Waiting 10 seconds
Formatting command complete event
Sending command complete event
Received event name=CommandComplete typename=SendCommandCompleteEvent domainname
=LiquidR-SMC
```

- In the same terminal send:

```
$ ./run_server BasicSample ‘paramTest hello’
You sent me the following parameter: hello
```

- To end the SMC sample, send the following commands and close all the terminal windows.

```
$ sudo pkill omniNames
$ rm -f /home/SMCDev/SMC/mpp2/common/Debug/children.txt
```

APPENDIX C: RECOMMENDED READING

C.1 C++

Boost Template Library for C++ Development: http://www.boost.org/

Standard Template Library for C++ Development: http://www.cplusplus.com/reference/

C.2 CORBA

omniORB 4.1 Reference: http://omniorb.sourceforge.net/omni41/omniORB/

C.3 Eclipse

Eclipse web site: http://www.eclipse.org

C.4 Lua

Lua Reference Manual (Lua 5.1): http://www.lua.org/manual/5.1/

Programming in Lua: http://www.lua.org/pil/

C.5 Nvidia Tegra X1

One may want to sign up as a developer with Nvidia: https://developer.nvidia.com/content/tegra-x1

C.6 Ubuntu

Official Ubuntu Documentation: https://help.ubuntu.com/

Ubuntu Community Documentation: https://help.ubuntu.com/community/

C.7 Other Useful Links

makepp home page: http://makepp.sourceforge.net/

VirtualBox downloads: https://www.virtualbox.org/wiki/Downloads