PURDUE
UNIVERSITY

Radio Occultation (ROC) Software Documentation

and Operations Manual

Revision: V15
Date: 23 September 2010

Prepared by:
Jennifer Haase and Phil Wyss
ROC Concordiasi PI
Purdue University
Department of Earth and Atmospheric Sciences
550 Stadium Mall Dr
West Lafayette, IN 47907-2051

Revisions

Revisions	Date	Description	Pages	edited by
V01	13 July 2009	Separate documents prepared by P. Wyss	All	
V02	9 Sept 2009	Combined into one document	All	
V03	2 Feb 2010	Added ground processing section	22-27	
V04	8 Mar 2010	Corrected for changes in filename conventions and post-processing changes	22 and beyond	
V05	23 Mar 2010	Corrections and amplification to post processing features	22 and beyond	
V06	8 Apr 2010	Added install notes to section 2 Added SolPSB notes to section 3		JH
V07	22 Apr 2010	Changes reflecting separate position/occultaion angles. Update to	7 through 43	PjW

		changeConfig.documentation. Addition of @Status device independent command. Make command line UTC offset in rt17_to_rinex most authoritative.		

Additions to make:

10/20/2010:
status -r serial 2
will get you the serial number of receiver 2

7/6/2010
The following snippet should be added to the device independent command list for the CNESserver. I could not test it completely, but since the coordinate program (which is the program that registers itself) is highly decoulpled from the CNESserver, I'm pretty sure this will work. The syntax is funky to make more digits available in the small TC packet.

==========

@@:ddddd - delay ddddd minutes before signaling configuration changes. The ddddd are decimal digits.. The delay is in minutes after a subsequent @FINISH is issued. Note the configuration is updated at @FINISH time and normally all processes registered for notification are signaled. It is only this notification that is delayed. If the system were shut down during the delay, the new configuration would be seen on startup. If the collection process failed during the delay, the new collection process would use the new configuration. Less than five digits may be specified.

@START - This is an option "start command block" command. Currently it's only purpose is to clear the delay timer. This can be used to make a change in the configuration after a delay timer has been issued. If the delay has not elapsed, the notification of the previous configuration change is lost. The next @FINISH command will signal all registered listeners immediately unless another delay command is entered after the start command.

==========

The changeConfig program list a few more configurations (Stream Ephemeris: YES) but they are not choices during installation. The defaults used in initialization is what the collect program expects for normal operation. They are became variable as I was modifying the collect program and shifted design several times. In particular the raw position data is not streamed because the ECEF position is streamed in the GSOF record that also contains the GPS week at a lower rate.

When I send the following commands at 2300 Z on Oct 22:

@@:00060 (Setting delay timer to 00060 minutes from current time)

L10020 (Set lower elevation threshold to 20 degrees)

@FINISH (Writes a new config_state file)

When the commands are executed at 0000 Z on Oct 23 and the configuration file is changed, are all the files that are in transmit deleted?
It appears kind of like that occurred. I am still checking this and will update
when In know more thoroughly

The files in transmit are not designed to be deleted at that time. Coordinate should simply terminate the current collect process, reset the receiver and start a new collect with the new configuration. However, any parts of the current dataset that have not been arbitrated completely are lost as the arbitrator knows that the new collect will not have all the state information to send the appropriate events that will drive the arbitration for the dataset created by the old collect process.

==========

Table of Contents

1 Instrument Operation Principal....5
2 ROC system components schematic....6
3 Quick Start Guide....7
4 Instrument Flight Operation Plan....7
4.1 ROC....7
4.1.1 Operation principle....7
4.1.2 Starting/stopping....8
4.1.3 TC command message....8
4.1.4 TM measurement message....9
4.1.5 Description of TC messages for ROC....10
4.1.6 Example command sequences for special observation periods....12
5 Installation and Testing....15
5.1 Startup Sequence....15
5.2 Shutdown Sequence....15
5.3 Startup Sequence for ROC only test configuration (no PSB)....16
5.4 Shutdown Sequence for ROC only test configuration (no PSB)....16
5.5 Physical connections from ROC....16
5.6 Physical Connections from PSB to ROC and PC computer....17
5.7 Configuration of communications connections for monitoring....20
5.8 Optional physical connections....20
5.9 Preparation of data storage USB memory stick for operational mode....20
5.10 Configuration of alternate communications connections....21
5.10.1 Troubleshooting IP connection....24
6 ROC software installation and testing....24
6.1 The SBC full linux development vs linux lite operational environments....25
5.1 Running ROC software in test mode from the full linux development env....29
6.2 Testing communication between ROC and PSB without GPS receivers....31
5.2 Running the ROC software in operational mode....31
6.3 Onboard data storage locations....33
5.3 Installing the software....33
5.4 Testing....35
5.5 Changing configuration locally....37
5.6 Installation (Alexandria Version)....38
6 SolPSB Interface Software....43
6.1 PSB Software installation....43
6.2 Setting up the first flight....44
6.3 Running the PSB Software....45
6.4 Quickstart to interrogate the PSB:....46
6.5 Detailed description of SOL PSB V2 software display screen....48
6.6 Notes on Frame counting....50
6.7 Monitoring the PSB using hyperterminal....50
6.8 SolPSB Notes and email clarifications....53
5 Ground Post-processing Software and Archiving Programs....53
5.1 Data Format....53
5.1.1 Pathnames on ROC....53

5.1.2 Filenames on ROC....54
5.1.3 Pathnames for files received from Ground Station....55
5.1.4 Filenames for Decoded Data Files....55
5.2 Decoding CNES Sequence Files....56
5.2.1 resequence....56
5.2.2 unarchive....56
5.2.3 rt17....57
5.2.4 RT17 to RINEX....58
6 PSB - ROC interface diagrams....61
7 GPS ROC Onboard Software Architecture....63
7.1 Pending requests for software upgrades....63
7.2 GPS ROC software schematic....64
7.3 Architecture of the coordinate program....65
7.4 Architecture of the collect program....65
7.5 Architecture of the arbitrator program....66
7.6 Architecture of the CNES Server....67
7.7 CNESserver API....70
7.8 Software Control Commands....73
8 Clarification Emails....77
9 Email Clarifications (installation and testing)....80

1 Instrument Operation Principal

The ROC, Radio OCcultation instrument, records dual frequency GPS signals from setting GPS satellites. The signals, which pass nearly horizontally through the atmosphere are refracted and delayed by the non-zero index of refraction. The index of refraction depends on pressure, temperature, and humidity. By measuring the bending of the signal ray path, a profile of refractivity and thereafter, temperature and humidity can be retrieved.

During the period the satellite is setting, the phase data (inferometric measurement of range) from that satellite is recorded as well as the range data from a non-occulting high elevation reference satellite for time calibration. Two GPS receivers are contained in the ROC to view the horizon in opposite directions.

The precise position of the receiver must be known. The dual frequency range data from 8 to 10 non-setting satellites are recorded in order to determine a precise position of the balloon. The post-processed position results will also be useful for observing gravity waves when used in combination with pressure measurements recorded by TSEN.

The GPS Radio OCcultation (ROC) software manages the operation of the two GPS receivers contained in the unit, repackages the data, makes the data available for transmission to the balloon Payload Supervisory Board (PSB), and manages the communication with the Payload Supervisory Board for commands and data transmission.

Since the full datastream is too voluminous to send, only the best occultation data is sent to the ground. The general algorithm for data selection is as follows:

- Data is collected continuously for all satellites at a high sample interval set by the parameter o_rate.
- At each data collection interval, each satellite is monitored to determine if it is setting (elevation angle decreasing) or rising (elevation angle increasing) and whether it is below l_elev, the lower elevation angle threshold.
- When a satellite is below l_elev, all of the high rate data (sampled at o_rate) for the setting satellite and one high elevation satellite will be stored, until the setting satellite disappears.
- The data for all satellites is resampled at a lower sample rate, r_rate, to calculate the position for all satellites that are above u_elev, the upper elevation angle threshold. These data will be passed on to be transmitted to the ground.
- After num_occ occultations have occurred, the one best recording will be selected among all the occultations and considering recordings from both of the receivers. The selected dataset will be transmitted to the ground (this requires a long delay before the first occultation data set is sent to the ground)
- The high sample rate data for the non-chosen occultations will be discarded.
- The high sample rate data for non-occulting satellites will be discarded, except for the r_rate data samples.
- In practice, it can take 1.5 hours for 3 satellites to set, so if num_occ is set to 3, then nothing is transmitted to the ground until then.

2 ROC system components schematic

Two instrument schematics are shown below, one for the ROC system as it will be deployed in the stratospheric balloons, and a second which represents the typical laboratory testing setup.

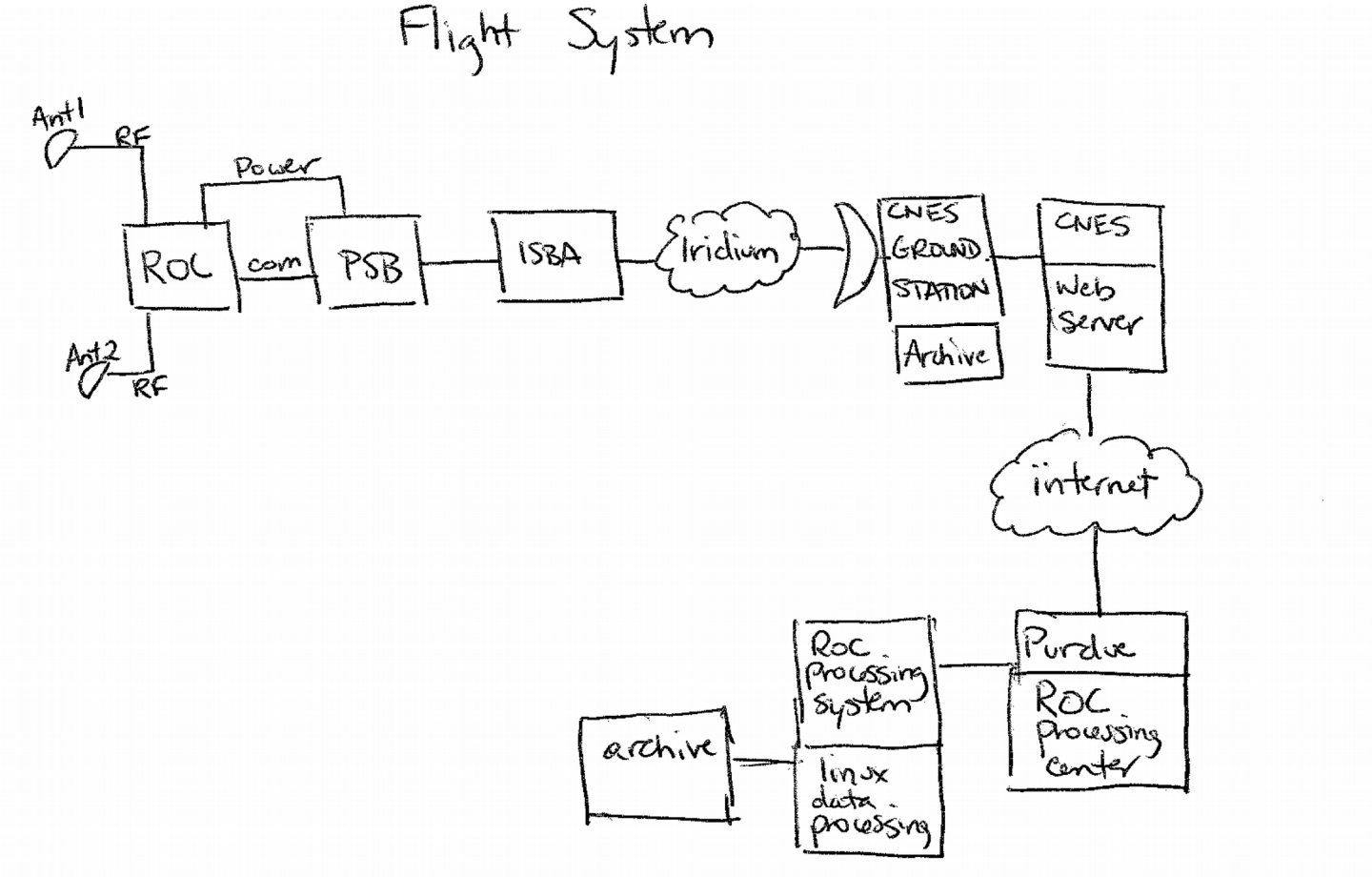

Figure 1 Components of the ROC in its configuration in the balloon flight system.

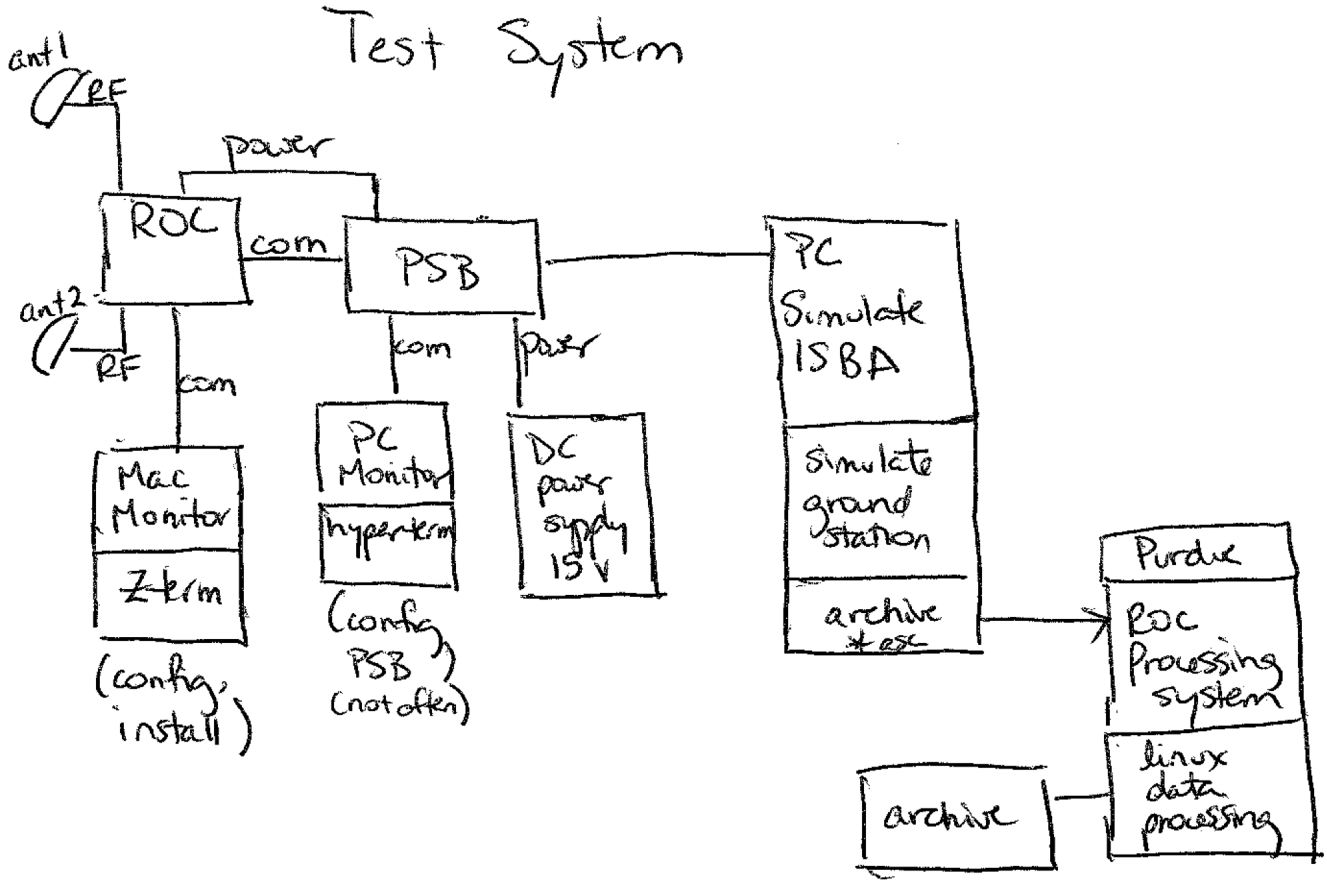

Figure 2 ROC instrument in its test configuration.

3 Quick Start Guide

4 Instrument Flight Operation Plan

4.1 ROC

(this section is the description provided to CNES for their document:
CIASI-NT-0-1352-CN_v3_2010-06-01_inst-oper-plan.doc

4.1.1 Operation principle

The ROC, **R**adio **OC**cultation instrument, records dual frequency GPS signals from setting GPS satellites. The signals, which pass nearly horizontally through the atmosphere are refracted and delayed by the non-zero index of refraction. The index of refraction depends on pressure, temperature, and humidity. By measuring the doppler shift and therafter deriving bending of the signal ray path, profiles of refractivity, temperature and humidity can be retrieved.

The dual frequency range data from non-setting satellites are also recorded in order to determine a more precise vertical position of the balloon, which is required for deriving the occultation profiles and which will also be useful for observing gravity waves when used in combination with pressure measurements recorded by TSEN.

4.1.2 Starting/stopping

The ROC is operated continuously. It is started before launch and is only stopped if power resources become limited and power consumption must be reduced.

The ROC will be activated in the following manner:
- Activation by a "Forcé ON" command before launch.
- In exceptional situations where power is low during the flight, the ROC will be
    - stopped by the command "Forcé OFF"
    - restarted by the command "Forcé ON"

When the ROC receives the shutdown command it will begin its own power down process so that the instrument is off before the power relay is finally cut.

PSB configuration can specify the duration of warmup
(TBC) Do I need to choose something operationally here?

4.1.3 TC command message

Operation of the ROC occasionally requires sending a TC command message to make adjustments to
- certain calculation parameters (ex: elevation angle and occultation duration constraints for occultation measurements)
- measurement mode and sample rate

This type of command will be sent following a request from the ROC scientific investigators transmitted to the CCT at least 12 hours in advance of the requested time for the command to be executed.

Regardless of the type of data recorded by the ROC, either in a routine observation period or intense observation period, the packets transmitted to the PSB will have the same structure so will be transparent to the PSB.

4.1.3.1 Routine observation period

In routine operation (default operation mode) the ROC will record both occultation and positioning data continuously from one receiver. It will remain in this mode, sending one data packet to the PSB each minute, indefinitely.

4.1.3.2 Intense observation period

Intense observation periods will be scheduled with more than 12 hours in advance for the following objectives:
- High sample rate positioning data only, for one receiver, for intense gravity wave sampling, ie, over the Antarctic peninsula.
- Increased number of occultation recordings selected, coordinated with dropsonde releases.
- High sample rate positioning with two receivers, for understanding balloon rotation and dynamics.

These three modes will result in accumulation of data internal to the ROC at a rate greater than can be transmitted to the PSB. Therefore, following each of these modes, the ROC will be configured to stop individual GPS receiver operation for a limited time (on the order of a day) while leaving the ROC SBC powered on to finish transmitting the data to the PSB. Then the ROC will be returned to routine observation.

4.1.4 TM measurement message

4.1.4.1 Description

The nominal sampling interval for occultation measurements, intocc = 5 sec
The nominal resampling interval for position range measurements, intpos = 30 sec

The data is stored in files in a transmit directory in the ROC data partition. The CNES_server process running internally on the ROC breaks the files up into consecutive frames, with additional ROC packet information added to the start and end of each packet in order to reconstruct the original ROC data file. When the PSB sends a TC/TM request for telemeasure, one frame is sent. The TM frame header bytes are removed when the data are received at the CCT. The ROC science team retrieves the datafiles from the CCT. The ROC packets are then reconstituted into the original data files for further processing.

One TM frame is sent upon request each 1 minute continuously.
The length of a frame is 1028 bytes.
5 bytes of these are used for frame packaging and 2 bytes are used for the frame counter.
This leaves 1021 bytes available for scientific data.
6 bytes of the science data are used for ROC packaging information.
This leaves 995 bytes available for the actual ROC data measurements.

8 pages of 64 Kb memory on the PSB are allocated to the ROC.
8 pages * 64 Kb / 1028 bytes gives 510 frames that can be stored on the PSB.
510 frames at one frame per minute implies that 8.5 hours of memory is available.
Therefore the ROC data must be downloaded at least every 8.5 hours.
Currently the CCT is planning to download ROC data 3 times per day.

