Getting Raspberry Pi and Cosmos Communicating

Version 2

Guilherme Korol

6/30/2015

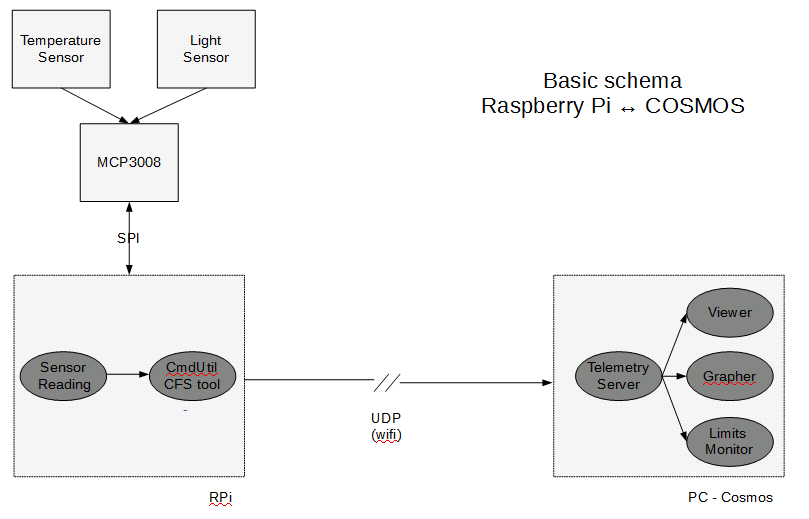
Edited by Rebeca Rodrigues

06/01/2016

**Introduction**

This document will guide you in the installation process of Cosmos, The Core Flight System (CFS), and other additional pieces of software in order to establish communication between a Raspberry Pi-based CubeSat and a Control Center using the Cosmos from Ball Aerospace. First, a telemetry stream will be configured to run from a Raspberry Pi to Cosmos.

In order to understand the big picture, take a look at the schema.



The CubeSat is using a Raspberry Pi connected to an analog digital converter, which is reading light and temperature sensors. SPI protocol is used to communicate Raspberry Pi and the MCP3008 (ADC). Other sensors can be added.

In the future, the full integration of the Pi with the Core Flight System is planned, and its tool will be used as an UDP packet sender. The current system works as follows: the sensor reading, which is proceed by a python script, is getting raw data from the ADC. The MCP3008 is a 10 bit converter, hence it is delivering a binary number from 0 to 1023 from each sensor channel. The program sensors.py is converting this number to a four ASCII characters string. This was done because cmdUtil (the CFS tool) expects a string of characters as input. After this first conversion, sensors.py ‘call’ cmdUtil to send the data. This has the destination IP address with port, command ID, packet ID, and the ‘string’ (in this case, the data from both sensors – in the format: TTTTLLLL, where TTTT is the temperature value and LLLL is the light value).

In fact, CmdUtil is a command sender for CFS applications. Here, it is being used as a general data sender. In the future, a field such as the command ID can be used to identify what kind of data is being sent.

The control center software Cosmos is running on a Windows 7 machine. Some configuration is required to create a new ‘target’ and interface protocol as well as break the UDP packets into well-known fields. Additionally, two Ruby scripts are responsible for converting the incoming data string to measurable units. Light is converted to a percentage value and temperature is converted to degree Celsius. “Target” is what Cosmos calls the Raspberry Pi.

**Step by step configuration**

Some steps were retrieved from Cosmos and CFS readme files.

**- Installing Cosmos on PC.**

Download Cosmos from <https://github.com/BallAerospace/COSMOS> .

Unzip the file.

* **Windows**

Under COSMOS-master\vendor\installer\windows edit the file INSTALL\_COSMOS.bat

Line 28: Change to if “%USERDNSDOMAIN%”==“AERO.BALL.COM” (

Line 134: Change to if “%PROCESSOR\_ARCHITECTURE%”==“x86” (

Line 186: Change to if “%PROCESSOR\_ARCHITECTURE%”==“x86” (

Using the windows terminal go to COSMOS-master\vendor\installer\windows and type INSTALL\_COSMOS.bat

* **Linux**

Run the following command in a terminal:

bash <(\curl -sSL <https://raw.githubusercontent.com/BallAerospace/COSMOS/master/vendor/installers/linux_mac/INSTALL_COSMOS.sh>)

During the installation, questions may be asked. Answer “yes” to yes/no questions, and answer “s” to srn questions in which you are asked if you want to run as root, sudo or none.