The amount of actual scientific data collected depends on the number of satellites that are being recorded and the compression of the data.
Current empirically determined data rates estimates are summarized below:

Positioning: ~30 bytes/satellite/sample for 120 Kb files
Occultation: ~40 bytes/satellite/sample

With these estimates and the values obtained from the simulation we've determined that the following configurations are our current best bet for getting the most, high quality data we can while staying close to our 1.2 Mb/day limit (assuming the receivers run in the described mode for an entire day):

A – One receiver in occultation and positioning mode
30 second position sampling, 5 second occultation sampling
6 occultations of data kept, with occultations lasting 16 min
Amount of data/day: 1.2 Mb

B – One receiver positioning mode
20 second position sampling
Amount of data/day: 1.24 Mb

4.1.5 Description of TC messages for ROC

Current occultation software version: 282

A configuration file is stored in the ROC, which specifies the operating configuration. The configuration can be modified in flight by a remote user using TC commands through the SolPSB ground station software. Device dependent TC commands are used for modification of this configuration file. Device independent TC commands require no knowledge of the configuration file. Once the device dependent commands are specified to modify the configuration file, the command @FINISH sends a signal to all the ROC software processes that the configuration has been updated and at this point the commands take effect.

The commands specified below are entered directly into the command box in the SolPSB software. The software adds the "TEXT" characters to the command to complete the TC packet before transmitting the command to the PSB.

Device independent commands, and only those commands, begin with an @ character, are shown below:

@FINISH – Write the configuration to a file with a standard path name and signal all requesting processes that the configuration update has completed. If the system should shut down or fail between before this command is sent, the updated configuration (and all the commands that have been sent to update the configuration) will be lost.

@HARD - Do a hard reset. The system writes the file /reset and a reboot of the system is forced with no acknowledgment of the command nor notification to any other system module. The reboot process detects the presence of the /reset file, causing it to remove all datasets and segments from the system. The default configuration will be copied to the current configuration file after rebooting. A hard reset should be used as a last ditch effort.

@SOFT - Do a soft reset. This is exactly the same as a hard reset except that it will not do an immediate reboot. The actions to be taken on reboot will be done on the next normal shutdown/startup cycle. It will also remove all datasets and segments from the system transmit directory. A soft reset is the preferred way to reset the system. This should be followed by a power off and power on sent to the ROC via the PSBSol interface.

@STATUS - Transmit the file count and total byes of all files in /var/occult/transmit waiting to be transmitted. when it finished the current buffer full of data it will send the status information in the next buffer. When you run resequence on that data, then you will see the info on size, etc.

@@:ddddd - The command sets up a delay in minutes after a subsequent @FINISH is issued before a new configuration file is initiated. Note the configuration is updated at @FINISH time and normally all processes registered for notification are signaled. It is only this notification that is delayed. If the system were shut down during the delay, the new configuration would be seen on startup. If the collection process failed during the delay, the new collection process would use the new configuration. Less than five digits may be specified.

@START - This is an option "start command block" command. Currently it's only purpose is to clear the delay timer. This can be used to make a change in the configuration after a delay timer has been issued. If the delay has not elapsed, the notification of the previous configuration change is lost. The next @FINISH command will signal all registered listeners immediately unless another delay command is entered after the start command.

Device dependent commands are shown below. Commands are a single character followed by up to three parameters. Each parameter may be either a four digit decimal number (dddd), a single decimal digit representing a device number or a true/false indicator (T/F or 0/1). Note that numeric data (indicated by dddd) are up to four digits (or an initial sign and three digits) that are left justified and space padded.

The power on/off command:

Powers a GPS receiver on or off and if on, begins the current application

P*DS* - where *D* is a device number and *S* is the true/false state of the power being on.

Example:

P1T - set power on for device 1
P2F - set power off for device 2

Set Elevation angle upper threshold:

Sets the elevation at which position recording is stopped.

U*Ddddd* – where *D* is the device number and *dddd* is the elevation in degrees. This may be signed.

Set Elevation angle lower threshold:

Sets the elevation at which occultation recording is started.

L*Ddddd* – where *D* is the device number and *dddd* is the elevation in degrees. This may be signed.

Maximum Gap:

The time in seconds since the last transmission of a single satellite before it is assumed that the satellite has set.

G*Ddddd* – where *D* is the device number and *dddd* is an unsigned decimal number.

Resample Rate:

Rate at which data is recorded when satellite is not in occultation or rate at which data is resampled for position application.

R*Ddddd* - where *D* is the device number and *dddd* is an unsigned decimal number. This will be coerced to a multiple of the occultation rate in seconds per record.

Occultation Rate:

Rate at which data is recorded during a satellite's occultation period. This is the rate that the receiver outputs data and therefore must be a rate supported by the receiver. The rate is in seconds per record. When the receiver is in positioning mode only, the higher sample rate specified by Occultation Rate is used for the positioning data and the Resample Rate is not used.

O*Ddddd* - where *D* is the device number and *dddd* is an unsigned decimal integer in seconds per record.

Occultation Selection:

This is the number of occultations to be acquired before considering which is to be transmitted.

S*dddd* – where *dddd* is an unsigned number. The maximum is 32. Note that this is not device dependent, but it is listed here because it is implemented in config.c to keep the high level of modularity for the CNESserver. The server code knows nothing of the actual data or devices it serves except in the config.c module.

Application Mode:

Sets the application mode to position or occultation for a device.

M*DA* – where *D* is the device number and *A* is either P or O.

Dataset Size:

This sets the amount of data that the position application will acquire before attempting to send it off. This is done by closing the current dataset causing the arbitrator to transmit up to the CLOSE_DATASET event. The size is given in units of 1K bytes.

K*Ddddd* – where D is the device number and dddd is the unsigned size

4.1.6 Example command sequences for special observation periods

Below example command sequences are provided to go from the default positioning observation mode to occultation observation mode and the special high sample rate positioning observational mode and then return to default operational mode. Note that the commands being used also depend on the version of occult software installed on the ROC.

Setting up the routine operational mode default configuration prior to launch using zterm:

To see what the default configuration parameters are for receivers 1 and 2:

```
$ changeConfig –d 1 /var/occult/default_config_state
$ changeConfig –d 2 /var/occult/default_config_state
```

To see what the current configuration parameters being used by receivers 1 and 2:

```
$ changeConfig -d 1
$ changeConfig -d 2
```

To see all environment variables that are set and required for routine operation:

$ export

To see the program that run on start-up to set parameters:
$ ls /etc/init.d

Method 1 for resetting all the parameters and configuration files:

-Reboot the ROC
$ reboot
-Log in
-Kill unneeded processes:
$ ps
$ kill [watch.sh process i.d.]
$ kill [CNESserver i.d.]
-To change the default_config_state file:
$ changeConfig -n 2 /var/occult/default_config_state
$ changeConfig -d 1 -p /dev/ttyTS0 -P 1 –m O –g 1800 –k 120 –o 5 –r 30 –l 0 -u 20 -f C -I ephemeris /var/occult/default_config_state
$ changeConfig -d 1 -p /dev/tts3 -P 0 –m O –g 1800 –k 120 –o 5 –r 30 –l 0 -u 20 -f C -I ephemeris /var/occult/default_config_state
$ changeConfig -s 20
-Tell the system to use the default_config_file as config_file
$ touch /reset
-Reboot the ROC
$ reboot

Method 2 for resetting all parameters and configuration files:
(Note: The system MUST be in deploy mode to use this method for resetting)

-Reboot the ROC
$ reboot
-Log in
-Kill unneeded processes:
$ ps
$ kill [watch.sh process i.d.]
$ kill [CNESserver i.d.]
-Removing unwanted files
$ rm /var/occult/default_config_state
$ cd /root/install
$ rm ./init.d/device_address.sh
-Running install and setting parameters as prompted:
$ sh install.sh

> What is the decimal device address of the ROC for this flight? **(enter ROC address #)**
> What is the decimal flight ID for this flight? **(enter ROC flight #)**
> Initial number of occultations to consider before selection? **20**
> Adjust system time on synchronization frame (y/n)? **y**

Starting installation process ...

> Building default configuration for receiver 1 at serial port /dev/ttyTS0
> Upper elevation threshold (integer degrees): **20**
> Lower elevation threshold (integer degrees): **0**
> Gap used to declare satellite set (integer seconds) [default 1800]: **1800**
> Occultation sample rate (integer seconds) [default 5]: **5**
> Position/re-sample rate (integer seconds) [default 30]: **30**
> Data set size for position application (integer KB) [default 4]: **120**
> Application Occultation/Position (O/P): **O**
> Receiver power enabled (Y/N) [default Y]: **Y**
> Initial number of occultations to consider before selection: **20**
> Use sync packet for adjusting time (y/n): **y**

> Building default configuration for receiver 1 at serial port /dev/tts3
> Upper elevation threshold (integer degrees): **20**
> Lower elevation threshold (integer degrees): **0**
> Gap used to declare satellite set (integer seconds) [default 1800]: **1800**
> Occultation sample rate (integer seconds) [default 5]: **5**
> Position/re-sample rate (integer seconds) [default 30]: **30**
> Data set size for position application (integer KB) [default 4]: **120**
> Application Occultation/Position (O/P): **O**
> Receiver power enabled (Y/N) [default Y]: **N**
> Initial number of occultations to consider before selection: **20**
> Use sync packet for adjusting time (y/n): **y**

-Checking the configuration files:

```
$ changeConfig -d 1
$ changeConfig -d 2
$ changeConfig -d 1 /var/occult/default_config_state
$ changeConfig -d 2 /var/occult/default_config_state
```

Method 3 for resetting all parameters and configuration files:

-Reboot the ROC

```
$ reboot
```

-Log in

-Create a file that say the boot procedure has failed, such that upon rebooting the system will just go into quiescent mode

```
$ touch /failed
$ reboot
```

-Log in

-Reset /var/occult/default_config_state:

```
$ changeConfig -n 2 /var/occult/default_config_state
$ changeConfig -d 1 -p /dev/ttyTS0 -P 1 –m O –g 1800 –k 120 –o 5 –r 30 –l 0 -u 20 -f C
-I  ephemeris /var/occult/default_config_state
$ changeConfig -d 1 -p /dev/tts3 -P 0 –m O –g 1800 –k 120 –o 5 –r 30 –l 0 -u 20 -f C
-I  ephemeris /var/occult/default_config_state
$ changeConfig -s 20
```

-Reboot

```
$ reboot
```

Explanation of set parameters:

-n 2 = create new /var/occult/default_config_state file for both 2 receivers
-d 1 = for device 1 (GPS receiver 1) [1 = GPS1, 2=GPS2]
-p /dev/ttyTS0 = use port /dev/ttyTS0 of ROC SBC for communication with GPS 1 [/dev/ttyTS0 for GPS1, /dev/tts/3 for GPS2]
-P 1 = power on GPS 1 [0 = power off, 1= power on]
-m O = mode is occultation [O = occultation, P= positioning]
-u 20 = upper elevation angle threshold is 20 degrees, record positioning data for satellites above this threshold [0-90 degrees]
-l 0 = lower elevation angle is 0 degrees, record data for occulting satellites below this threshold [-10 to 90 degrees]
-g 1800 = gap of time that determines a satellite has set, and will not reappear
-o 5 = occultation data sample rate is 5 sec
-r 30 = re-sample rate for positioning data sampling is 30 sec (make o and r the same when in positioning mode)
-k 120 = dataset size for positioning data is 120 kbytes (not used in occultation mode)
-f C = format Concise [-f E = expanded, has more sig figs for SNR data]
-I ephemeris = get ephemeris from GPS messages and calculate elevation angle to determine if elevation angle in GPS message is reliable (always set this when running software version 280 and above).
-s 20 = select one occultation to transmit of every 20 that are observed (must be a separate changeConfig command)

SOLPSB commands for occultation software version 282:

This revision of occult contains both occultation prediction, through the use of ephemerides data, and the option of delaying the initiation of a newly created configuration file. The delay timer begins counting down once the @FINISH command is received.

- Default mode is occultation with 5 second sampling and 30 second re-sampling,
changing to default mode:

P11	(Receiver 1 powered on)
P20	(Receiver 2 powered off)
M1O	(Sets receiver 1 to occultation mode)
O10005	(Sets sampling rate to 5 seconds)
R10030	(Sets resampling rate to 30 seconds)
U10020	(Set upper elevation threshold to 20 degrees)
L10000	(Set lower elevation threshold to 0 degrees)
S0020	(Sets number of occultations to choose from to 20)
@FINISH	(Writes a new config_state file and restarts coordinate)

-Changing to default mode with the delay timer:

@@:ddddd	(Setting delay timer in minutes ddddd)
P11	(Receiver 1 powered on)
P20	(Receiver 2 powered off)
M1O	(Sets receiver 1 to occultation mode)

O10005	(Sets sampling rate to 5 seconds)
R10030	(Sets resampling rate to 30 seconds)
U10020	(Set upper elevation threshold to 20 degrees)
L10000	(Set lower elevation threshold to 0 degrees)
S0020	(Sets number of occultations to choose from to 20)
@FINISH	(Writes a new config_state file and restarts coordinate)

-Changing to high rate position sampling mode:

P11	(Receiver 1 powered on)
P20	(Receiver 2 powered off)
M1P	(Set receiver 1 mode to positioning)
O10015	(Set sampling rate to 15 seconds)
R10015	(Set resampling rate to 15 seconds)
U10020	(Set upper elevation threshold to 20 degrees)
L10000	(Set lower elevation threshold to 0 degrees)
K10120	(Set file size to 120 Kb)
@FINISH	(Writes a new config_state file and restarts coordinate)

-Changing to high rate position sampling mode with the delay timer:

@@:ddddd	(Setting delay timer in minutes ddddd)
P11	(Receiver 1 powered on)
P20	(Receiver 2 powered off)
M1P	(Set receiver 1 mode to positioning)
O10015	(Set sampling rate to 15 seconds)
R10015	(Set resampling rate to 15 seconds)
U10020	(Set upper elevation threshold to 20 degrees)
L10000	(Set lower elevation threshold to 0 degrees)
K10120	(Set file size to 120 Kb)
@FINISH	(Writes a new config_state file and restarts coordinate)

-Changing to high occultation selection mode:

P11	(Receiver 1 powered on)
P20	(Receiver 2 powered off)
M1O	(Set receiver 1 mode to positioning)
O10005	(Set sampling rate to 5 seconds)
R10030	(Set resampling rate to 15 seconds)
U10020	(Set upper elevation threshold to 20 degrees)
L10000	(Set lower elevation threshold to 0 degrees)
S10004	(Sets number of occultations to choose from to 4)
@FINISH	(Writes a new config_state file and restarts coordinate)

-Changing to high occultation selection mode with the delay timer:

@@:ddddd	(Setting delay timer in minutes ddddd)
P11	(Receiver 1 powered on)
P20	(Receiver 2 powered off)
M1O	(Set receiver 1 mode to positioning)
O10005	(Set sampling rate to 5 seconds)
R10030	(Set resampling rate to 15 seconds)
U10020	(Set upper elevation threshold to 20 degrees)

L10000 (Set lower elevation threshold to 0 degrees)
S10004 (Sets number of occultations to choose from to 4)
@FINISH (Writes a new config_state file and restarts coordinate)

-If a mistake has been made in the commands in the delayed message:

@START (Erase the newly created, delayed configuration file)

After this enter the appropreate commands as listed above including a new delay time (if desired). This can be done any time before the delay timer expires.

5 Installation and Testing

5.1 Startup Sequence

Make all physical connections as indicated below.
While power is off, make sure USB data storage memory stick is installed in ROC
Power on PC's and Mac
Power on RS232/485 converter
Configure communication connection for Mac to ROC
Power on PSB
Start up hyperterminal on PC connection to PSB
Start SolPSB software (see running SolPSB in section 7, and quick start for SolPSB)
Use SolPSB software to turn on power to the ROC
Use SolPSB software to make any configuration changes to the ROC
Use SolPSB to recover data from ROC to PSB and then from PSB to PC computer

5.2 Shutdown Sequence

Use SolPSB software to turn off power to the ROC
Quit SolPSB software
Power off PSB
quit hyperterminal program
quit Zterm program
Power off RS232/RS485 connection
Power off PC and Mac

5.3 Startup Sequence for ROC only test configuration (no PSB)

Make all physical connections as indicated below for ROC only test configuration.
Power connection to ROC is direct from trimble power adapter to ROC.
While power is off, make sure USB data storage memory stick is installed in ROC
Power on Mac
Start up Zterm on Mac and configure communication connection for Mac to ROC
Power on ROC
ssh to ROC through Zterm and continue as described below.

5.4 Shutdown Sequence for ROC only test configuration (no PSB)

From Zterm terminal connection to ROC
$ umount /mnt/cf (if /mnt/cf is not unmounted then there will be warning messages when the disk is remounted when it is started up again the next time)
$ shutdown -h now
Disconnect power to ROC
Close Zterm
Power off Mac

5.5 Physical connections from ROC

1) Serial connection from ROC Single Board Computer (SBC) to MAC or PC
This connection is used for:

- monitoring the SBC and ROC operations
- installing new ROC software
- only for testing not for flight

ROC: Console(Test) => 9 pin round test cable DB9
DB9 test cable => serial crossover cable (female-female)
DB9 serial crossover cable => CPUS03 serial adapter to USB
USB => computer (ie Mac running Zterm using)

2) Serial connection from ROC:SBC to PSB
This connection is used for

- PSB control of ROC instrument
- Data communication from ROC to PSB
- used for flight and testing using PSB

ROC: COMM => round DB 3 pin connector =>
3 strand brown,black, and white cable => Sub DB37 pin female connector: PSB:J10

3) DC power connection from ROC:SBC to PSB
This connection is used for powering the SBC which powers the GPS receivers
ROC: POWER => round DB 3 pin connector =>
2 strand black and red cable => Sub DB37 pin male connector: PSB:J1

4) RF connection from ROC to antennas
ROC: ANT2 : SMA female connector => SMA male: RF cable to antenna 2
ROC: ANT1 : SMA female connector => SMA male: RF cable to antenna 1

5) USB connection from ROC:SBC to USB data storage memory stick
This connection is used for

- storing recorded data on memory stick while waiting
- storing archived data
- Open cover of ROC:

single board computer (SBC) has four connectors on short end:
serial, 2 x USB, ethernet
SBC: USB => USB elbow connector
USB elbow => USB memory stick with 2 partitions for storing data
See below for configuring USB memory stick

5.6 Physical Connections from PSB to ROC and PC computer

Figure 3 General Diagram for PSB test configuration with ROC

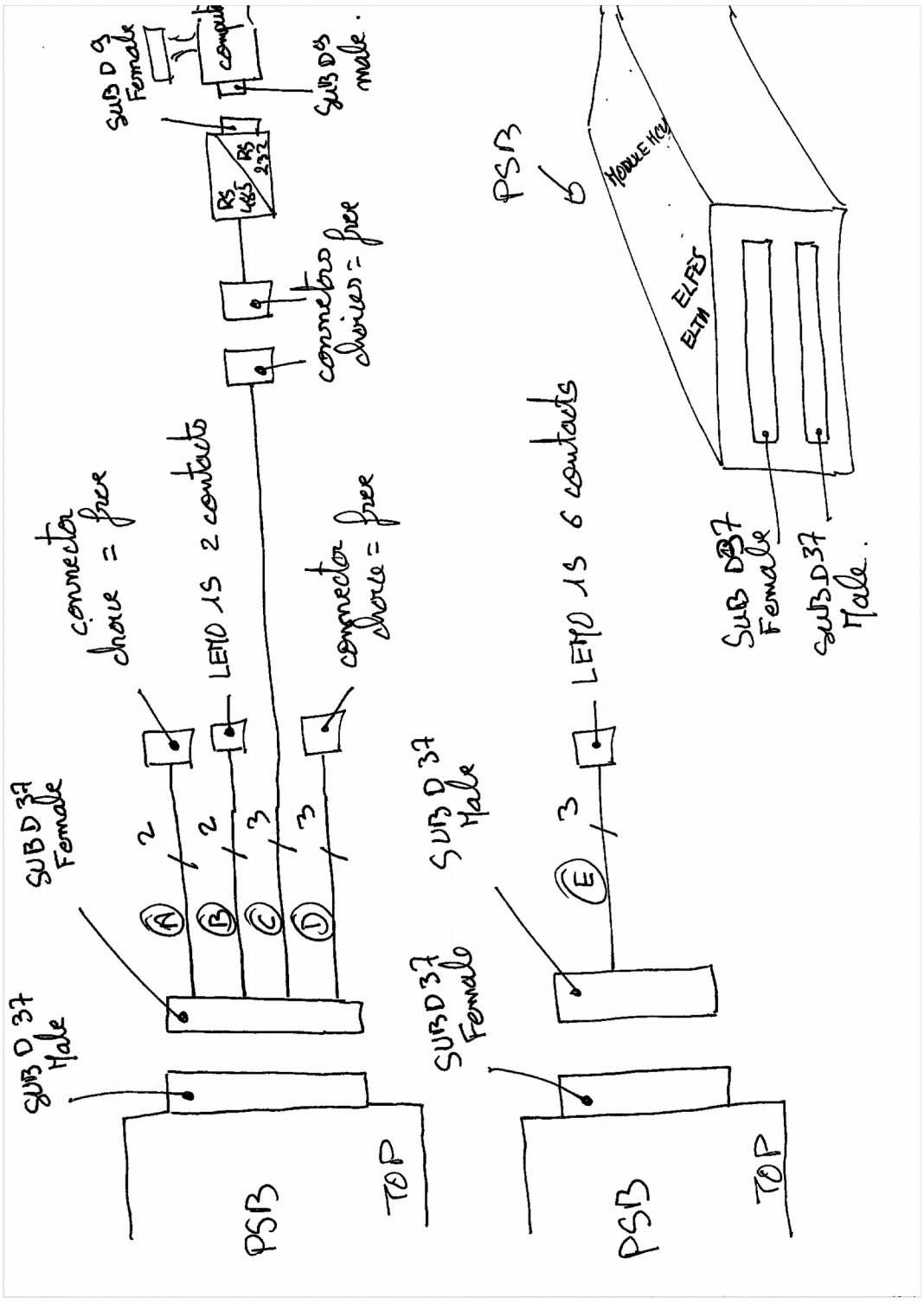

Figure 4 PSB connections

There are two connectors from one end of the PSB.