If needed install curl: sudo apt install curl

Cosmos should be installed and ready to run. Cosmos website: cosmosrb.com

**- Adding TARGET**

- Adding target files. (files in NecessaryFiles\_COSMOS-RPI). Few explanations:

* Cmd\_tlm\_server.txt: set communications parameters. Set the target’s address in lines 18: “INTERFACE RPI\_INT udp\_interface.rb <ip\_address> 1235 5005 nil nil 128 10.0 nil”.
* System.txt: declare target. Apparently it also disables sound. It is a modification of original system.txt, the modifications are in line 12 (“DECLARE\_TARGET RPI”) and in line 33 (“# ENABLE\_SOUND”).
* About the RPI folder:
  + Target.txt: every target has its target.txt, it points what are the ‘components’ of the target.
  + Cmd\_tlm folder: list of commands and their parameters (rpi\_cmds.txt) and how the telemetry data is decoded (rpi\_tlm.txt)
  + Lib folder: ruby codes called to convert the light and the temperature data. They are called on rpi\_tlm.txt.

- **Windows**

Replace C:\COSMOS\Demo\config\tools\cmd\_tlm\_server\cmd\_tlm\_server.txt

Replace C:\COSMOS\Demo\config\system\system.txt

Add tlm\_grapherRPI.txt to C:\COSMOS\Demo\config\tools\tlm\_grapher\

Add tlm\_Raspberry\_Light\_Temp.txt to C:\COSMOS\Demo\config\tools\tlm\_extractor

Add the folder RPI to C:\COSMOS\Demo\config\targets\

**- Linux**

Replace ~/cosmosdemo/config/tools/cmd\_tlm\_server/cmd\_tlm\_server.txt

Replace ~/cosmosdemo/config/system/system.txt

Add tlm\_grapherRPI.txt to ~/cosmosdemo/config/tools/tlm\_grapher/

Add tlm\_Raspberry\_Light\_Temp.txt to ~/cosmosdemo/config/tools/tlm\_extractor

Add the folder RPI to ~/cosmosdemo/config/targets/

**- Installing CORE FLIGHT SYSTEM on the pi:**

- **Building and running NASA developed open source Core Flight Executive.** At this moment the full Core Flight Executive.

If you intend to use graphical interface for the python tools on CFS install qt4, type on terminal sudo apt-get install python-qt4 pyqt4-dev-tools

Download Core Flight Executive and OSAL. <https://sourceforge.net/projects/coreflightexec> (Download the 6.4.1 version) and <https://github.com/nasa/osal>

Unzip both files.

Copy the content from osal-master\ to cFE-6.4.1-oss-release\osal\

Now, some Makefiles modifications are needed. (Maybe that is not the best or right way to do it). The need for some of these modifications are that the flag -m32 is no longer used in raspberry pi. This flag usually points issues related to the architecture, but the architecture of a pi is by default 32-bits, in such a way this flag no longer exists on pi.

Under cFE-6.4.1-oss-release\psp\fsw\pc-linux\make\ edit compiler-opts.mak commenting out ARCH\_OPTS line.

Also, set linker options to ld –m armelf\_linux\_eabi (LINKER = ld –m armelf\_linux\_eabi).

It is originally LINKER = ld -melf\_i386.

Still under cFE-6.4.1-oss-release\psp\fsw\pc-linux\make\ edit link-rules.mak deleting -m32 from line 21 (LDFLAGS = -m32 -Wl,-export-dynamic)

Go back to cFE-6.4.1-oss-release\ type . ./setvars.sh

Under cFE-6.4.1-oss-release\build\cpu1\ type make config

You are going to get an error. Under cFE-6.4.1-oss-release\build\cpu1\elf2cfetb\ edit Makefile removing –m32 from ARCH\_DEFS line.

\*PS: the directory referred above is only created DURING the **make config** command\*

Again, under cFE-6.4.1-oss-release\build\cpu1\ type make config

Under cFE-6.4.1-oss-release\build\cpu1\ type make

Under Under cFE-6.4.1-oss-release\build\cpu1\exe run core-linux.bin (./core-linux.bin)

CFE core should be running with no applications.

- **Compiling cmdUtil**

Under cFE-6.4.1-oss-release\cfe\tools\cmdUtil\ type make

OR instead:

under cFE-6.4.2-OSS-release\cfe\tools\cFS-GroundSystem\Subsystems\cmdUtil\ type make

**- Sensors**

- Preparing and using sensors.py. Files on NecessaryFiles\_CFS.

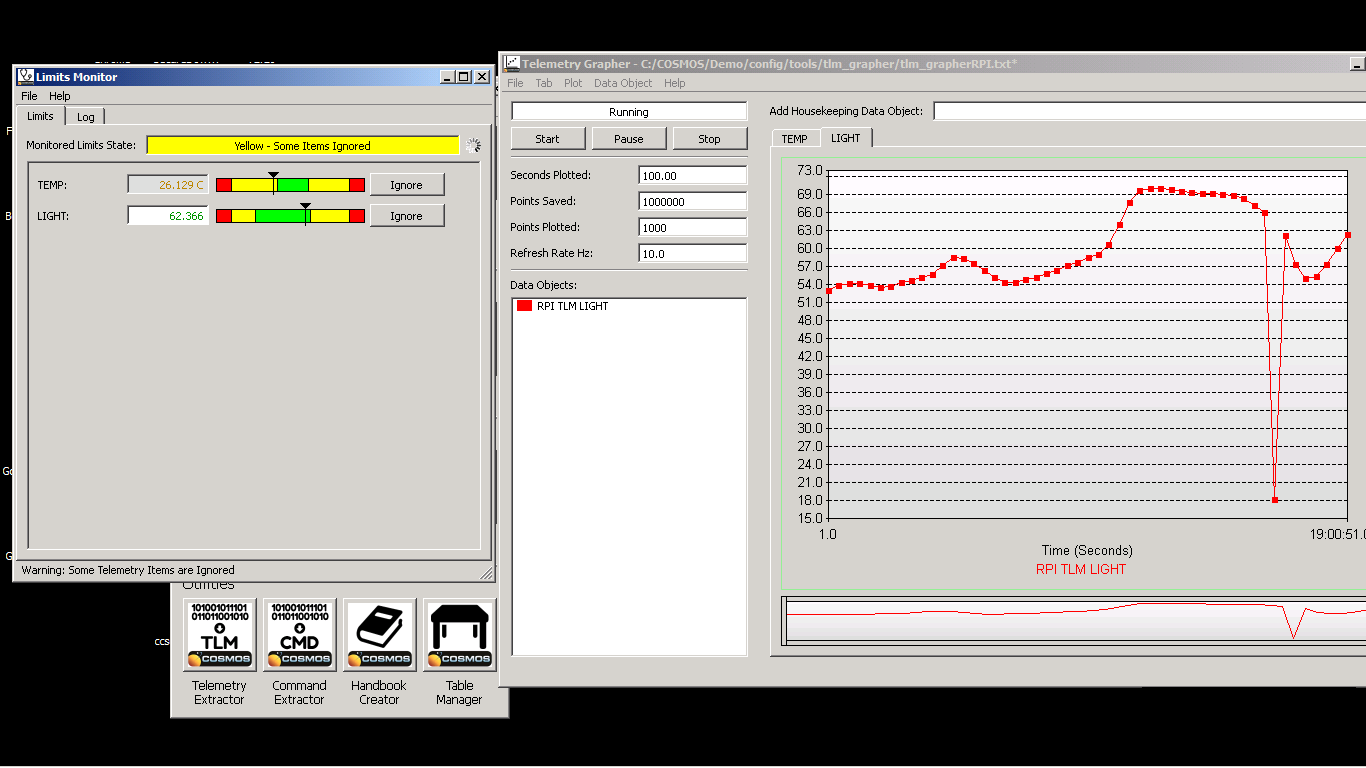
Copy the folder flatSat to cFE-6.4.1-oss-release\cfe\tools

**- Telemetry**

- Set connection between sensors.py and the telemetry system. Files on NecessaryFiles\_CFS.