If the PSB is lying flat with the label "MODULE MCU" on the top and the serial number sticker on the bottom, call the top connector "top" and the bottom connector "bottom".

1) Serial connection from PSB to the ROC:SBC
See ROC Physical Connections above

2) Power connection from PSB to the ROC:SBC
See ROC Physical Connections above

3) Serial connection from PSB to PC (or ISBA) that is running the SOL PSB software
This connection is used for

- SOLPSB commands to control PSB
- SOLPSB commands to control ROC via PSB
- SOLPSB recovery of ROC data via PSB

PSB:J1 Sub DB37 pin female connector (bottom) =>
3 strand brown,black, and white cable => RS422/485/232 converter

4) Power connection from DC power supply to PSB
This connection is used for

- power to PSB
- power to ROC through

PSB:J1 Sub DB37 pin female connector (bottom) =>
2 strand black, orange cable => 15V DC power supply (12V minimum)

5) Serial connection from PSB monitor to PC hyperterminal
This connection is used for

- hyperterminal display of commands exchanged between PSBSOL and PSB
- hyperterminal configuration of the PSB

PSB: J1 three Sub DB37 pin female connector (bottom) (P79 in wiring diagram) =>
3 strand orange, black, red to DB9 female connector =>
DB9 male connector: (crossover??) serial cable : DB9 female connector =>
DB9 male connector: CP-US-OD serial to USB adapter =>
PC: USB port (USB port is simulating COM6)

6) Power connection from AC power supply to RS422/485/232 converter
This connection is sued for

- power to converter

RS422/485:barrel connector to 12V adapter to AC power supply

For further details, consult the wiring diagrams Figure 5 and Figure 6:
MX-3501N_20090625_144252.pdf and
MX-3501N_20090625_145846.pdf
Note that both the PSB at CNES and the PSB at Purdue had trouble connecting to the ROC until the TX+ and TX- wires were switched. Same is true on the PSB to RS485/RS232 adapter. It may indicate an inconsistency in wiring of the connector on the PSB.

you can put voltage probes on the TR/TX cables of the RS232 connection to monitor when the PSB sends a synchro or a TR/TM command (in the jargot TR/TM means a TC command that requests a TM packet)
You can tell when the synch packet goes by because it is just a single spike.

The TR/TM packet looks like a set of 3 spikes (it repeats the message twice if it gets no response)

In the lab here at CNES with the power supply that they have, they adjust the voltage to 15V 2 Amperes.

5.7 Configuration of communications connections for monitoring

1) serial connection from ROC:SBC to MAC
Open Zterm before powering on ROC
Use Zterm with the following settings:
Data Rate 115200; Data Bits 8; Stop Bits 1; Parity None; Flow control Xon/Xoff;

2) serial connection from PSB to PC using hyperterminal
Click desktop icon for netrs hyperterminal connection
file=> open => PSB MONITOR.ht => OPEN
connect using=> COM6
=> configure => 19200 bits per second =>Data bits 8 => Parity N=> Stop Bits 1 => Flow control Xon/Xoff => OK => OK
menu call=> connect
Activity of the PSB will be displayed.
For more information on using this connection to configure PSB see configuring PSB section below.

5.8 Optional physical connections

1) (optional) USB connection to USB memory stick with full linux operating system
This is used for

- running linux commands not on linux lite SBC flash memory
- to install new software releases

Open cover of ROC to replace USB data storage memory stick with USB linux memory stick:
single board computer (SBC) has four connectors on short end: serial, 2 x USB, ethernet
SBC: USB => USB elbow connector
USB elbow => USB memory stick with development operating system (full linux)

2) (Optional) IP connection from SBC of ROC to MAC or PC
This is an alternate method for transferring software updates to/from the SBC.
It requires that the full linux operating system memory stick is installed
Open cover of ROC:
single board computer (SBC) has four connectors on short end: serial, 2 x USB, ethernet
SBC:ethernet => ethernet crossover cable
ethernet crossover cable => Mac computer (Mac ssh to address of ROC)

3) (optional) Power connection directly from ROC:SBC to 12V power supply
This connection is to run ROC without PSB and without power provided from PSB
ROC: POWER => DB3 pin connector =>10cm power cable
10cm power cable => Trimble 18V power adapter

5.9 Preparation of data storage USB memory stick for operational mode

Get a regular USB memory stick (for example 500 Mbytes) (need to check total size of archive that you might want to store, in case the balloon is recovered)
Create two partitions on the USB stick.
Here is one way to do it using the SBC computer and a MAC. On the SBC computer:

fdisk /dev/scsi/host0/bus0/target0/lun0/disc
type m <return> for help
create new partition table
create new partition, primary, partition 1, first cylinder, about 1/4 of the number of cylinders on the disk.
create new partition, primary, partition 2, start with the next cylinder, go to the last cylinder on the disk
write new partition table
exit

put the memory stick in the MAC computer
Applications => Utilities => Disk Utility
Two disks will show up under the USB memory stick, disk1s1 and disk1s2 :
=> Select disk1s1
click on tab => erase
format => MSDOS VFAT
=> erase
=> select disk1s2
click on tabl => erase
format => MSDOS VFAT
=> erase
close Disk Utility
Eject USB memory stick

Now you have two partitions on this disk, one where the data will be archived.
If you fill up the partition where the data is archived, you can still access the other partition so there is space to do everything needed to communicate with the SBC.

5.10 Configuration of alternate communications connections

1) IP connection xover-to-roc
To use when connected directly from user computer to ROC via an ethernet crossover cable.
Ethernet settings on user computer:
Location: xover-to-roc (name of connection)
configure: Manually
IP Address: 192.168.0.2
Subnet mask :255.255.255.0
Router: (blank)
DNS Server: (blank)
Apply (not Connect!)

Ethernet settings on ROC SBC:
cd /etc/sysconfig
more network_cfg (to look at the file, vi to edit it):

```
###################################
### Technologic Systems
```

```
### General Network Configuration File
###
# if NETWORKING=yes, then IP connect starts on powerup
        NETWORKING=yes
# comment these 4 lines out for dhcp, uncomment for static
        GW_DEV=eth0
        GATEWAY=192.168.0.1
        HOSTNAME="ts7200"
        BOOTPROTO=static
# uncomment the 1 line below for dhcp, comment for static
#       BOOTPROTO=dhcp
        DEFRAG_IPV4=no
        FORWARD_IPV4=no
##################################
```

more ifcfg-eth0 (to look at the file, vi to edit it):

```
######################
DEVICE=eth0
# comment out 5 lines below for dhcp, uncomment for static
IPADDR=192.168.0.50
NETMASK=255.255.255.0
NETWORK=192.168.0.0
BROADCAST=192.168.0.255
BOOTPROTO=static
# uncomment 1 line below for dhcp, comment out for static
#BOOTPROTO=dhcp
ENABLE=yes
######################
```

open an Xwindow on user computer
check connection by typing:
ping 192.168.0.50
login to SBC computer:
ssh root@192.168.0.50
Password: !tlucco
$

2) IP connection Roc-chem
To use when the ROC is connected to IP network in Purdue JAFCI building. Actually we used this connection through the hub in Phil's lab after unplugging all the other computers, so it was actually a local network, because the user computer did not have an authorized IP address. In this case the user computer was a Mac.
Create a network connection on the user's computer with the following settings:
Location: Roc-chem (name of connection)
Configure: Manually
IP Address: 128.210.72.144
Subnet Mask: 255.255.255.0
Router: 128.210.72.1
DNS Server: 128.210.11.5
apply (not Connect!)

3) IP connection with DHCP
To use for connecting to a network so that Phil can access SBC remotely. Beware this has not been implemented successfully yet, and caused problems connecting to the ROC afterwards. See resetting IP address below.
Ethernet connections on user computer (TBC):
Location:
configure:
IP Address:
Subnet mask :
Router: (blank)
DNS Server: (blank)
Apply (not Connect!)

Ethernet settings on ROC SBC:
$ cd /etc/sysconfig
edit file network_cfg:

```
##################################
### Technologic Systems
### General Network Configuration File
###
# if NETWORKING=yes, then IP connect starts on powerup
        NETWORKING=yes
# comment these 4 lines out for dhcp, uncomment for static
#        GW_DEV=eth0
#        GATEWAY=192.168.0.1
#        HOSTNAME="ts7200"
#        BOOTPROTO=static
# uncomment the 1 line below for dhcp, comment for static
        BOOTPROTO=dhcp
        DEFRAG_IPV4=no
        FORWARD_IPV4=no
##################################
```

edit file ifcfg-eth0:

```
######################
DEVICE=eth0
# comment out 5 lines below for dhcp, uncomment for static
#IPADDR=192.168.0.50
#NETMASK=255.255.255.0
#NETWORK=192.168.0.0
#BROADCAST=192.168.0.255
#BOOTPROTO=static
# uncomment 1 line below for dhcp, comment out for static
BOOTPROTO=dhcp
ENABLE=yes
######################
```

5) setting automatic IP networking on startup
(Currently this is already set up and doesn't need to be repeated)
NETWORKING=yes and ENABLE=yes must be set in files as described above
$ cd /etc/rc.d/rc3.d
$ ln -s /etc/init.d/rc.inetd S20inetd

$ reboot
when it runs S20inetd on startup it runs rc.inetd which starts internet

5.10.1 Troubleshooting IP connection

1) If after reboot, the IP connection to the ROC is not responding, it is because after rebooting in production operation mode, the startup did not automatically start the INET server. Note that the ROC should run operationally without having the IP connection working in order to reduce power consumption.
You can start the inet server to get internet working manually by typing
/etc/init.d/rc.inetd start

2) if you can't get the IP connection and the hostname prompt is (none) then try this:
/etc/init.d/network start
/etc/init.d/rc.inetd restart

It will be fixed in the next package.
If you want a permanent fix until next package

cd /etc/rc.d/rc4.d
ln -s ../../init.d/network S05network

3) If you make a mistake with configuring the IP address and then on reboot it automatically tries to connect to an incorrect address that it can't find, it may go into an infinite loop waiting and not respond to commands even through the serial connection. In that case, try the following:

Cycle power on the ROC to reboot the SBC.
As soon as it powers up type CTRL-C over and over until it stops in boot loader mode.
You know you are successful if you see
RedBoot>
Type the following commands:
fis load vmlinux
exec -c "console=ttyAM0,115200 root=/dev/mtdblock1 2"
This will eventually boot to run level two to login.
Set your terminal type
export TERM=vt102 (or whatever terminal your z-modem program
emulates)
Fix the /etc/sysconfig/network_cfg and ifcfg-eth0 files with vi, the safest is to use the static IP address option above.
Type the following to reboot the SBC:
$ umount /mnt/cf
$ reboot

6 ROC software installation and testing

The mode that the ROC SBC is in depends on the environment variables and files that are present when the system reboots.

The order of events is the following:

Power on system
Boots up in linux lite environment
runs the files that are present in /etc/rc4.d
These files are linked to /etc/init.d
Therefore it runs the files /etc/init.d/S* files in numerical order
The S* files configure the system in different ways.

boots up /etc/rc4.d=>/etc/init.d/S* files contain: deploy.sh (export OCCULT_DEPLOY=TRUE) tries to mount /dev/scsi/.... /var/occult OCCULT_HOME=/var/occult /reset file is present looks for /var/occult/config_state copies default_config_state to config_state OCCULT_DEPLOY=TRUE starts operational deployment login prompt	boots up etc/rc4.d=>/etc/init.d/S* files contain: OCCULT_HOME=/var/occult /reset file is not present looks for /mnt/cf/ocult/config_state uses existing config_state OCCULT_DEPLOY=TRUE starts operational deployment login prompt	boots up etc/rc4.d=>/etc/init.d/S* files contain: tries to mount /dev/scsi/.... /mnt/cf OCCULT_HOME=/mnt/cf /reset file is not present uses existing config_state OCCULT_DEPLOY=" " does not start operational deploy login prompt

Starting from new USB Disk that has 2 partitions:
do_deploy creates directories on new USB Disk

6.1 The SBC full linux development vs linux lite operational environments

On power-up the Zterm screen will show:

Partition check:
/dev/scsi/host0/bus0/target0/lun0: p1 p2

It will initialize the two GPS receivers and print the GPS receiver info to the screen:

Receiver Info:
Serial #: 4636A72282
Type: BD950
NAV Process Version: 00214
Channels: 24
L1 Channels: 12
==========

GPS receiver 1 successfully initialized and reset

It will start up ROC software and start sending satellite status messages to the screen:

Starting up the occultation system modules
Running coordinate from repository At revision 263.
Running CNESserver from repository At revision 263.
Satellite 9/Receiver 2: state SEEN @20 to state POSITION @21 (rising)

Check what mode the receiver is in:
$ echo $OCCULT_DEPLOY (=TRUE if in deploy mode, = " " if not in automatic deploy mode)
$ echo $OCCULT_HOME (= /var if in deploy mode, =/mnt/cf if not in automatic deploy mode)

Check which disks are mounted
$ df

Filesystem	1k-blocks	Used	Available	Use%	Mounted on
/dev/root	128000	46728	81272	37%	/
/dev/scsi/host0/bus0/target0/lun0/part1	971176	411676	559500	42%	/mnt/cf

Check current configuration of the receivers:
$ changeConfig -d 1
Device #: 1
Last updated: Thu Jun 3 14:39:05 2010
Serial port: /dev/ttyTS0
Power on: no
Position Elevation threshold: -10
Occultation Elevation threshold: -10
Time Gap for lost satellite (seconds): 1800
Occultation sample rate: 5
Position/re-sample rate: 5
Elevation Mask: -10
Application Mode: Position
RT17 Record Format: Concise
Dataset Size: 4K

$ changeConfig -d 2
Device #: 2
Last updated: Thu Jun 3 15:11:16 2010
Serial port: /dev/tts/3
Power on: yes
Position Elevation threshold: -10
Occultation Elevation threshold: -10
Time Gap for lost satellite (seconds): 1800
Occultation sample rate: 5
Position/re-sample rate: 5
Elevation Mask: -10
Application Mode: Position
RT17 Record Format: Concise
Dataset Size: 4K
(set for no recording from receiver 1, positioning mode only for receiver 2)

To change the configuration, use the changeConfig command, then send a kill -USR2 signal to notify processes that the configuration has been changed.

$ changeConfig -d 2 -k 120

```
$ ps
PID  Uid     VmSize Stat Command
  150 root       188 S   coordinate
  151 root       536 S   sh /etc/init.d/watch
  152 root       692 S   -sh
  154 root       248 S   CNESserver
  156 root       380 S   arbitrator
  199 root       196 S   collect 2
$ kill -USR2 150
```

It will restart GPS receiver 2 with new configuration parameters.

To view the data that is being collected:
$ ls /var/occult/datasets (this is the data that is currently being collected)
2_01586_445610
(format N_GWEEK_SOW where N is receiver number)
(SOW is second of the week, when file started)

$ ls /var/occult/transmit
2_01586_403360 2_01586_403620 2_01586_445010 2_01586_445230
2_01586_403475 2_01586_403690 2_01586_445100 2_01586_445300
2_01586_403550 2_01586_444860 2_01586_445170 2_01586_445365
(these are the datasets that are completed and closed and are waiting for tramission to PSB)
(a positioning dataset is closed when it reaches the file size specified with the -k command)
(an occultation dataset is closed when several events have occurred: the occulting satellite has set, and the occultation has been determined to be the best out of the N_OCC that it is choosing from)

$ ls /var/archive
2_01586_400415 2_01586_400890 2_01586_403760
2_01586_400475 2_01586_400940 2_01586_444860
2_01586_400530 2_01586_400990 2_01586_445010
(these are the datasets that have been transmitted, then compressed and copied to the /var/archive directory, and removed from the /var/transmit directory)

To stop the ROC software:
$ rm /etc/init.d/deploy.sh
$ reboot
(will reboot without OCCULT_DEPLOY = TRUE, so will stop after login prompt)

To start the ROC software up again

$ cd /root/install
$ do_deploy
$ reboot

The SBC has a dual boot: full linux development environment and operational linux lite environment.

To use the full linux development env, the USB full linux memory stick must be connected to the SBC via the USB port.

Use the full linux development env for transferring and installing software, especially for zmodem file transfer, and for testing the ROC.
The terminal connection can be from either a zterm/hyperterminal serial connection or through an IP ssh terminal connection.

The operational environment has limited linux commands.
When running in the operational env, the full linux USB stick can be removed thus leaving the USB port free for the USB data storage memory stick for archiving data.
The terminal connection must be from a zterm/hyperterminal serial connection.
Use the operational mode for operational flights.

Installation and development	Testing ROC GPS operation	Operational deployment
boot full linux	boot linux lite	boot linux lite
USB = full linux mem stick	USB = full linux mem stick	USB= data storage mem stick
Prompt: (ts7200) login: #	Prompt: (none) login: # or $ (??)	Prompt: (none) login: $
zterm login	xterm login ssh root@192.168.0.2 zterm login	zterm terminal login
Programs, data, archive in:		
/root		/mnt/cf/root
/root/install ? /root/occult /root/archive	/root/install? /mnt/cf/occult /mnt/cf/archive	/root/install ? /root/occult ? /root/archive ?
		/var/occult? /var/archive?
Config files in:		
??	??	??
Startup files in		
/etc/sysconfig/ntwork_cfg /etc/init.d	??	??

The default on startup is that it uses the operational linux lite env
To go to the full linux development env:
you must type the commands in the Zterm window using the RS232 connection because it changes the IP address on the SBC when you change to the development env, and then you can't communicate anymore.

$ loadUSBModules.sh (this loads the USB drivers to the kernel)
(it makes the things in /dev/scsi.../part1 appear to be just in /)
$ loadUSB.sh (this changes to the development mode OS)

To transfer files to the USB memory stick:
rz --delay-startup 15
zterm menu=> File > send files => browse to files => okay
the file will be found in the home directory /root

[this is a note I have from a test we did, don't know what it does]
Look at resolve.conf
[end of unknown step]

To exit the development env and return to production env:
ctrl-D

The directory /root seen in full linux development environment can be accessed when in the operational environment by mounting the USB drive.
This needs to be mounted for running in testing mode.
$ mount /dev/scsi/host0/bus0/target0/lun0/part1 /mnt/cf
After that /mnt/cf/root (production env) = /root (development env)

6.2 Running ROC software in test mode from the full linux development env

Make sure power is off to ROC.
Insert USB full linux memory stick into ROC USB port.
Start Zterm on Mac.
Connect Trimble part 62546 18V power adapter to PWR port on ROC.
Power ROC on - light will quickly flash on SBC board
login to SBC:

(none) login: root
password: !tlucco

check you are indevelpment mode:

echo $OCCULT_HOME
$OCCULT_HOME = /mnt/cf/occult (full linux development mode)
$OCCULT_HOME = /var/occult (linux light operational mode)

Remove parameter file device_address.sh to see prompts to reset environment variables

$ cd install
$ rm init.d/device_address.sh (removes env variables that set ROC address and FlightID, etc)
$ sh install.sh
What is the decimal device address of the ROC for this flight? 09
What is the decimal flight ID for this flight? 43
Default number of occultations to consider before selection 1
Adjust system time on synchronization frame (y/n)? n
File system % usage limit for stopping data collection (default 55) 55
File system % usage limit for restarting data collection (default 10) 10

The receiver will print the following messages:

Starting installation ....
Detected a saved network configuration already present
==========
Directories for binaries already exists
==========
Installed binaries
==========
Install boot up scripts
==========
Removing all previous symbolics link in at run level 4
Creating new links...
S05network

```
S10rc.initd
S11portmap
S50initdevices
S50mount_data
S65archive
S75reset
S90ts7kvfb
S99occult
Created symbolic links for run level 4
==========================================================
USB development environment already mounted
==========================================================
/mnt/cf/occult already exists
/mnt/cf/archive already exists
Finished creating mount points
==========================================================
Finished creating data directories
copied start file for loading USB modules
copied the run level 4 inittab succesfully
Copied system wide-shell profile to /etc
Successfully updated system files
==========================================================
Successfully created configuraton files
Installation complete
```

Set the ROC recording parameters the way you would like them for device 1 (port /dev/ttyTS0) and device 2 (port /dev/tts/3):

```
sh config_receiver.sh 1 /dev/ttyTS0
Building default configuration for receiver 1 at serial port /dev/ttyTS0
Position elevation threshold (integer degrees) [default is 15]: -10
Occultation elevation threshold (integer degrees) [default is 5]: -10
Gap used to declare satellite set (integer seconds) [default is 300]: 1800
Occultation sample rate (integer seconds) [default is 2]: 5
Position/resample rate (integer seconds) [default is 30]: 5
Dataset size for position application (integer KB) [default is 4]: 120
Application Occultation/Position (O / P) [default is O]: P
Receiver power enabled (Y/N) [default is Y]: Y
RT17 record format (Expanded/Concise) [default is Concise]:Concise

sh cofinfig_receiver.sh 2 /dev/tts/3
```

Start the ROC software:

```
$ coordinate &
```

To stop the test operation type

```
$tst   (may need several times or a few seconds to stop)
```

Reboot if things go amiss to start over again:

```
$ umount  /mnt/cf
$ reboot
```

6.3 Testing communication between ROC and PSB without GPS receivers

To test the SBC to PSB communication without running the data acquisition
(useful when there is no antenna connected):

Change the configuration file for the ROC to turn off GPS receivers:

changeConfig -d 1 -P 0
changeConfig -d 2 -P 0

Create a set of text files and put them in the directory
/mnt/cf/occult/transmit
vi create_test_files.sh

Run the coordinate program in background:
$ coordinate &

It will run the following:
coordinator
CNESserver
arbitrator
collect

It will send the files in the /mnt/cf/ocult/transmit directory to the PSB.
They can be decoded and then compared against the original files to make sure they are the same.

6.4 Running the ROC software in operational mode

In operatonal mode, the system expects that a thumb drive with two partitions be installed with vfat file systems[1]. One partition is for temporary storage as data is processed, filtered and transmitted. The other is for archiving the data sets. If the archive becomes full, the system ignores the archiving errors and continues its normal processing as files in the temporary storage partition is re-cycled. The system can fail however, if there is insufficient space to hold all the data being held to determine the best occultation as well as the files to be transmitted in the first partition.

The system also starts the collection software as the system finishes booting in deployment mode.

To place the ROC in deployment mode first shutdown the system and swap thumb drives:

$ shutdown –h now

After to the system gives the halted message, power off the system and swap the thumb drives.

Power up the system. Some startup scripts will recognize the old thumb drive is not there and will refuse to execute. That is by design. After bootup, login and switch to the install directory.

[1] The vfat file system is less efficient but more robust for this type of application. If the ROC is powered off before the system dismounts the file systems on shutdown, they are much more likely to survive.

$ cd install

Then issue the deployment command:

$ sh do_deploy

This runs a script which asks for the configuration of the receivers:
building config file for receiver 1:
Elevation threshold: 10 degrees
Gap used to declare satellite set: 300 s
Occultation Sample Rate: 2 sec
Position resample rate: 30 sec
Dataset size for the position application: 4 kb
Application Occultation or Positioning: O
receiver power enabled: Y

building config file for receiver 2:
Elevation threshold: 10 degrees
Gap used to declare satellite set: 300 s
Occultation Sample Rate: 2 sec
Position resample rate: 30 sec
Dataset size for the position application: 4 kb
Application Occultation or Positioning: O
receiver power enabled: Y

Messages from the different system modules will start appearing on the console on the next reboot. (The terminal characteristics are not set to read the messages easily. If you log in, the messages will be more easily read.)

When in operational mode, these are the differences:
it doesn't start networking via IP, it turns off ethernet
after

To get back into testing mode after deployment mode:
---
while it is running:
(none) login: root
password: !tlucco
rm /etc/init.d/deploy.sh
tst (terminate and shutdown because in deployment mode it shuts down too)
power off
replace archive USB with full linux USB
reboot
(ts7200) login: root
password: !tlucco
cd install
sh do_test. (restores network config files and test flags in program config files)

6.5 Onboard data storage locations

TBC 4 File structure on SBC

Install - set environment parameters

Coordinate - create config file in */*

Run collect - write gps data to dataset directory in */* (how many files)

Run arbitrate - read gps data, select data, write events* or segments* in transmit directory

Run CNES-protocol - read segments in transmit directory */*, put in CNES packets, transmit to PSB

Archive - Store data in the */* directory

Installation Mode	Testing Mode	Deployment Mode
boot full linux	boot linux lite	boot linux lite
USB = full linux mem stick	USB = full linux mem stick	USB=vfat archive mem stick
Prompt: (ts7200) login: #	Prompt: (none) login: $	Prompt: (none) login: $
zterm login	xterm login ssh root@192.168.0.2 zterm login	

6.6 Installing the software

Operational linux lite env: (wait! can you use IP connection when in operational linux lite mode?)

From user computer, use ftp to copy the occult.tar file to /root (prod env)

```
> ftp root@192.168.0.50
Password: !tlucco
cd /root
put occult.tar
```

```
on SBC:
```

$ loadUSBModules.sh (if you haven't already loaded the drives in previous steps)
$ mount /dev/scsi/host0/bus0/target0/lun0/part1 /mnt/cf
$ cd /root
$ tar –xvf occult.tar
$ cd install
$ rm init.d/device_address.sh (if you want it to prompt you to enter flightID)
$ sh install.sh

or

Full linux development env:

Start Zterm
Install USB full linux development memory stick
connect Trimble Part 62546 18V power adapter to PWR port on ROC
Light quickly flashes on SBC board

SBC reboots
SN 4636A72282
GPS reeiver 1 succesfully initialized and reset
SN 4835A2853
GPS receiver 2 successfully initialized and reset
No archive directory specified for archiving
Clearing old dataset from processing area
TS-LUNIX/arm 70
(none) login: root
password: !tlucco
default configuration file is /mnt/cf/occult/default_config_state
unprocessed datasets are in /mnt/cf/occult/datasets
Data segments to be transmitted are in /mnt/cf/occult/transmit

Use zmodem to transfer the occult.tar file to /root (dev env)
exit Development env with ctrl-D
$ mount /dev/scsi/host0/bus0/target0/lun0/part1 /mnt/cf
$ cd /root
$ tar –xvf /mnt/cf/root/occult.tar
$ cd install
$ rm init.d/device_address.sh (if you want it to prompt you to enter flight number)
$ sh install.sh

The install.sh script will prompt you for the address of the ROC and the flight number for this balloon if it has not already been specified in a previous installation:

> What is the decimal device address of the ROC for this flight? 09
> What is the decimal flight ID for this flight? 42
> Initial number of occultations to consider before selection? 5 (to be confirmed)
> Adjust system time on synchronization frame (y/n)? n (for now we'll say no)
starting installation ....

It will write the address to the file /root/install/init.d/device_address.sh:
export CNES_ASSIGNED_ADDRESS=09
export CNES_FLIGHT_ID=42
export OCCULT_SELECT_COUNT=5
export ADJUST_TIME_ON_SYNC=NO
Remove the file to change the address for a different flight

The install script will prompt you for the initial default configuration for data collection for each receiver if it does not already find a default configuration file:
/mnt/cf/occult/default_config_state :

Building default configuration for receiver 1 at serial port /dev/ttyTS0
Elevation threshold (integer degrees) [default is 15]: -10
Gap used to declare satellite set (integer seconds) [default is 300]:
Occultation sample rate (integer seconds) [default is 2]:
Position/resample rate (integer seconds) [default is 16]: 30
Dataset size for position application (integer KB) [default is 4]:
Application Occultation/Position (O / P) [default is O]: P
Receiver power enabled (Y/N) [default is Y]: Y

Initial number of occultations to consider before selection
Use sync packet for adjusting time (y/n)

Building default configuration for receiver 2 at serial port /dev/tts/3
Elevation threshold (integer degrees) [default is 15]: -10
Gap used to declare satellite set (integer seconds) [default is 300]:
Occultation sample rate (integer seconds) [default is 2]:
Position/resample rate (integer seconds) [default is 16]: 30
Dataset size for position application (integer KB) [default is 4]:
Application Occultation/Position (O / P) [default is O]: P
Receiver power enabled (Y/N) [default is Y]: Y

The production environment has two modes, testing and deployment.

For testing mode:
It is assumed that the development thumb drive is installed.
The data directories for the programs in testing mode are in
/mnt/cf /occult
/mnt/cf/archive
Reboot to run in the production env testing mode
Should always leave modem console window open to monitor messages from ROC
$ umount /mnt/cf
$reboot

It will close the IP connection
After reboot the SBC will restart
Monitoring the console test serial interface will display that the GPS receivers have been successfully initialized and reset.
It will prompt for login on the serial interface
login: root
Password: !tlucco
It will display the following:
Default Configuration file is /mnt/cf/occult/default_config_state
Configuration file is /mnt/cf/occult/config_state
Unproceesed datasets are in /mnt/cf/occult/datasets
Data segments to be transmitted are in /mnt/cf/occult/transmit

6.7 Testing

When testing, the running modules will produce messages to the console.

Messages beginning with “Protocol” indicate they are coming from the CNES protocol server. They indicate the events and state changes as described in the CNES Protocol Server Architecture document.

Messages starting with “Satellite” or “Receiver” come from the collect module. The collect module writes position and occultation records to a dataset according to the state of the satellite. The messages describe the changes of state or elevation of a given satellite. The states names as printed (they differ than how they are named in the code) are

NOT SEEN – the satellite has not recently been listed in a data packet delivered by the receiver.
STABILIZING – the satellite has just been detected, packets are discarded for 1 minute to allow elevation/azimuth to settle. They seem to vary when they are first seen.
SEEN – the satellites have stabilized and now the module is trying to determine if they are rising or setting.
POSITION – the satellite's data is being recorded at the resampling rate in occultation mode or the only sampling rate for the position mode
OCCULT – the satellite's data is being recorded at the high/occultation rate.
RISING – this occurs in position mode when the satellite is rising but not above the elevation threshold. The satellite's data is not being recorded.
IGNORED – the satellite's data is not being recorded. This occurs in the position mode when the satellite is setting and is below the elevation threshold. This also occurs at start up in occultation mode for satellites initially seen setting as below the elevation threshold.
INSANE – if a satellite is detected from going from a setting state to a rising state it is marked insane. Any satellite using this for calibration is set to having no calibration satellite.

One should expect a "Receiver not responding" message from the collect module for the first several second of the collection process as the receiver is initializing itself with the new application settings just received. If it continues for more than a minute, the connection to the device and receiver health should be investigated.

Messages starting with "Event" are from the arbitrator module. Events from the collect module are passed to the arbitrator module that determines what occultations are selected. Each event is associated with a specific location of a file (dataset) collected from the collect module. The events are queued up to be processed appropriately and the data in the dataset are written or discarded up to that location in the dataset. The messages indicate which event is to be processed next. The message "Event removed" indicates that the message at the head of the queue has been processed.

NEW DATASET

The collect module has opened a new file in which to record satellite data. This event is processed after all the events associated with previous datasets have been processed.

CLOSE DATASET

The collect module has closed the dataset. When this shows at the head of the queue, it indicates that all other events have been process for the dataset and all valid records in the dataset can be processed for transmission.

OCCULTATION START

This event means that an occultation has begun at this point in the dataset. The datasets record cannot be processed until this event has been marked as being chosen or not chosen for transmission. Once it is marked as such, the records in the dataset up to the next event are transmitted as appropriate. Until the event is marked as chosen or not chosen, it remains at the head of the queue and nothing is written. Until enough occultations have been collected to make a selection, this will remain at the top of the queue and no other data can be processed. That number can be changed with a TC TEXT command.

END SATELLITE

As this event is processed, it marks the associated satellite and it's calibrator as no longer needing its occultation records to be transmitted.

6.8 Changing configuration locally

Use the changeConfig program to configure the system locally

This program is used to initialize and change the configuration used by the collection programs (collect). This is mainly used for installation and development.
The parameters are initially set when the ROC software is installed (see above).

Some of these are settable when the ROC is in flight using the remote telecommands from the ground station using CNES TC TEXT commands.
System modules that are already running are not notified when the configuration files are changed locally using changeConfig, therefore stop the coordinate progam and restart for the changes to take effect.

changeConfig [Command] [Options] [File]

Where [Command] is one of

-n # - create a configuration file for this # of receivers
-d # - list/change the configuration for this receiver (first one is 1)

[File] is a valid system filename. If not present, the environment variable CONFIG_STATE_FILE is sought and used if present or /var/occult/config_state_file if not.

[Options] is zero or more of the following, no options just lists the configuration for command –d.

-P 0 The receiver should be powered off and the collection program not run
-P 1 The receiver should be powered on and the collection program should be run

-m O (capital letter O) sets the receiver to occultation application mode
-m P sets the receiver to position application mode

-p device – sets the serial port connecting the receiver to the system

-g seconds – sets the number of seconds the collection program use to assume a satellite has set if no reading is seen from it

-k n - sets the dataset size in position mode before transmission begins. The n is measured in KB.

-r n - sampling rate for position, this is used for non-occulting satellite in occultation mode and for all satellites in position mode. he n is a number specifying seconds per raw data record. This will be rounded to a multiple of the occultation rate.

-o n – sampling rate for occulting satellites. This must be one that is supported by the receiver. The n is a number specifying seconds per raw data record.

to look at existing configuration files:
changeConfig -d 1
changeConfig -d 2

6.9 Installation (Alexandria Version)

This installation manual assumes the ROC is currently in deploy mode, If you are unsure of what mode the ROC is in do the following:

$ echo $OCCULT_DEPLOY
If the the system responds with TRUE are you in deploy mode

If the system is in deploy mode:
$ rm /etc/init.d/deploy.sh (removes exported variable OCCULT_DEPLOY=TRUE)
$ reboot
while rebooting, replace USB data memory stick with USB stick containing occult.tar.

$ login: root
$ password: !tlucco

$ echo $OCCULT_DEPLOY (the response should be blank)

After coming out of deploy mode or if the ROC was not in deploy mode initially:
$ df (to see what devices are mounted)
Unmount any mounted devices besides /dev/root that appear, that is type 'umount'
followed by what appears in the last column of what got printed when df was called.

$ cd /mnt/cf/occult
If this directory does not exist, create it by doing the following:
$ cd /mnt/cf
$ mkdir occult

$ mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part1 /mnt/cf/occult
If this doesn't work the first time try umounting and remounting the flast drive, it may
help to also check the actual location of the flash drive as it may be different from that
listed above. To do this:
$ cd /dev/scsi and tab until you can't anymore then add /part1 to the end of the
address followed by /mnt/cf/occult
$ cd /mnt/cf/occult (to check that it contains what you'd expect)
$ cp /mnt/cf/occult/occult-rev-###.tar /root
The current version is 265. If you put the .tar file inside another file on your flash drive
then you would enter this into the file location path. Note that copying will take a while
$ cd /root
$ tar -xvf occult-rev-###.tar
$ cd install
$ sh install.sh
The installation procedure install.sh first looks for a system configuration in
./init.d/device_address.sh

If a current configuration is present you will be prompted with something like the following:
Here is your current configuration:

export CNES_ASSIGNED_ADDRESS=09
export CNES_FLIGHT_ID=43

export OCCULT_SELECT_COUNT=2
export ADJUST_TIME_ON_SYNC=NO
export COLLECTION_STOP_LIMIT=55
export COLLECTION_RESTART_LIMIT=10

$ Would you like to replace the configuration (y/n)?

If you would like to change the configuration or if it is non-exisitant you will be prompted for the following:

- Device address: enter the decimal representation of the CNES-assigned address for the ROC. This is absolutely required for ROC operations. ROC is assigned address 09
- Flight ID: enter the decimal representation of the CNES flight ID. This is absolutely required for ROC operation. ROC-1 has FlightID=43; ROC-2 has FlightID=42
- Occultation Count: The initial number of occultations to be considered before choosing one for transmission. This can be overridden with a TC TEXT command. If this parameter does not appear as an installation parameter and is not overwritten, a compile-time value of 5 is used.
- Adjust system time with sync: This should be answered no. It is no longer relevant now that coordinate gets it's time information directly from the GPS messages..
- Collection stop limit: Enter an integer as a percent of the data partition used to trigger the coordinate program to stop data collection. If this is not present, a compile-time value of 55 is used.
- Collection restart limit: Enter an integer as a percent of data partition used that will trigger a restart of data collection caused by reaching the collection stop limit. If not specified in the installation parameters, a compile-time default of 10 is used.

Installation will begin upon completion of inserting new configuration if changed or upon deciding not to change the configuration

$ changeConfig -d 1 (to check the configurations in /mnt/cf/occult/config_state)
$ changeConfig -d 2

You should begin with the default configurations which are as follows:

Device #: 1
Last updated: Fri Jun 11 18:54:32 2010 << Your date will not be the same
Serial port: /dev/ttyTS0
Power on: yes
Position Elevation threshold: 20
Occultation Elevation threshold: 0
Time Gap for lost satellite (seconds): 1800
Occultation sample rate: 30
Position/re-sample rate: 30
Elevation Mask: -10
Application Mode: Position
RT17 Record Format: Concise
Dataset Size: 120K

Device #: 2
Last updated: Fri Jun 11 18:54:32 2010
Serial port: /dev/tts/3
Power on: no

Position Elevation threshold: 20
Occultation Elevation threshold: 0
Time Gap for lost satellite (seconds): 1800
Occultation sample rate: 30
Position/re-sample rate: 30
Elevation Mask: -10
Application Mode: Position
RT17 Record Format: Concise
Dataset Size: 120K

If not in the default mode you can change to the default mode by:

$ cp /mnt/cf/occult/default_config_state /mnt/cf/occult/config_state

If the default state is incorrect it can be reset via two methods. The first method is as follows:

$ changeConfig -d 1 -P 1 -m P -g 1800 -k 120 -o 30 -r 30 -u 20 -l 0
/mnt/cf/occult/default_config_state
$ changeConfig -d 2 -P 0 -m P -g 1800 -k 120 -o 30 -r 30 -u 20 -l 0
/mnt/cf/occult/default_config_state

Please note that we had some trouble with this method of changing the default configuration at Purdue, if you experience problems as well try the second method

The second methods is as follows:

$ cd /mnt/cf/occult
$ rm default_config_state
$ rm config_state

$ umount /mnt/cf/occult
$ reboot

you will be told that the flash drive could not be mounted and that is okay

$ cd /root
$ cd install
$ sh do_deploy

If the default configuration was reset via method two you will be prompted with the following for each receiver, after which the default_config_state file will be built from your responses:

Position elevation threshold (integer degrees) [default is 15]: 20 << what should be entered
Occultation elevation threshold (integer degrees) [default is 5]: 0
Gap used to declare satellite set (integer seconds) [default is 1800]: 1800
Occultation sample rate (integer seconds) [default is 2]: 30
Position/resample rate (integer seconds) [default is 30]: 30
Dataset size for position application (integer KB) [default is 4]: 120
Application Occultation/Position (O / P) [default is O]: P
Receiver power enabled (Y/N) [default is Y]: y for receiver 1, n for receiver 2
RT17 record format (Expanded/Concise) [default is Concise]: C

If you start out with a configuration file or after creating the new default you will be prompted with:

disable receiver 1 [y/n] (n) : n
disable receiver 2 [y/n] (n) : y

If you'd like to re-check which receiver you disabled in this step, after rebooting do the following:

$ cd /root/install

$ vi deploy.sh

If a receiver is labeled with DISABLED after it, that means you disabled the receiver in this step. Note that when this happens the answer to 'Power on:' in the default_config_state file is automatically changed to 'no', but will not automatically change to 'yes' if the power is not disabled.