Copy EventMessages.py to cfe\tools\tlmUtil

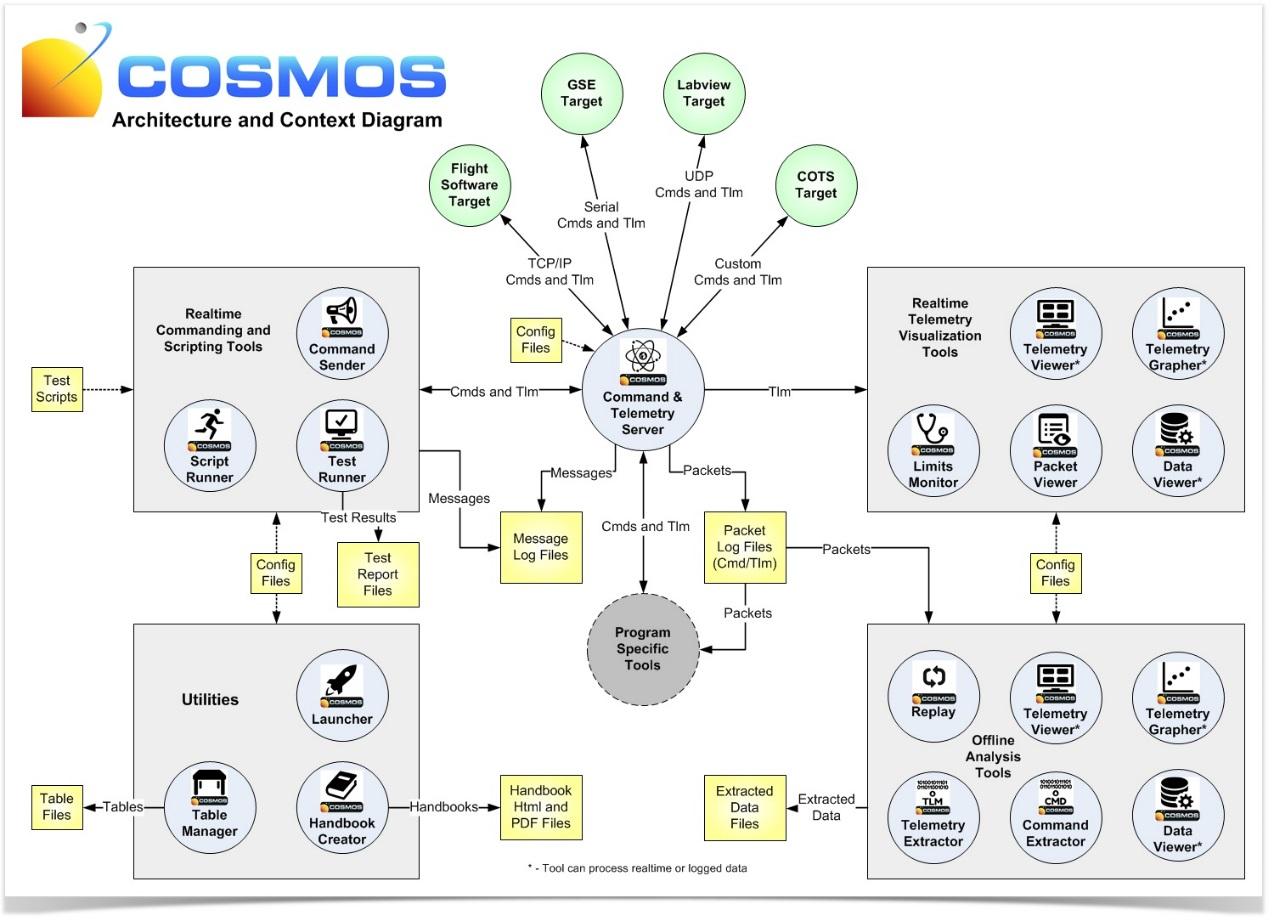
**Cosmos screenshot:**



**Cosmos:**

“Cosmos is a piece of open source software from Ball Aerospace (see, http://www.cosmosrb.com). It implements the functions of a satellite control center. It can be used for simple embedded systems, cubesats, or larger satellite systems, both in orbit, and under test.”

“It Includes functions such as data ingest and logging, limits monitoring, a packet viewer, a telemetry viewer and grapher, the ability to create telemetry and command handbooks, command sender, and an ability to replay logged telemetry data. It includes a demo mode. Data is accepted over Ethernet or serial connections. Cosmos runs on a laptop or desktop environment.”



(Diagram from Ball Aerospace)

**CFS:**

“The Core Flight Executive is a portable, platform independent embedded system framework developed by NASA Goddard Space Flight Center. This framework is used as the basis for the flight software for satellite data systems and instruments, but can be used on other general embedded systems.”