$ reboot

This time when the system starts up it should be in deploy mode and ready to go. /var/occult and /var/archive directories should be mounted automatically, and you should be able to access the changeConfig to review the receiver configurations being used.

---

touch /reset (/reset is a file that says to go back to the default configuration next time it boots up)
when you reboot it uses config_state as the configuration to start up coordinate.
if /reset is present, then it will copy what is in default_config_state to config_state, then use config_state
We should set default_config_state to have the parameters that are the most often used operational configuration.

changeConfig will change the file config_state
kill -USR2 [coordinate process_id] (will send a signal to all processes that config_state has been modified, then all processes will use the new config_state, it restarts the GPS receiver so phase tracking restarts)

There are three types of USB memory sticks
USB full linux memory stick: 1 partition
USB memory stick containing occult.tar : 1 partition
USB data memory stick: 2 partitions, configured for operational deployment mode.

regardless of the disk, on bootup the disk will show up as a device
/dev/scsi.../part1
and for the USB data memory stick, there will also be a device
/dev/scsi... part2

the startup commands mount this disk as /mnt/cf
loadUSB makes it so the top level directory appears as /

The production environment has two modes, testing and deployment. In testing mode, it is assumed that the development thumb drive is installed. The data directories for the programs in testing mode are in /mnt/cf (occult and archive directories) The system is in testing mode after installation and reboot. The USB modules will be loaded and the development thumb drive mounted at boot time. (To go to the development system, just type

$ loadUSB.sh

CTRL-d drops out of the development environment. In most cases the development environment is not needed beyond the z-modem transfer.

Rebooting and logging back in should set everything up to test. All modules are started by the coordinator. Start the coordinator in the background allows you to type other commands interactively.

$ coordinate &

The terminate system test command should stop the all the system modules if all goes well.

If the
$ tst

(You can always reboot if things should go amiss.)

$ umount /mnt/cf
$ reboot

In deployment mode, the system expects that a thumb drive with two partitions be installed with vfat file systems[2]. One partition is for temporary storage as data is processed, filtered and transmitted. The other is for archiving the data sets. If the archive becomes full, the system ignores the archiving errors and continues its normal processing as files in the temporary storage partition is re-cycled. The system can fail however, if there is insufficient space to hold all the data being held to determine the best occultaion as well as the files to be transmitted in the first partition. The system also starts the collection software as the system finishes booting in deployment mode. To place the ROC in deployment mode first shutdown the system and swap thumb drives:

$ shutdown –h now

After to the system gives the halted message, power off the system and swap the thumb drives.

Power up the system. Some startup scripts will recognize the old thumb drive is not there and will refuse to execute. That is by design. After bootup, login and switch to the install directory.

$ cd install

Then issue the deployment command:

$ sh do_deploy

Messages from the different system modules will start appearing on the console on the next reboot. (The terminal characteristics are not set to read the messages easily. If you log in, the messages will be more easily read.)

The file occult.tar has to be transferred to the ROC for deployment of the software. This can be via scp(?) if the network is enabled or through Z-modem. The Z-modem software is available only in the development environment. If the development thumb drive is installed, you may go to that environment with the command

$ loadUSBModules.sh
(The $ is the prompt you get when the system is ready for a command. This command causes the system to make asynchronously writes info to the console, so it is not clear that this is complete, typing a return should get you a prompt to be sure) Then type

[2] The vfat file system is less efficient but more robust for this type of application. If the ROC is powered off before the system dismounts the file systems on shutdown, they are much more likely to survive.

$ loadUSB.sh

To start a Z-modem transfer use the rz program

$ rz --delay-startup 15

The startup delay allows for time to get your Z-modem terminal emulator to start sending the file.

After transferring the file typing CTRL-D will return you to the production environment. You will need to be able to access the file on the development thumb drive if you used the z-modem method:

$ mount /dev/scsi/host0/bus0/target0/lun0/part1 /mnt/cf

If the tar file was transferred to /root in the development environment, you will find it on /mnt/cf/root in the production environment. Making that assumption, you are ready to install the software (change /mnt/cf/root to the proper location if occult.tar is not there). However, the development environment must be available as /mnt/cf since it is used in testing mode (see below). . The installation will attempt to mount it if it cannot find normally occurring files in the mounted file system.

7 SolPSB Interface Software

7.1 PSB Software installation

Download the SOLPSB software here:
https://echange.elta.fr/CNES
retrieve the directory "InstallSOLPSBV2_2.8.0.0" and SolPSB_V2.exe
login: dezen1
Password: cnes_08ELTA
use setup.exe to install the software
install in the directory Program Files/ELTA/SOLPSBV2

Installation of 3 versions of PSBSOL:
create a shortcut without anything = classic operational mode
create a shortcut with the words -modeExpertPSB after the executable name
create a shortcut with the words -modeDirect9600 after the name of the executable.

PSBSOL... (the one they use operationally)

PSBSOL... -modeExpertPSB (only use this one once at the beginning to set up the type of flight configured as PSB seul = PSB only; otherwise PSB seul will not appear in the choices in the drop down menu for PSBSOL... -modeDirect9600)

PSBSOL... -modeDirect9600 (the one we use to connect the PC directly to the PSB This will start the program in the mode where it uses the PC to simulate the ISBA messages and communicate directly with the PSB without having to pass through the ISBA. This is the mode we want.)

Note that none of the versions after version 2.8 support the PC simulation of ISBA.

7.2 Setting up the first flight

The first time you run SolPSB you have to configure the menus so that they know what type of flight you have.
You need to know the FlightID programmed into the PSB before you start.
If not known see "checking PSB FlightID" section below.

open a MS-DOS command window
cd c:\"Program Files"\ELTA\Setup_SolPSB_V2_8_0_0
run:
SolPSB_V2.exe -modeExpertPSB
=>Fichier = File => NouveauVol = NewFlight
It opens the window Nouveau Vol=New Flight window
click on nouveau=new
It opens the window configuration de type de vol=configuration of type of flight
=> click on nouveau =new
Nom du type = Name of type => PSB seul = PSB only
dropdown menu NSO:
PSB 0 (says we don't have a PSB controller in the NSO in this simulation)
MER 0 (says we don't have a MER energy controller in the NSO in this simulation)
dropdown menu NCU:
PSB 1 (says we do have a PSB controller in the NCU payload in this simulation)
MER 0 (says we don't have a MER energy controller in the NCU payload in this sim)
=> enregistrer=save
=> okay

Return to the window Nouveau Vol = new Flight
now we see PSB seul in the menu of window NouveuVol
Choose PSB seul in drop down menu
section of PSB window:
configurer
Configuration version de vol
click on nouveau=new
Version du vol => PSC2
Adress PSB 4
Type SCI TSEN
Adress SCI 1 => 6
Type SCI 2 => LMDOZ
Adress SCI 2 => 7
Type SCI 3 => ROC
Adress SCI 3 => 9
=> enregistrer
dropdown menu => PSC2
=> configuration PSB
new window configuartion PSB
unclick all the boxes for temperature
unclick all the boxes for tension=voltage
Relais 1 => PSB heater => evenement T
Relais 2 => TSEN => evenement Date
Relais 3 => LMDOZ => evenement Date
Relais 4 => ROC => evenement Date
unclick Relais 5

Relais 6 => ambient heater => evenement T
unclick all the CO 1 2 3
=> okay
=> okay (in the configuration version du vol window)
Now we see PSC2 as a configuration for NCU in the window NouveauVol

Now finish creating the flight
A=annee=year=09
V=vol=flight=03 or whatever
N=nacelle = flightid=42 (43 for the ROC at Purdue)
=> okay
return to the display window
everything is gray and can't modify anything
exit PSBSOL mode expert

7.3 Running the PSB Software

Now to run the SolPSB to start controlling the PSB, exit the expert mode and start up in Direct mode.
You need to know the FlightID programmed into the PSB before you start.
If not known see "checking PSB FlightID" section below.

cd c:\"Program Files"\ELTA\Setup_SolPSB_V2_8_0_0
run:
SolPSB_V2.exe -modeDirect9600
The Vol (flight) exists already. Now we can see PSBseul in the menu.

Create a flight and flight data directory:
Fichier=File => NouveauVol=NewFlight (create a new Vol)

The terminology here is confusing.
The flightID is programmed into the PSB because it will remain unchanged for that piece of hardware and will be programmed into each data frame.
It is misleading to call it a "flightID" because it is really the Nacelle ID=payload ID.
The "Vol" (translated to "flight") is usually assigned consecutively as the balloons are launched.
For example the payload that has the PSB with flightID = 42 might be the 3rd balloon launched in the campaign.
One would create with the software a new "Vol" with the following:
Vol = 03, flightID=42 and N=42.

After you choose NouveauVol=NewFlight a window opens.
Type de Vol = Type of Flight
From the dropdown list, choose:
PSB seul = PSB only (for testing mode where you are using the PSB without ISBA)
A = Annee=Year (put in YY here, ie 09)
V = Vol=Flight (Careful! put in Vol=03, NOT the flightID of the payload, for testing it can be any number)
N = Nacelle=payload =FlightID (enter the flightID of the payload, in this case 42, if this number is not what has been programmed into the PSB hardware then you will not be able to communicate with the PSB)
Date = today's date (it may do this for you based on PCtime?)
Numéro Com=Com Port = com port on PC where PSB is connected

Lieu=Place = location where launch is taking place - not used here.

PSB (lower window - in test mode, you can only access this part):
from the drop down menu choose:
Version du vol =Version of flight => PSC 2
(PSC 2 means the configuration of balloon where one ROC is flying with a TSEN and an LMDOz instrument)
Then you get the addresses for each instrument (ROC=SCI3 = address 9)
=> OK

It creates a folder for the flight in:
c:\"Program Files"/ELTA/Setup_SolPSB_V2_8_0_0/PSBxxVyyNzzA
Where xx is the year YY, yy is the "Vol" = flight, and zz=Nacelle=payload=FlightID
for example:
PSB09V04N42A

It opens the flight window.
See section below for a Flight Window description and interaction with the PSB.

Once you have created a "Vol", if you close the program and want to get your flight back to monitor it again later or download more data to it:
Menu => Vols=Flights (choose a flight that has been set up) ie. PSB09V04N42A
Note here that the 42 is the FlightID, the two digits after V (in this case 04) have a different meaning.

7.4 Quickstart to interrogate the PSB:

Start the ISBA simulator sending 1frame per minute simulated GPS time and location to PSB:
simulateur ISBA => 1 trame/minute radio button
Ordre TM =Send a TM command (This requests the housekeeping information)
you should see time updated (heure) and temperatures updated (?)
This indicates the PC => PSB connection is working well

Click on R4 tab (upper right) to access relay 4 which powers on ROC
Click on Forcé ON=Force ON. Turns on the power from PSB to ROC at next minute.
You will hear the PSB click when the relay goes on.
The ROC will power up. You will see the bootup process messages on the Zterm screen
The ROC will automatically start saving data internally to /var/occult/datasets.
When it has finished accumulating a dataset of defined size, it will move it to /var/occult/transmit.
Now when the PSB sends requests to the ROC, the ROC will start sending data to the PSB.
Data is transferred at one frame of 1028 bytes (995 bytes useful science data) per minute.

To interrogate the PSB to see how much ROC data it has, first turn off the 1 frame/min so there is no interference between data transmission from PSB to PC with the simulated ISBA message 1 frame/minute from the PC to the PSB

Right after startup check to see what the current counter number is:
Set Curseur=0 Taille=1 => click transfert
It will report the current frame counter position.

Turn on 1 frame/min so that ROC will start sending data to PSB

Then wait for the ROC to accumulate some measurements and transfer them to the PSB.

To download data from the PSB to local disk, turn off the 1 frame/min.
set curseur=0 Taille 1 => click transfer
It will report the current frame counter position.
The difference between this frame counter position and the initial frame counter position is the amount of new data that has been transmitted from the ROC to PSB.
set curseur = 0 Taille N => click transfer (where N is the number of frames to be transferred)
Note!! if the ROC is making measurements faster than it can transfer the data to the PSB (this is often the case for testing in the lab), then the amount of new data in the PSB is NOT all of the new ROC data that has been collected.

Turn back on the 1 frame/min so that the ROC will continue sending its data to the PSB.

At the end of the test measurement period, power off only the GPS receivers to stop collecting new data.
Go to the Télémesure Diférée Scientifique part of the screen (lower left)
In the TC Paramétrable field, type a command to the ROC:
=> P10 => envoi (send) (power to device 1 zero=off)
=> P20 => envoi (send) (power to device 2 zero=off)
=> @finish => envoi (send) (sends all the new config information to the ROC and starts using it)
=> @status => envoi (send) (sends the status information on the number of files and bytes remaining to be transmitted in the next data frame)

Wait until all of the data has been transferred from the ROC to the PSB.
The approximate data collection rate is 50 bytes/sample/satellite.
So for parameters of positioning mode (~8 satellites per sample) and 1 sample each 5 seconds, you have:
50 bytes/sample/satelllite * 8 satellites/sample * 1sample/5 seconds * 60 seconds/min = 4200 bytes/min
It will take ~4 times longer to download all the data than the recording time.
You can also check how much data is left to be downloaded by accessing the ROC SBC directly from the Zterm window and looking at the files in /var/occult/transmit.

Transfer the last of the data from the ROC to the PSB by doing the following:
set curseur=0 Taille 1 => click transfer
It will report the current frame counter position.
The difference between this frame counter position and the frame counter position at the end of your last transfer is the amount of new data that has been transmitted from the ROC to PSB.
set curseur = 0 Taille N => click transfer (where N is the number of frames to be transferred)

For testing, make sure not to exceed the maximum number of frames that can be stored in the PSB (509) before stopping the GPS receivers and transferring data. Otherwise the PSB will overwrite the previous data.

7.5 Detailed description of SOL PSB V2 software display screen

Top Left:
GPS Mode 2 (it the ISBA GPS - not ROC)
The PC is simulating messages with information from the ISBA ROC in the message.
Mode 2 means the GPS has calculated a 3d position.

The mode will change if the GPS does not track satellites, for example.
compteur=counter (maximum number of frames that can be stored in the PSB memory) (after it acquires this many frames it starts over-writing the first frames in memory)

Simulateur ISBA = ISBA simulator
1 Trame/min = 1 frame/minute
1 Trame / 5 sec = 1 frame/5sec
Aucune=none
After you start the program, start the ISBA simulator to start sending 1 frame/min
If you want PSB to automatically request data from ROC at the regular interrogation interval, then you need to set this to 1 Trame / 5 sec. Then PSB will get the time synchronization message each 5 seconds and then it will be possible for PSB to request data from the ROC at regular intervals.
To set the interrogation interval, use the hyperterminal configuration menu for the PSB (second serial output from PSB)

ELTA => menu 4 program scientifique => interval of interrogation => 1min

Top Center:
Relais=Relays
These icons indicate if the power relay from the PSB to the individual instruments and heaters are on or off.

Top Right:
Tab R4 is for ROC
Force On = turn on power from PSB to ROC manually
Force Off = turn off power from PSB to ROC manually
Auto turn power to ROC on and off at a specified time (set time ON and Off in fields to the right)
Consigne=Send start and stop times to PSB
Don't need to turn on any other tab (R1 R2 R3) except for R4
Tab R1 is for the PSB heater
Tab R2 is for the TSEN = LMD's dynamic pressure temperature sensor
Tab R3 is for the LMDOz = LMD's ozone meter (might also be UCOz for university of Colorado's ozone sensor)
R6 is for the ambient heater.

Middle Left:
Compteurs Evenements = Event Counters
Number of resets is shown
each time you cycle power on the PSB it augments the resets
or when you click on reset PSB button at the lower right

Lower Middle Left:
There are 2 types of frames stored in the PSB
PSB housekeeping frames (telemesure differe PSB Operationdelle
or Science data frames (telemesure differe Scientfique)
They are stored in 2 different places in PSB memory
you can download these frames stored in PSB on command:

Telemesure differe PSB Opérationnelle = manual download of PSB housekeeping frames
Taille = Size = the number of frames to download starting from cursor
Curseur =Cursonr= which frame number to start with (0=last frame, 1= 1 frame before that, 2= 2 frames before that)
Transfert= Transfer (click to start transfer)

Suite=Next ??
Fin=End ??
Trames restaintes=remaining frames (how many left to download after clicking start transfer)
Cpt Trames=Frame counter (which frame in PSB memory the counter is at; this is updated during the transfer command)
Heure GPS=GPS time (the time of the record(?) in the PSB frame
After you click transfer, all of the fields in the GPS, Temperature, and Voltage sections should update with their current values.
Program writes the frames to the folder (for example):
Programs/ELTA/SOLPSBV2/ PSB09V00N42A
new folder for each flight (this one is flight 42)
Writes housekeeping data to
PSBNCU09V000N42ADataOpDiff.asc
There is one line per minute of housekeeping data in here
It appends to the end of the data file each time the data is downloaded
It can store about 2.5 days of housekeeping data in the part of the PSB memory allocated for that.

specify the size (number of frames we want to recover from the PSB memory)
if there is one each minute, you can get 2 frames by requesting 2 frames
with the size=2 and cursor = 0 gets the last 2 frames.
size=2 and cursor =2 gets the last 4th and 3rd frame but not last 2nd and 1st
cursor 0 is where the cursor is at the moment that you make the request.

Lower Left:
Telemesure Differée Scientifique = manual download of science data frames
Choose ROC Tab
Taille = Size = the number of ROC data frames to download starting from cursor
Curseur =Cursor= which frame number to start with
Transfert= Transfer (click to start transfer)
Suite=Next ??
Fin=End ??
Trames restaintes=remaining frames (how many left to download after clicking start transfer)
Cpt Trames=Frame counter (which ROC data frame in PSB memory the counter is at; this is updated during the transfer command)
Heure GPS=GPS time (the time of the ROC data frame (?)
Program writes the frames to the folder (for example):
Programs/ELTA/SOLPSBV2/PSB09V00N42A
Writes to one file for each instrument, for example for ROC:
PSBNCU09V000N42ADataROCDiffSci.asc
It appends to the end of the data file each time the data is downloaded using the Telemesure differe command or using the Ordre TM command or at regular intervals if AUTO setup is used.

You can send a TEXT command to the ROC.
Type the character command (without the "TEXT) into the blank window next to the word envoi=send, then click on envoi=send

The format of the *DataROCDiffSci.asc file:
Hour -- 1028 bytes of data -- frame counter
not clear in this output if the 1028 bytes of data includes the frame counter and if it is redundant and is the same as the frame counter

the maximum number of frames stored in the PSB memory from the ROC that you can recover is 5 pages of 64 Kb:
one frame is 1028 bytes, therefore
5 pages * 64 Kb / 1028 => 311 frames => 5.18 hours

clicking on transfer get the DataOpDiff file for housekeeping
clicking on transfer in the science box gets you file ROCDiffSci
clicking on Ordre TM or AUTO gets you the file DataInstrum (for debugging)

7.6 Notes on Frame counting

When the PSB sends a trtm (=TC that requests a TM data packet), the ROC send ack
Then ROC sends a data frame then PSB sends an Ack.
If the PSB doesn't send an Ack, then it did not receive data and ROC needs to send the data packet again.
When the PSB receives the data and the CRC is good, then it increments the counter,
which corresponds to the order in memory.

An example:
The ROC sends a frame and PSB receives it and gives it counter number N
Then you turn off ROC for a while.
Then you turn on ROC 8 hours later and ROC sends another frame.
The PSB is going to give it frame number N+1

If you use the SOLPSB software to ask to download a frame you'll see the date of download of the data and the counter.
The ROC needs to have the actual date and time of the measurement in the data because date of download and counter number don't give us that information.
The order of the counter in the file is the oldest to youngest (logical).

If there is anything lost between the PSB and the ground station software it will show up as a discontinuity in the counter and the Ground station will automatically request the data again from PSB.

7.7 Monitoring the PSB using hyperterminal

ROC-1 has flightID=43
ROC-2 has flightID=42
You can verify the FlightID of the PSB is 43 using hyperterminal

Connect a PC to the PSB from the bottom PSB connector using a cable as defined in P79 in the wiring diagram (page 4/5 of the file sent 7/11/08 ??))
Note that both the PSB at CNES and the PSB at Purdue had trouble connecting to the ROC until the TX+ and TX- wires were switched. Same is true on the PSB to RS485/RS232 adapter. It may indicate an inconsistency in wiring of the connector on the PSB.

Start Hyperterminal

Set the hyperterminal so that all typing is in CAPITAL letters

Ater you connect, type the following password (upper case) to get to the menu:
ELTA

Menu de configuration=configuration menu
1 visualisation des parametres=1 View parameters
2 programmation de l'indentificateur = 2 program the flightID
3 Programmation de l'adresse = 3 program the address of the science instrument
4 Programmation des scientifiques = 4 program the science instruments
5 Programmation des relais et dessorties a C.O. = 5 program the power relays and outputs
6 Programmation de la date et de l'heure du PSB = 6 Program the date and hour of the PSB
7 Programmation des parametres avec les valeurs par defaut = 7 Program the parameters with the default values
8 Effacement du compteur de RESET = 8 Erase reset counter
9 Initialisation de la zone memoire NTS = 9 Initialize the NTS memory zone

Q= Quitter = quit

menu => choose 1 = View parameters

It shows flight ID, for example flight 042 (needed before starting SOLPSB program)
it shows also the address of the PSB = 04
it shows the addresses of the scientific instruments
For each instrument it also shows:
the duration of warmup ??? Do I need to choose something operationally here???
interval for data requests from PSB to ROC = 1 minute
duree cutoff = cutoff time = time period between when the PSB sends the TC message to turn off the power and the time the power is cut off. It requests the last data before it powers off the instrument.
taille de trame TM = size of frame for data = 1028 for ROC
nombre de pages of stockage = number of pages of data stored in PSB memory =8 pages of 64kb each

data for each relay:
R1 relay is temperature....
Relay4 is the ROC

menu => choose 2 = program the flightID

2 choose the flight number by typing + or - to get to desired number

menu => 3 = program the address of the PSB = 04
(we don't ever really have to do this)

menu => 4 program scientifique = program the Science package number that you want to assign (ROC is number 3)
then get another menu => enter the address = 3

Configuraton Menu for Science instrument 3
---
1 Programmation de l'adresse du scientifique = 1 program the address of the science instrument = 09
2 Programmation du type d'alimentation = 2 program the type of power = Relais (relay)
3 Programmation de la duree de 'Warm up' = 3 program the length of time for "Warm-up" = 0 minute

4 Programmation de l'intervalle entre interrogations = 4 program the interrogation interval = 1 minute
5 Programmation de la duree de 'Cut off' = 5 program the 'Cut-off' duration= 1 minute
(duree cutoff = cutoff time = time period between when the PSB sends the TC message to turn off the power and the time the power is cut off.)
6 Programmation de la taille de la trame TM = 6 program the size of the TM frame = 1028
(8 pages de 0 a 7, 510 trames, adresse de fin: FFF7)
7 Programmation du nombre de pages de stockage = 7 program the number of pages of storage => 8
(nombre de pages of stockage = number of pages of data stored in PSB memory =8 pages of 64kb each)
8 Initialisation de la zone memoire = 8 Initialize the memory zone (takes a long time!)
(??? how is this different from reinitialze the memory zone of the PSB???)
9 Selectionner un autre scienifique a configurer = Select another science instrument to configure

Q = Revenir au menu precedent = return to preceding menu

The remaining menu items are kind of incomplete below...
menu => 5 programming relays

menu => 6
programmation de la date et l'heure de PSB
gives date as 2/1/1 1/1 +A =port modifier la date

menu 7 parametre avec les valeurs de defaut

menu 8 Effacement du compteur de RESET = 8 Erase reset counter

menu 9 reinitialize the memory of the PSB whenever you change the parameters
Initialize the memory zone will replace all space with zeros in the memory of PSB (you don't have to do this every time you start up the PSB, only when you change the parameters)

menu =>? module automatique ??

In order to clear the memory of the PSB and reset the counter numbers:
hyperterminal
=> ELTA
=> menu 7 parametre avec les valeurs de defaut
=> menu 9 reinitialize memory

don't forget you might want to reset the interrogation interval again after this:
=> menu 4 => interval of interrogation

7.8 SolPSB Notes and email clarifications

O. Gallien 2010-01-29:
PSB memory as divided in two parts:
1- HouseKeeping memory for PSB own telemetry
2- Scientific memory
Scientific memory is composed of 10 pages of 65536 octets each. Through hyperterminal you can choose how many pages are dedicated to each scientific instrument. On PSC 2 configuration we

have: TSEN 1 page, LMDOZ 1 page and ROC 8 pages.

If you want to clean ROC memory you have to have to access to PSB configuration menu via hyperterminal with ELTA command. Then the choose "4 - Programmation des scientifiques". At the request "Saisir le numero du scientifique a configurer (de 1 a 4)" tip 3, ROC is declared as scientific 3 (hope it is the same for you, it depends on which PSB relay is commanding the ROC, relay 4 for scientific 3).
In the scientific 3 configuration menu, choose "8 - Initialisation de la zone memoire" and confirm by tipping "O". The memory erasure start. At the end press any key to come back to scientific configuration menu. Then press "Q" two times two quit configuration menu.
Then if you download ROC frames from SOLPSB you will find frames containing only 00 00 00.... With counter starting from 0.

This procedure is applicable only for clean

8 Ground Post-processing Software and Archiving Programs

8.1 Data Format

8.1.1 Pathnames on ROC

When the ROC is running in test mode, data will be written to the following directories:
/mnt/cf/occult/datasets
/mnt/cf/occult/transmit
/mnt/cf/archive (separate file system so that if it fills up, operational files will be unaffected)

When the ROC is running in operational deployment mode, data will be written to the following directories:

/var/occult/datasets
/var/occult/transmit
/var/archive (separate file system so that if it fills up, operational files will be unaffected)

On the physical memory stick itself, these locations are on separate partitions:
Partition 1 (sometimes called Untitled):
/var/occult/datasets
/var/occult/transmit
Partition 2 (sometimes called Untitled 1):
/var/archive

8.1.2 Filenames on ROC

A directory contains the "datasets" that are currently being collected and processed. It is specified by the environment variable DATASET_LOCATION, e.g., /occult/datasets. Datasets grow to a maximum size. When completely processed, they are compressed and archived as long as the ARCHIVE_DIRECTORY environment variable is defined (e.g. /archive) and there is sufficient archive storage space. The archives of datasets use the same file name conventions as the original datasets. These archived datasets may be decompressed by the unarchive program but must use the –nometadata option. See the unarchive program documentation.

There is one growing dataset for each GPS receiver for each of positioning stream or occultation stream, so there are two growing datasets, maximum. However, any number of dataset may be in queue to be processed. As these datasets are being collected they are processed into units called segments. These segments may contain one occultation event or records between occultation events. Since the end of a dataset always forces the end of a segment, one occultation may be split between two segments. For this reason, occultation datasets' maximum size is set large to avoid split occultation. Since position datasets are not processed according to occultation events, positioning mode implies 1 dataset = 1 segment. Position datasets may have a separate maximum size from occultation datasets and is configurable. Once a segment has been identified, it is written to the transmission directory (defined by environment variable BASE_TRANSMISSION_DIRECTORY, e.g., /transmit).

As the PSB requests data to be transmitted, the segments are broken down into "frames", these frames fit into a CNES protocol packet. The frames contain segment headers to re-assemble segments from a sequence of frames. These headers contain segment sequence numbers. After all the frames of a segment have been acknowledged as received by the PSC, the segment is deleted from the /transmit directory. The first frame of a segment contains metadata: gps week and gps seconds within week for start of the segment along with satellite occultation and calibration data. These are used to create metadata headers for the segment files created PSB .asc files.

Both segment and dataset files are named chn_GPSweek_GPSseconds where the GPS parts are the time the segment or dataset started. GPSweek is the gps week number in decimal, GPSseconds are seconds into GPSweek and chn = 1 for receiver 1 and 2 for receiver 2. The week and seconds are left-zero padded to provide for fixed-length filenames whose lexical order is the same as temporal order per receiver.

Once a PSB file is received from CNES that represent a series of frames, it is re-sequenced into archived segment files named by their sequence number: 1.seq, 2.seq … . See the resequence documentation below for details on its use. The resequence program has options to take care of CNES sequence number repeats and splitting of segment files across PSB files.

The unarchive program takes as input a file output from resequence and generates the uncompressed segment file with metadata header. Metadata contains the information to generate the original chn_GPSweek_GPSseconds format. The output file name is then the standard stated above. The unarchive program can also print the metadata in ASCII format. The metadata contains information about satellites in occultation and their calibrating satellite. See more information about the unarchive program below.

8.1.3 Pathnames for files received from Ground Station

Files are automatically transferred from
PSB => ISBA running Iridium software => Iridium satellite => Ground station => decryption => web site

PSBSOLVOL....DATAROC.ASC
The format of these files is
PDATE frame counter
where
PDATE is some date like the download time
frame is one "frame" of data in hex ascii format, 1024 x 1 byte characters
counter is a consecutive frame counter corresponding the label of the frame inside the PSB.

The files are continuously appended to each time the ground station downloads
Note: the counters could be duplicated or out of order, but should all be there.

Check the counter is consecutive by typing
awk '{print $1024}' PSBSOL*ASC
awk 'BEGIN {s=0} $1024 != s+1 {print} {s=$1}' PSBSOL*ASC

8.1.4 Filenames for Decoded Data Files

The typical sequence for decoding processing the data files is the following:

resequence.pl PAB*ASC
outputs:
XX.seq
These correspond 1 to 1 with the segment files in chn_GPSweek_GPSseconds above

unarchive *.seq
outputs:
chn_GPSweek_GPSseconds.seg (metadata for each segment file, optional)
chn_GPSweek_GPSseconds (trimble RT17 file for each segment file)

print chn_GPSweek_GPSseconds (outputs ascii information about what is in input file)

rt17_to_rinex chn_GPSweek_GPSseconds > chn_GPSweek_GPSseconds.10o
outputs:
chn_GPSweek_GPSseconds.10o (rinex format file)

Notes on machine dependency for binary double floating-point formats:

Trimble GPS rt17 output is IEEE 'standard" big endian
SBC is 32-bit little endian ; use makefile.arm
Linux on i686 is 32-bit little endian ; use makefile.i686
Mac (Intel) is 64-bit little endian ; use makefile.mac

Note the double floating point on SBC Linux and i686 Linux are the same. Other platform dependencies, notably the use of RS-485 on the SBC represent the differences in makefile.arm and makefile.i686.

8.2 Decoding CNES Sequence Files

8.2.1 resequence

There are several steps to decode CNES sequence (*.asc) files. First the files must be resequenced into compressed GPS segments, decompressed, and then turned into readable text. Multiple sequence files must be processed at once since sequence file boundaries are not segment file boundaries. This is done with the resequence program:

resequence [--s start_sequence_number] [--c counter] sequence_file_1 sequence_file_2 .....

where [] represents optional argument. The files must be ordered chronologically on the command line. The program will read through all the sequence files in order reading frames and creating new files when a start-of-

file header is found in a frame. These are the segments files that had been transmitted save for the fact that they now have there sequence tag as well (this is of no real consequence). The optional start parameter will not write segment files until the segment with that sequence number is found in the stream of frames. This allows for todays incomplete segment at the end of the stream to be recovered completely tomorrow. When processing the last sequence file (say, sequence_file_n), the program will report the sequence number of the incomplete segment (e.g., 50) at the end. When the next sequence file is obtained the command

resequence -s 50 sequence_file_n sequence_file_n+1

will recover the complete segment (unless it is really long). The output files are named by their sequence number with the extension .seq.

Sometimes, the CNES may repeat frames identified by the CNES sequence number stated at the end of each record (line) of the .asc file. The --c counter option allows you to say, ignore all CNES sequence records before this sequence (counter). The resequence program outputs the CNES sequence numbers associated with the beginning of each segment file to aid in reconstructing the segment files correctly. The resequence program will ignore out-of-sequence records within the .asc file itself.

resequence by default uses the flag --metadata
the flag --!metadata turns off the metadata, i.e., it will not assume the first part of the file is metadata and attempt to place the sequence number generated by the ROC into the metadata.

resequence looks for the first intact segment if it finds that the PSB*asc file starts in the middle of a segment. It allows for several input files, for example on the first day you would have file1, file2, file3, file4:

resequence.pl file1 file2 file3 file4

Now probably the last file does not begin or end on a segment boundry, so to get the complete last segment of file 4 you can, the next day, do

resequence.pl file4 file5 ...

8.2.2 unarchive

These files can be decompressed by the unarchive program:

unarchive [-n] [-x] [-u] [-w] [-d] 50.seq

Again, [] represents optional arguments. This will create two files with names in the format chn_GPSweek_GPSseconds along with an extension .seg for the optional metadata file. The .seg file is a readable description of the segment and contains which satellite (if any) went into occultaton and the calibration satellite. The standard output contains the type 57H raw data records and has an extension of .rt17. The chn_GPSweek_GPSseconds format is as described above.

Options

-n The file contains no metadata header. In this case no .seg file can be produced. (This is the case for datasets that were archived to the archive directory on the ROC) Minimal metadata is gleaned from the

file name if it is an archived file with the chn_GPSweek_GPSseconds format. UTC offset from GPS time is assumed 15 seconds unless UTC_OFFSET environment variable is defined as offset in seconds.

-x Translate the pseudo code 58H record type (resampled) back to the true 57H code.

-u Do an unarchive rather than an archive. The program operates one way or the other depending on the file name of the binary program file. The -u is used for debugging operations to force unarchive.

-w Write the optional ASCII metadata output.

-d Discard metadata header, do not place it as a header on uncompressed rt17 records. (The print and rt17_to_rinex programs by default expect these headers)

8.2.3 rt17

To turn the uncompressed files to text use the gps_print command:

rt17_print [-o] [-s] chn_GPSweek_GPSseconds

where the chn_GPSweek_GPSseconds file is one from the unarchive process. The optional -o says to print only type 57H records, no resampled records. The -s (for short) prints only the satellite count and the elevation and azimuth of each satellite. Output is to standard out.

Complete example run of resequence, unarchive, print:

```
Macintosh:data wyss$ ls
PSBNCU09V04N42ADataROCDiffSci.asc  resequence.pl
rt17_print                         unarchive
process

Macintosh:data wyss$ cat process
#!/bin/bash


for i in $( ls *.asc ) ; do
     ./resequence.pl $i
done
for i in $( ls *.seq ) ; do
     ./unarchive $i
done

Macintosh:data wyss$ bash process
Creating 5.seq
Creating 6.seq
Creating 7.seq
Creating segment description for sequence 0, Receiver 2
Creating segment description for sequence 0, Receiver 2
Creating segment description for sequence 0, Receiver 2
Macintosh:data wyss$ ls
2_4A531935_0001.dat           6.seq
```

```
2_4A531935_0001.seg           7.seq
2_4A534636_0001.dat
     PSBNCU09V04N42ADataROCDiffSci.asc
2_4A534636_0001.seg           print
2_4A5347DA_0001.dat           process
2_4A5347DA_0001.seg           resequence.pl
5.seq                         unarchive
Macintosh:data wyss$ ./rt17_print 2_4A531935_0001.dat | more

GPS week receive time: 208028000.000000
Satellite Count: 6


Satellite: 2
Elevation and azimuth: 16, 213
L1 real-time survey data
       SNR: 42.000000 dB
       Phase: -511554.140625 cycles
       Pseudorange: 23713451.701571 meters
       Doppler: -810.136734 Hz
       L1 slip count: 169
L2 real-time survey data
       SNR: 28.400000 dB
       Phase: -388500.890625 cycles
       Pseudorange: 1.851186 meters
       Doppler: 0.000000 Hz
       L2 slip count: 155

Satellite: 7
Elevation and azimuth: 34, 51
L1 real-time survey data
Macintosh:data wyss$
```

8.2.4 RT17 to RINEX

Convert RT17 unpaged records to RINEX format

Synopsis

rt17_to_rinex [-n] [-w gps_week] [-r receiver#] [-o UTCoffset] [-t template] [-s snr] rt17-file

Where

-n Indicates there is no metadata header that has been cascaded down the processing sequence to the input file.

-w specifies the gps week number for the dataset. This is useful for files with no metadata. In this case, the –r and –o must be specified as well.

-r receiver number data was collected from

-o UTC offset in seconds from GPS time (specify as positive number even though the actual offset to UTC is negative to be consistent with the BD950) This will override metadata.

-t rinex template (see below)

-s snr table file

The rt17 fortmatted file name typically has the format (output from the unarchive program) of

chn_GPSweek_GPSseconds

as described above. If no metadata is present and no gps week is specified in the command line, the metadata is gleened from the file name format. The UTC offset is assumed as 15 unless the environment variable UTC_OFFSET is specified as seconds of offset.

The program will take a header template, fill in what information it can from the rt17 file and also from the file name, write the header and then convert the relevant satellite information to RINEX records.

By default, the program looks for the file rinex.hdr as the header template to use. The –t option allows the user to specify another header template. The template must be formatted according to the RINEX specification. The following RINEX header records will be augmented from file and filename information:

# / TYPE OF OBSERV	(number and type of observables – from file)
TIME OF FIRST OBS	(time of first observation – from metadata or filename)
PGM / RUN BY / DATE	(program name and date run are replaced)

The record labels (starting at column 60) must be present for the information to be inserted. All other header records are simply copied and padded to provide 80 character fixed length records.

If the standard file name convention is not used, the gps week and receiver number can be specified by the –r and –w options. Receiver is a single number and gps week is a decimal number.

The RINEX standard provides for a signal-to-noise level indicator from 1 (minimal) to 9 (maximum possible) with 5 standing for “threshold for good S/N ratio”. The –s option allows the user to assign the minimum SNR value for level 5. Other levels are assigned linearly above and below the threshold.

Output is to standard out that can be redirected with the output redirection character (>).

Using the data in the example dataset:
../rt17_to_rinex -t ../examp_rinex.hdr 2_01571_086938.rt17 > 2_01571_086938.rnx

9 PSB - ROC interface diagrams

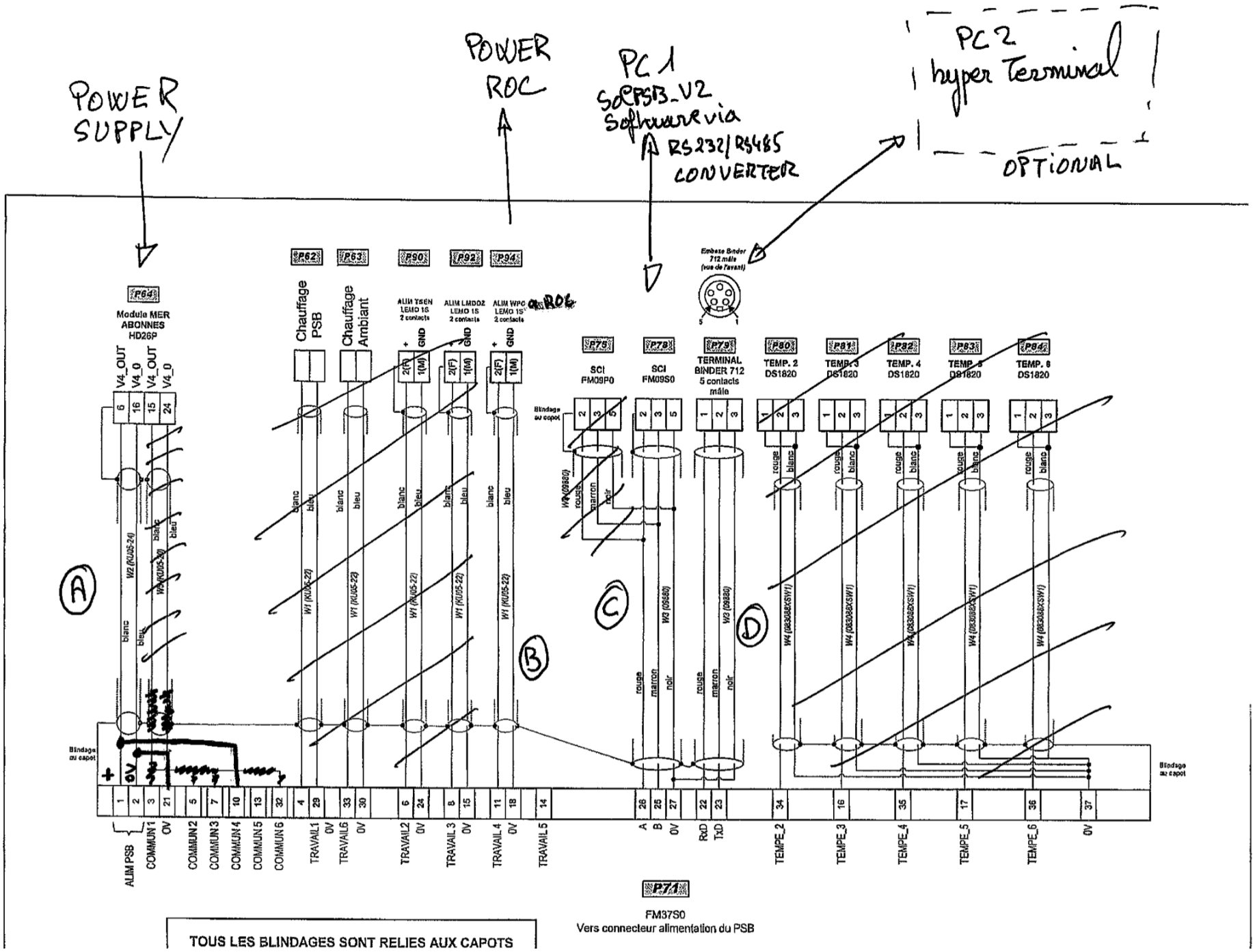
POWER SUPPLY
POWER ROC
PC 1
SoCPSB_V2 Software via RS232/RS485 CONVERTER
PC 2
hyper Terminal
OPTIONAL
P64
Module MER ABONNES HD26P
V4_OUT
V4_0
V4_OUT
V4_0
6
16
15
24
P62
Chauffage PSB
P63
Chauffage Ambiant
P90
ALIM TSEN LEMO 1S 2 contacts
P92
ALIM LMD02 LEMO 1S 2 contacts
P94
ALIM WPC LEMO 1S 2 contacts
GND
2(F)
1(M)
Embase Binder 712 mâle (vue de l'avant)
P75
SCI FM09P0
P76
SCI FM09S0
P73
TERMINAL BINDER 712 5 contacts mâle
P80
TEMP. 2 DS1820
P81
TEMP. 3 DS1820
P82
TEMP. 4 DS1820
P83
TEMP. 5 DS1820
P84
TEMP. 6 DS1820
Blindage au capot
W2 (KU05-24)
W4 (KU05-30)
W1 (KU05-22)
W3 (09880)
W4 (083088XSW1)
blanc
bleu
rouge
marron
noir
A
B
C
D
ALIM PSB
COMMUN 1
COMMUN 2
COMMUN 3
COMMUN 4
COMMUN 5
COMMUN 6
TRAVAIL1
TRAVAIL6
TRAVAIL2
TRAVAIL 3
TRAVAIL 4
TRAVAIL 5
RxD
TxD
TEMPE_2
TEMPE_3
TEMPE_4
TEMPE_5
TEMPE_6
0V
P71
FM37S0
Vers connecteur alimentation du PSB
TOUS LES BLINDAGES SONT RELIES AUX CAPOTS

Figure 5 PSB Pin wiring diagram for bottom DB37 connector to PC/SOLPSB com port and power connections.

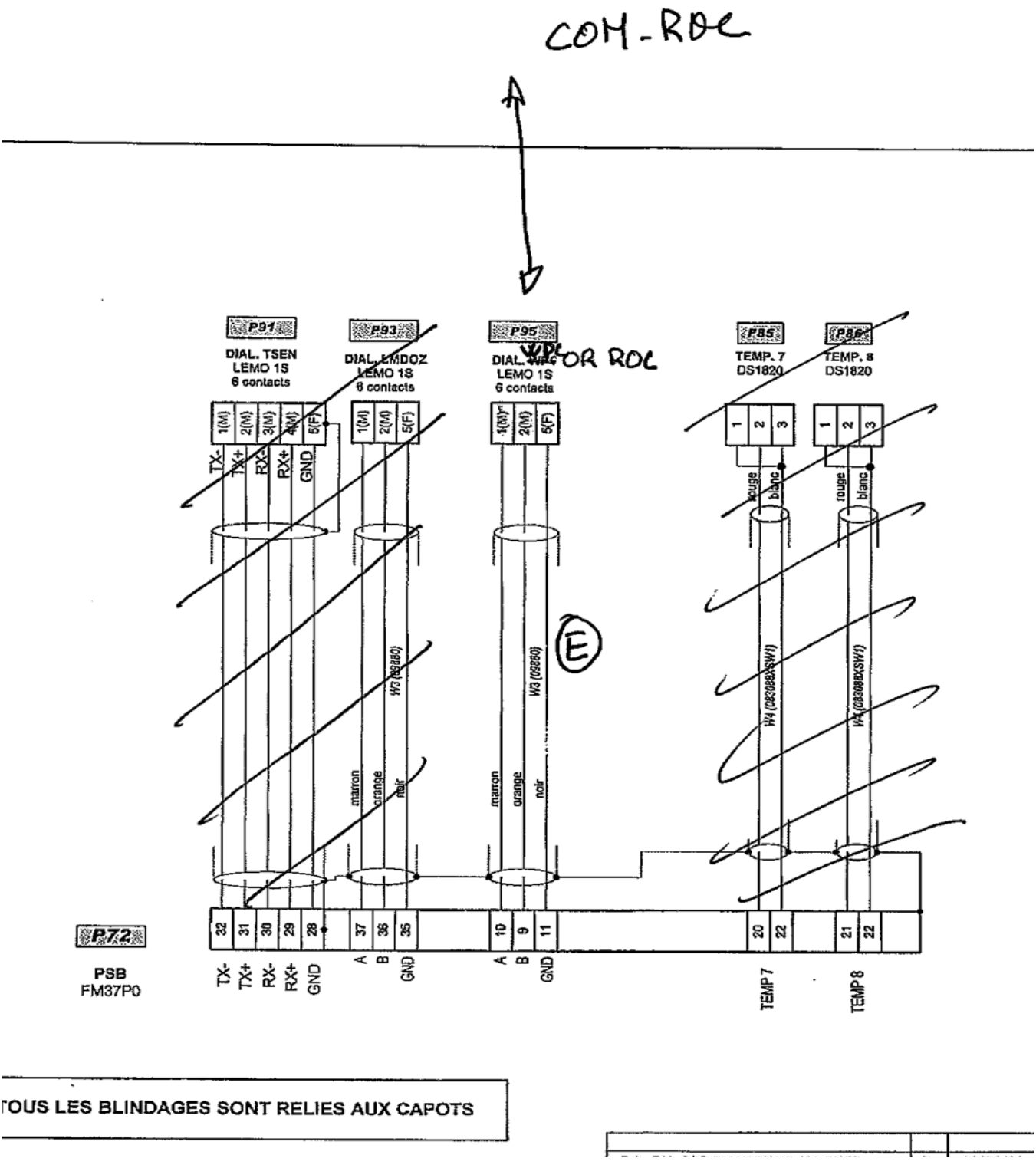

Figure 6 PSB Pin wiring diagram for top DB37 connector to ROC comm port.

10 GPS ROC Onboard Software Architecture

10.1 Pending requests for software upgrades

- command to move all transmit files to archive
- program to send time to send a text command

TBC 1 Who are these signals sent to/from?
TBC 2 CNESService does communication of satellite events to the arbitrator?
TBC 3 Define "event"
TBC 4 File structure on SBC

10.2 GPS ROC software schematic

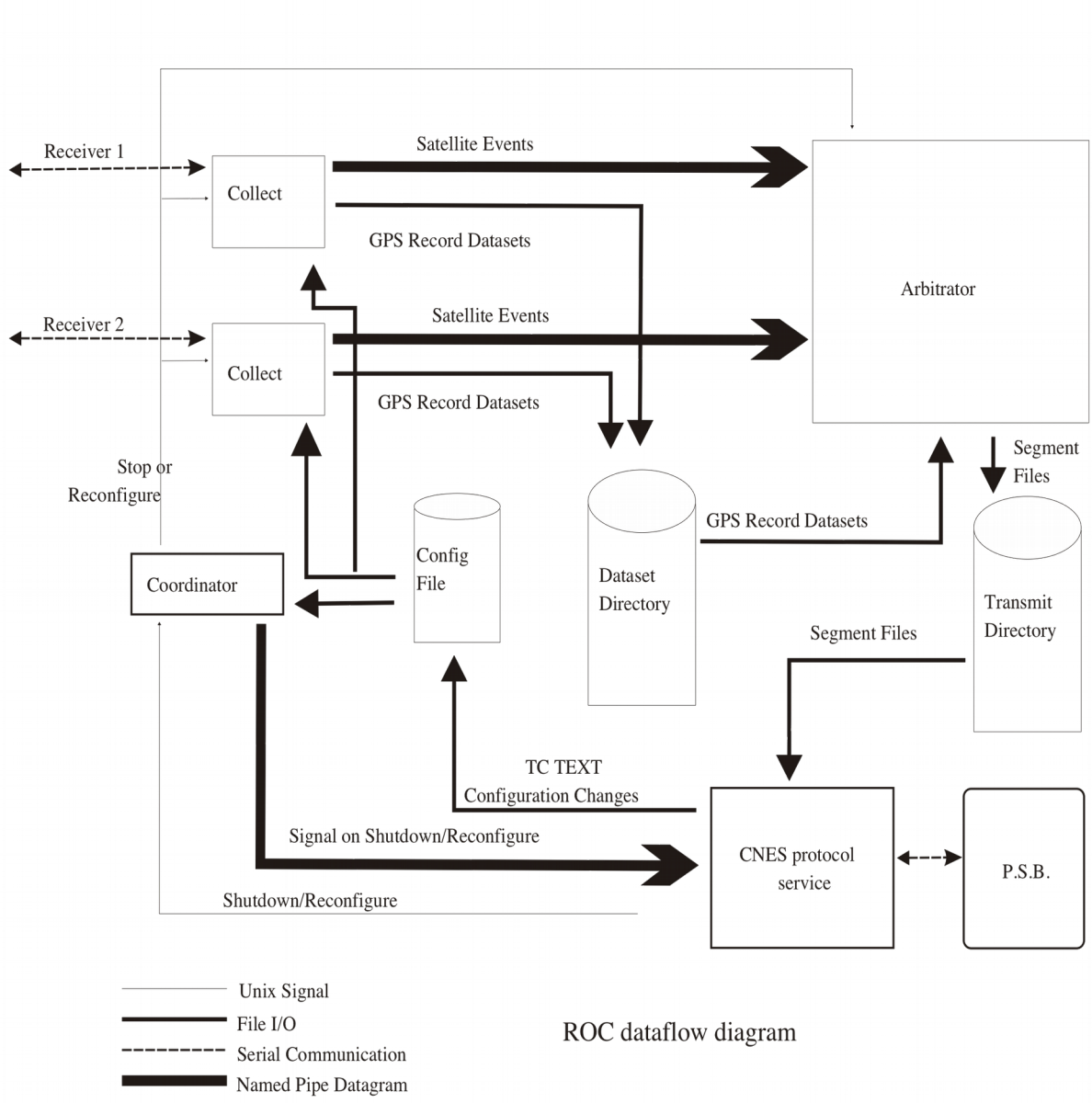

Figure 7 ROC dataflow diagram.

The main modules of the GPS Radio Occultation (ROC) software are the following:

- **Coordinate** - coordinates operation of all software modules based on the current configuration
- **CNESServer** - CNES protocol service handles communication with the payload supervisory board (PSB), which controls operation of the balloon platform
- **Collect** - this GPS handler controls and collects data from an individual GPS device
- **Arbitrator** - selects which of the GPS data are valid for occultation and passes the data segments on to be transmitted

See ROC dataflow (Figure 7) for a system schematic.

The coordinate module starts on system startup and starts other processes. The coordinate module tracks whether other processes sporadically die and so can recover. A background script will reboot the ROC if the coordinator exits unexpectedtly. The coordinator will provide an exit code of 0 if shutdown request was made and 1 for any other reason. System scripts will shutdown the ROC on code 0 and reboot on code 1.

The coordinate module detects changes to configuration files with the USRSIG2 signal. Depending on the configuration change, the coordinator will signal the collect module GPS handlers to either re-read their configuration file or terminate (as in the case of application mode change). The coordinator will also signal the collect module GPS handlers to terminate gracefully if a shutdown GPS type of command is sent. The coordinate module starts up the CNES protocol service to manage the interface with the PSB.

The CNESServer module is the CNES protocol service that will handle all transmitted frame acknowledgements, acknowledgement timeouts and CRC computation and checks. It will discover flight ID through an environment variable determined in the software installation procedure. The service interprets and executes command packets manipulating the configuration files if needed. The service will signal changes to the configuration file with signal SIGUSR2 (signal these changes to who?). The service will signal an imminent shutdown with SIGUSR1. The service uses the Unix socket datagram protocol for communication of satellite events to the arbitrator TBC 2 and for requests to the CNES service for shutdown/reconfigure notification. No other process should make decisions based on the external transmission protocol used. File I/O is used to transfer data and configuration information. All modules run in their own process/processes.

The collect module GPS handler will have knowledge of GPS devices, GPS data formats, data processing requirements and other experiment-specific information. The GPS handler program will control a single GPS card. Multiple processes running different configurations are used to handle multiple GPS devices. The GPS card reads its configuration from a configuration file on startup and when a USRSIG2 is sent.

The GPS handler will send an arbitrator process events containing information necessary to determine which occulations are to be selected for transmission (see prefix code comment for more detail). After selection the arbitrator will compress the data and write it to a queue directory where the CNES service shall find and transmit them. They are sent in order of the files creation/modification date.

10.3 Architecture of the coordinate program

10.4 Architecture of the collect program

Main Routine:

The main routine of the collect program does initialization of the GPS packet protocol, creates and loads the GPS receiver application file that controls the configuration of the receiver based on the contents of the ROC

configuration file, and then simply loops to wait for a GPS data packet, processes it and waits again. As it stands now, only one type of packet is received: the type 57H raw data packet. Type 57H records come as paged packets so a routine is called to gather additional paged packets and bind them together in one GPS raw data record. This record is then processed by the routine process_all_satellites located in source module filter_raw_data.c.

Workhorse Routines:

The filter_raw_data.c source module is the workhorse module for the program. The process_all_satellites subroutine walks through the type 57H record which contains all satellites and processes the data from each satellite individually with process_one_satellite. That routine, if the data is valid, calls the change_state routine. This routine looks at the current state of an individual satellite track to determine if it is setting or rising (such as the satellite has not been seen recently), looks at the current elevation and determines whether a state change is appropriate. Upon state changes, the arbitrator is notified as to the particulars of the event that caused the state change (such as satellite reaching the minimum occultation elevation).

Other modules of importance:

GPSpacket.c – does the low-level serial I/O for communicating with the receiver.

paged_packet.c – sends and receives receiver messages that span more than one GPS packet.

dataset.c – creates, read and writes GPS records. It generates the NEW DATASET and CLOSE DATASET events that are sent to the arbitrator.

getConfiguration.c – opens and reads the configuration file.

datagram.c - The collection program communicates with the arbitrator using the datagram.c source module.

TBC 3 Define "event"
Program event_recorder can be used to record the events output by the collection program (they are stored in standard output of the event_recorder). These events can be played back to the arbitrator with event_player which reads events from standard input.

For the collect program, the only useful command line argument is the receiver to use for collection. The receiver is indicated by a number (1 or 2). All other information is obtained by reading the first or second configuration block of the configuration file or environment variables that are set by the install scripts (see the CNESService description for information on environment variables).

Stopped editing here

10.5 Architecture of the arbitrator program

Each receiver/antenna device (these terms are used to mean the same thing) has an event fifo queue. OCCULTATION_START, SATELLITE_END, NEW_DATASET, and
CLOSE_DATASET are the only events queued. SATELLITE_START is process to initialize various data structures but not queued.

Each event contains the satellite and device associated with the device, what dataset it belongs to, the last record recorded before the event, what the calibration satellite for the event is, minimum S/N ratio and other info to

decide upon an occultation track. Note the type of event implicitly define which is valid and which are not. (NEW_DATASET has all satellites associated with it.)

The main program receives event messages from the collect program (SATELLITE_START, SATELLITE_END, OCCULT_START NEW_DATASET and CLOSE_DATASET are the important ones to know about the algorithm). Each one is initially processed and named-above events go to the per-device event queue to be process in the order they occur in the dataset stream.

When a satellite start event occurs a track collection structure is allocated. It contains information about the state of the occultation track for each device and it refers to the event structures that contain information to be used to decide on which device's information may be transmitted. A backpointer in the event structure also points back to the track collection. It is useful to think of track collection and event structure as orthogonal organizations of the same information. The track collection is a per-satellite structure (or more accurately a per-satellite occurrence from rising to setting). The event queue is a per-device structure.

The calibrator of the occultation track collection also has a track collection
as it is currently seen as well (the collection program rigorously
tries to enforce this). In that track collection, it is recorded which satellites have been using it as a calibrator.

When a satellite sets, all the satellites that were calibrated by this one is marked as a candidate track if the state of the track is marked as complete[3], othewise they are marked as invalid. When all the receivers for a given satellite are marked as something other than TRACK_FORMING or TRACK_COMPLETE, arbitration between devices are done. One track among all devices in the collection is marked selected and the others not selected[4]. The track collection is placed in a list. When enough occultations (programmable parameter) are listed, one is chosen among and marked chosen and the others are mark not chosen.

When the OCCULTATION_START event gets to the head of the queue of a given device the backpointer in the event is used to get the state of that device's track. If it is chosen, then the device's data structures are modified to indicate that satellite's information from occultation records are to be written. When the chosen track is detected, the corresponding calibration satellite has its calibration count incremented. (END_SATELLITE will decrement it). Satellites with non-zero counts are also to be written.

At the beginning of processing of a device queue, the first and last record written are set equal. Processing events appropriately set the last record to be at the record which occured just before the top event that cannot yet be processed. At the end of queue processing, if the last record now is greater than the first, records from first to last are written (with a filtering of occultation records) to a segment to be transmitted. The location of the file is controlled by the environment variable BASE_TRANSMISSION_DIRECTORY. See *Arbitrator Behavior – Dataset and Segments* power point presentation for a diagrammatic view of queue processing.

The arbitrator is built as one monolithic source module except for some data compression routines in compress.c shared by the standalone archive program.

10.6 Architecture of the CNES Server

The CNES server handles two jobs: communication with the PSB to deliver data and system status to the PSB, communication with local processes (clients) that generate the files to deliver. During server initialization, both

[3] Done initially at it's END_SATELLITE event.

[4] There are currently only two devices, but the program is built to accommodate an arbitrary number defined by parameter MAX_DEVICES.

the tty device channel to the PSB and the named pipe channel for clients are opened. The channels' file descriptors along with a call back routine and initial callback flags are passed to a routine (the callback dispatcher) that waits simultaneously for activities described by both channels' callback flags (channel read-ready, write-complete or timeout). When such an activity happens on a file descriptor, the callback routine for that file descriptor is called. The protocol callback handles tty-read-ready, tty-write-complete and timeout accordingly (see below). The callback routine for the named pipe (the command side) handles read-ready only as it is only waiting on commands from clients which are immediately satisfied and responded to. The callback routines return the set of flags they now want to respond to (the protocol callback doesn't need to respond to write complete if a packet can be written in one write call). When no flags are returned, the callback is never called again. If there are no callbacks left to call, the callback loop ends and returns to the main program which cleans up and exits.

As well if the command side needs intervention from the protocol side (send this data), it signals that need to the callback dispatcher which will schedule a call to the protocol side which will honor the request and eventually respond with a status to the callback dispatcher which then schedules a callback to the command side. The opposite happens as well as in the case of a TC TEXT command or a TC STOP command (although there is no response to the protocol side for STOP).

Although this architecture may seem overly complex, it is much easier to logically understand the structure of each side of the operation and therefore easier to debug.

Client Command Handling

Command side operation is fairly straight forward. It is called whenever there is outstanding read from the named pipe or a request or response from the protocol side. The callback handles it immediately and returns. Responses from the protocol side such as transmission success leads to a new transmission request to the protocol side keeping the protocol busy.

CNES Protocol handling

The coding for the CNES protocol at Purdue is described below by a state transition matrix. For each event possible, some action is taken and the protocol may change states. The matrix describes what action will be taken (if any) in any state given the event with the "new state,action-number" entry in the matching current state and event. The action descriptions follow the matrix. The internal system is notified of important events through command side operations.

Only valid packets and command side responses define an event. Packets with checksum errors do not generate an event. The "?" state indicates that the state has sub-states (transmission attempt count) that will determine the new state.

Events

SYNC – received a sync packet
TM – received a tmTR TC packet
ACK – received an acknowledgment
NCK – received a non-acknowledgment
TCS – received a TEST TC packet
TCD – received a STOP TC packet
TCT – received a TEXT TC packet
TO – timeout occurred

RR – received response to internal request

States

WS – Protocol is waiting for sync packet. This is the initial state.
IDLE – Protocol can enter any dialog
WA – Protocol waiting for a TM packet acknowledgment in TM dialog
WR – Protocol waiting on a response from internal request
S1 – Protocol waiting for TM in shutdown dialog
S2 – Protocol waiting for ACK of TM in shutdown dialog
S3 – Protocol exited

	WS	***IDLE***	***WA***	***WR***	***S1***	***S2***
SYNC	IDLE,1	IDLE,1	IDLE,2	WR	S3	S3
TM	WS,3	WA,4	WA,9	WR	S2,15	S3
ACK	WS,3	IDLE,5	IDLE,19	WR	S3	S3,19
NAK	WS,3	IDLE,5	?,10	WR	S3	?,16
TCS	WS,3	IDLE,6	IDLE,2	IDLE,17	S1,18	S3
TCD	S3,11	S1,7	IDLE,2	S1,7	S3	S3
TCT	WS,3	WR,8	IDLE,2	WR,13	S3	S3
TO	WS,3	IDLE	?,10	IDLE	S1	?,16
RR	WS,3	IDLE	WA,12	IDLE,14	S1	S2

Actions

1 Sync packet is parsed and stored for internal usage.
2 Unexpected, tell system that transmission has failed.
3 Packet/response/timeout ignored. Flight id is not known.
4 ACK sent. TM packet is transmitted and timeouts are initialized.
5 Packet ignored. Not in a known dialog.
6 Packet acknowledged/not-acknowledged per documented behavior.
7 STOP request acknowledged, signal shutdown to internal system.
8 Command sent to internal system.
9 ACK sent, timeout reset and TM resent.
10 Transmission attempt count incremented. If too many attempts, state is set to IDLE and signal the system transmission has failed. Internal system will request re-transmission later. Otherwise retransmit and stay in state WA.
11 Signal system to shutdown.
12 Response ignored.
13 Packet assumed to be retransmitted.
14 ACK/NAK sent depending on response.
15 ACK/Data sent, timeouts initialized.
16 Attempt incremented. If too many attempts,state is set to S3 and signal system transmission has failed (which may have already gone into shutdown state). Otherwise retransmit and stay in S2.
17 NAK sent.
18 ACK sent.
19 Internal system given transmission successful signal.

10.7 CNESserver API

The CNES server provides the data connection between processes (instances of running programs) in the ROC and the PSB. All data and status blocks are transmitted to the PSB via the CNES protocol. The main purpose of this document is to describe what is transmitted in the data frame, i.e., data portion of the CNES TM packet, and how other ROC processes instruct the server to transmit data and status

Transmission of data across CNES system

Since we want to compress data for transmission, we need to have some unit of compression. Typically the unit of compression is the file, so we will save the data stream in chunks to files, which are then compressed. In the following, these compressed files are called segments. We need then to indicate the beginning and ending of those segments in the CNES packet stream. To accomplish that the service would define several different data frames, that is the data portion of the CNES protocol TM packets. This scheme will work independently of the compression style chosen including no compression whatsoever.

Since there probably will be a need for some status information as well as data, there are several other frames types defined which may be intermixed with file frames and must be broken out of the stream. The different frame types have different headers. The header bytes would not be compressed.

The architecture of the data frames will be the following (the named data structures and header type definitions mentioned are contained in the header file commandLoop.h):

Byte 0: the data frame type

0 – start of segments
1 – segment continuation
2 – reply to command (Not all commands may have a reply, but may result in a change of status. This is just an anticipatory type)
3 – status update (unsure what would be, not implemented as the CNES protocol doesn't provide timely transmission of special packets)
4 – shutdown message – this has no content – it is three bytes long. It indicates that the next messages will begin at the beginning of a segment that was being transmitted at shutdown (if any)

Bytes 1 & 2: the data frame size. This is required as the data stream as delivered to the ground station may not keep the individual packet sizes and in general the files will not fit exactly into a set of full frames. The size includes header bytes.

For type 0 (start of segment):

Byte 3 and 4

File sequence number – all data frames with this identifier will refer to the same file until all possible sequence numbers are seen and then will be re-used. This provides for 65538 files before reuse.

Byte 5 - ?

File meta data: all the data that determined why this logical block of data was packaged into a file. The size of this portion should be fixed at some point. Currently, the only metadata is the size of the entire file. This will aid in the reconstruction of the file at the base station. The size of this may change during development but will be a fixed number of bytes in production. Responsibility for the meta data and compression has moved to the arbitrator. The CNESserver now simply transmits the segments byte by byte although it is true that the metadata lies at the start of the segment.

Remaining bytes: stream of initial bytes of the compressed file

This is represented in the code API as type **start_of_file_header** with type code **CMD_START_HEADER_TYPE**

For type 1 (segment continuation):

Bytes 3 & 4: file sequence number. This identifies the data frame as part of the file initially identified by a type 0 data frame.

Bytes 5 & 6: file frame sequence number. This tells us what frame of the file is being transmitted so we know how to put all the frames back together. This starts at 1 after the transmission of a type 0 frame. No file would be as big to reuse frame sequence numbers in the same file.

This header is represented in the code API as type **data_frame_header** with type code **CMD_DATA_FRAME_TYPE.**

Remaining bytes: continuation of the stream of bytes of the compressed file.

For type 2:

The remaining bytes will have to be defined according to what information pertinent to command sent expects. No current command (see “TEXT Command Types” below) generates a response of this type.

This header is represented in the code API as type **generic_frame_header** with type code **CMD_COMMAND_FRAME_TYPE**

Note the CNES protocol simply does not have a method for handling this frame adequately.

For type 3:

Status blocks will be transmitted on unexpected changes of state or on request. The structure of a status block is yet TBD. The status block contains device configurations as they actually are (power is off rather power should be off in this configuration). Currently the code expects the status block to be able to be contained within one data frame. Binary coding of the information should easily allow for this. (This still isn’t implemented wholely.)

This header is represented in the code API as type **generic_frame_header** with type code **CMD_STATUS_FRAME_TYPE.**

No real need has arisen to implement this and would not be handled well by the CNES protocol.

10.8 Software Control Commands

TEXT Command Types

These are the TC commands containing “TEXT” as the first four bytes of the TC data frame. For this section “command” refers to the string of bytes after “TEXT” There are two types of commands: device dependent configuration commands and device independent commands. The difference between the two is that device independent commands need no knowledge of the structure of the configuration data. Most of the CNES server code treats the status and configuration information as just a block of bytes of a given size. Only one section (config.c) has any knowledge of the structure. That section reads and writes configuration files and updates configurations given device dependent commands. The section receives the block of bytes along with its size from the rest of the CNESserver modules. This is a compromise between passing device dependent commands to a separate process and ensuring a quick response back to the PSB on all TC commands. Passing commands to another process would allow the CNESserver to not have to be re-tested at all when configuration commands are added. The additional overhead of using a separate process may take more time than allowed to respond to TC TEXT commands. The compromise allows unit testing of the config.c section without retesting the remaining sections.

Device independent commands, and only those commands, begin with an @ character.

@FINISH – write the configuration to a file with a standard path name and signal all requesting processes that the configuration update has completed. If the system should shut down or fail between the another TC TEXT command before this command is sent, the updated configuration will be lost. This simply calls a routine in the config.c section to write it however it need to be.

@HARD - do a hard reset. The system writes the file /reset and a reboot of the system is forced with no acknowledgement of the command nor notification to any other system module. The reboot process detects this file causing it to clean datasets and segments from the system. The default configuration will be copied to the normal configuration file.

@SOFT - do a soft reset. This is the same as a hard reset except that it will not do an immediate reboot. The actions to be taken on reboot will be done on the next normal shutdown/startup cycle. It will also remove all datasets and segments from the system transmit directory. This should be followed by a power off and power on sent to the ROC via the PSBSol interface.

@STATUS - Transmit status on next free PSB transmit request. The outstanding transmission already posted to the protocol side is transmitted first, but when command side of the service detects it has been transmitted, it will transmit the status. The status consists of some protocol statuses (e.g., number of checksum errors) as well as the number of segments in the queue to be transmitted and the total number of bytes to be transmitted.

In the current coding of config.c, commands are a single character followed by up to three parameters (though not all three may be of the numeric type)[5]. Each parameter may be either a four digit decimal number, a single decimal digit representing a device number or a true/false indicator (T/F or 0/1). To make adding additional commands easier, each defined command is described in a table which a parser uses to decode the parameters. The only addition to the code needed is to use the parsed parameters to change the configuration. Numeric data

[5] Adding parameter types would not be a difficult chore.

(indicated by dddd) are up to four digits (or an initial sign and three digits) that are left justified and space padded.

The power on/off command

Powers a GPS receiver on or off and if on, begins the current application

P*DS* - where *D* is a device number and *S* is the true/false state of the power being on.

Example:

P1T - set power on for device 1
P2F - set power off for device 2

Set Elevation

Sets the (Lower) elevation at which occultation recording is started.

L*Ddddd* – where *D* is the device number and *dddd* is the elevation in degrees. This may be signed.

Sets the (Upper) elevation at which position recording is started and stopped.

U*Ddddd* – where *D* is the device number and *dddd* is the elevation in degrees. This may be signed.

Maximum Gap

The time in seconds since the last transmission of a single satellite before it is assumed that the satellite has set.

G*Ddddd* – where *D* is the device number and *dddd* is an unsigned decimal number.

Resample Rate

Rate at which data is recorded when satellite is not in occultation or rate at which data is resampled for position application.

R*Ddddd* - where *D* is the device number and *dddd* is an unsigned decimal number. This will be coerced to a multiple of the occultation rate in seconds per record.

Occultation Rate

Rate at which data is recorded during a satellite's occultation period. This is the rate that the receiver outputs data and therefore must be a rate supported by the receiver. The rate is in seconds per record.

O*Ddddd* - where *D* is the device number and *dddd* is an unsigned decimal integer in seconds per record.

Occultation Selection

This is the number of occultations to be acquired before considering which is to be transmitted.

S*dddd* – where *dddd* is an unsigned number. The maximum is 32. Note that this is not device dependent, but it is listed here because it is implemented in config.c to keep the high level of modularity for the CNESserver. The server code knows nothing of the actual data or devices it serves except in the config.c module.

Application Mode

Sets the application mode to position or occultation for a device.

M*DA* – where *D* is the device number and *A* is either P or O.

Dataset Size

This sets the amount of data that the position application will acquire before attempting to send it off. This is done by closing the current dataset causing the arbitrator to transmit up to the CLOSE_DATASET event. The size is given in units of 1K bytes.

K*Ddddd* – where D is the device number and dddd is the unsigned size

Local Client Requests

The requests to the server and responses are sent and received by the datagram routines send_dgram and recv_dgram (this API is yet to be documented beyond code comments). The requests codes and data structures are defined in the header file commandLoop.h.

There is a requirement that none of these data structures be larger than the generic clientRequest structure. Each structure has their own typedef given below along with the request code. Those listed with typedef of clientRequest only use the command code portion of the packet, so only the command code of type/size client_command need be sent.

Unless otherwise documented, the response to the client process will either be CMD_OPERATION_SUCCEEDED or CMD_OPERATION_FAILED. The response contains the request code that generated the response and the response in a structure name response_block.

Update Device Status - This request will take data portion of the request and pass it to the config.c module to update the status of a given device. The device is known to the server by a device index (first device is 0). The request will fail if the device index is greater or equal to the number of devices in the configuration, known to the server through the config.c module.
(not fully implemented)
Request Code: CMD_SET_DEVICE_STATUS
Request Type: clientStatusRequest

Send Status - Force the transmission of the status block. This may be used for exceptional conditions such as the client detecting no power to a device that is supposed to be on. This call will fail if the size of the status block is larger than can fit into a data frame.
(not implemented)
Request Code: CMD_SEND_SYSTEM_STATUS
Request Type: clientRequest

Signal On Config – Send a signal to the given process on configuration @finish command. The request contains the process id of the process to signal (that process is not necessarily the requesting process). A failure is returned if the process list is full. However, the list is

curated for processes that no longer exist. The processes are signaled with SIGUSR1/SIGUSR2 when the configuration changes are started/finished respectively. If these one of the signals is not needed by the process, they can be ignored. A process may choose to catch either or both, e.g., ignore the SIGUSR1 signal but not the SIGUSR2 signal. Unfortunately, the default action for these signals is for the process to exit, so the signal must be explicitly ignored or caught.
Request Code: CMD_SIGNAL_ON_CONFIG
Request Type: signalRequest

Signal On Shutdown – Send a SIGHUP signal to the given process when the protocol receives a TC command of type STOP. The request contains the process id of the process to signal (that process is not necessarily the requesting process). A failure is returned if the process list is full. However, the list is curated for processes that no longer exist. Unless the process catches the signal, the process will be immediately terminated.
Request Code: CMD_SIGNAL_ON_SHUTDOWN
Request Type: signalRequest

Environment Variables

There are constants defined within the server for several file locations and the queue size. These may be overridden at server startup by means of environment variables. These are the names of these variables and there uses (some actually are only used by other modules, but they are all listed together here):

SIGNAL_LIST_SIZE - Number of processes that can fit into the configuration and shutdown lists.

QUEUE_STATE_FILE - File to save current file queue sequence number to restore in case of shutdown.

CONFIG_STATE_FILE - File to read/write for device configuration.

DEFAULT_CONFIG_STATE_FILE – File to read on startup if there is no configuration file (the server will fail to start if neither configurations are there).

DATASET_LOCATION - Where files from the initial collection reside.

BASE_TRANSMISSION_DIRECTORY - Where CNES protocol service finds files to transmit.

PREP_TRANSMISSION_DIRECTORY - Where file temporarily reside when being created for transmission.

OCCULT_HOME - Used in installation and boot-up as the basis to place the dataset, prep and base subdirectories and assign a value to them.

ARCHIVE_DIRECTORY - If defined the arbitrator archives whole dataset in archive format. The variable should point to a separate partition in case it is filled so that it will not impact ongoing operation.

CNES_ASSIGNED_ADDRESS - The address of the ROC used to communicate with the PSB.

TIME_ZONE_OFFSET	This is for debugging purposes when the PSB is not really running UTC time. Set the value to the number of second to offset PSB time to be UTC.

Most of these have default. The exception is ARCHIVE_DIRECTORY. If not defined, no archiving is done. The overrides to the defaults are listed in /etc/init.d/environ.sh except for CNES_ASSIGNED_ADDRESS, which is defined in /ect/init.d/device_address.sh.

11 Clarification Emails

Warning! Potential PSB Communication Problem:

The following problem was noted by LMD that there were extra characters occasionally being transmitte dthat interfered with the interprestation of the frames from the PSB, and we also experienced a similar problem, and programmed a similar solution. Attached is the email exchange verbatim:

From: "Albert Hertzog" <hertzog@lmd.polytechnique.fr>
Date: July 4, 2009 9:12:03 AM GMT-04:00
To: jhaase@purdue.edu
Cc: francois.danis@lmd.polytechnique.fr, wyss@purdue.edu
Subject: **Re: extra character**
Reply-To: hertzog@lmd.polytechnique.fr

Hi Jennifer,

I read this message likely too late...
Nevertheless, here is the strategy we used to try to eliminate wrong frames (or wrong character) circulating on the bus:
1) We used the flight number as a synchronizing byte: the flight number is indeed the first byte of every frame. So, when powered on, our instrument is initialized with an irrealistic flight number (0xFF), and when it receives the first frame with a correct CRC (meaning that it is almost certainly a valid frame), it sets the flight number to the right value (the one in this first valid frame) and keeps it until it is switched off.
2) From this time on, when a character arrives to our instrument, we check that it is the right flight number. If it is the case, we wait for all the characters (the length of the frame is indicated in the first bytes). If it is not the case, we throw the character away.
3) In case we receive a frame thats start with the right character, we check its CRC, that its length is correct, and that is is really intended to us. If everything is clear, we consider that the frame is valid and do what we have to do.
4) Note also that we use a timeout of 1s which is set when the frame starts with the correct flight number: all the frames sent by the PSB should last less than 1s. This is a further safety. If we did not receive all the characters of the frame within 1s, we throw away the received characters and we initialize a new communication (meaning that we wait

once again for the byte with the flight number, i.e. step 2)

I hope this can be helpful. Don't hesitate to ask back if you need further information, and sorry for the delay in my response !

Cheers,
Albert

From: François Danis <francois.danis@lmd.polytechnique.fr>
Date: July 6, 2009 2:43:30 AM GMT-04:00
To: jhaase@purdue.edu
Cc: hertzog@lmd.polytechnique.fr, wyss@purdue.edu
Subject: **Re: extra character**
Reply-To: francois.danis@lmd.polytechnique.fr

And even later than Albert...

For more information... the problem I had observed was a character "arriving" after a message. That character was taken into account and, as it was the first new character, it was the flight number so I was changing the flight number before sending the answer to the previous message... Mainly, my software wasn't strong...

CNES has everything you need to identify the problem. The spy Marc Minois had put on the lines had shown that sometime I was sending correct messages some time messages with the wrong flight number.

I am at work if you need more information

Cheers

fr

Begin forwarded message:
From: "Wyss, Phillip J." <wyss@purdue.edu>
Date: July 6, 2009 10:59:13 AM GMT-04:00
To: <jhaase@purdue.edu>
Subject: **New Package**

Attached is a new package. It prints the value of the extraneous characters as they are removed. From the extra info that Francois sent, this is not the same issue as the extra character trails the packet.
This means you should get a least one good packet. If the characters you receive in no way resemble the pattern sent, I suspect hardware. My bet would be the +/- wires of the 485 connection are switched.

From: Phillip Wyss <wyss@purdue.edu>
Date: July 4, 2009 6:19:26 PM GMT-04:00

To: "Haase, Jennifer S" <jhaase@purdue.edu>
Subject: **Re: cone of silence**

Thinking about this, it seems a good idea to have this even if the PSBB software was perfect. There are sometimes when noise can still look like a real character. In any case, I pretty much came up with the same solution to the problem, even up to the point of insisting that all of one packet cpme together within one secomd. The only difference is that the flight id is now a installation parameter, which in a small way, makes it more robust.

On Jul 3, 2009, at 5:29 PM, Phillip Wyss wrote:

------=_NextPart_000_00B4_01C9FC03.D5430350
Content-Type: text/plain;
charset=us-ascii
Content-Transfer-Encoding: 7bit

Here's a new package. Since I used the flight id as a sort of
start-of-packet indicator, I can't look for a flight id in the packet, so it
has to be an installation parameter. This means the device_address.sh file
includes it. The installation procedure looks to see if it is already
configured by the existance of the file. You have to delete it to get the
installation to prompt you for the flight id.

12 Email Clarifications (installation and testing)

Additional IP configuration notes:

To be able to access the IP connection when you are in development mode
Set "No Networking" in network_cfg file then reset up the connection (?)

If IP network doesn't get set automatically then this is something we did once. I don't really understand how to use it:

cd /etc/rc.d/rc3.d
ln -s ../../init.d/rc.inetd S20inetd
reboot

or this either:
start again
cd ../rc.d/rc3.d
ln -s ../../init.d/network S10network

don't know what this is for either:
set up ssh server
cd /etc/dropbear
dropbearkey -t rsa -f dropbear_rsa_host_key

cd /root
mkdir .ssh
dropbearkey -t rsa -f id_rsa

Initially when we wanted to automatically start up ethernet on login we did this:
had to start network
turn_ethernet on
cd /etc/init.d
vi init_devices
comment out turn_ethernet off

Additional email notes, not yet integrated into document:

2009-07-08 notes about files:
when in occultation mode, the files in /var/occult/datasets are very large so that there is a small chance that there is an occultation that spans the dataset.
A small part of the dataset is sent while the stuff is going on .
(refer to diagram)

try the following:
gpsInit 1 reset
echo $?
if it is 1 then it did not reset properly
if it is a 0 then it did reset properly
maybe application record is bad??

From: "Phillip Wyss" <wyss@purdue.edu>
Date: July 6, 2009 8:41:48 AM GMT-04:00
To: <jhaase@purdue.edu>
Subject: **RE: new questions**
Reply-To: <wyss@purdue.edu>

6) where do I put my test text files that I want to transmit?
Unproceesed datasets are in /mnt/cf/occult/datasets
Data segments to be transmitted are in /mnt/cf/occult/transmit

Place them in /mnt/cf/occult/transmit

From: "Phillip Wyss" <wyss@purdue.edu>
Date: July 6, 2009 8:17:23 AM GMT-04:00
To: <jhaase@purdue.edu>
Subject: **RE: new questions**
Reply-To: <wyss@purdue.edu>

The installation doesn't look for the config file it looks for the DEFAULT config file. It still exists. Just do the copy below and it will restore the configuration.

cp $DEFAULT_CONFIG_STATE_FILE $CONFIG_STATE_FILE

From: "Phillip Wyss" <wyss@purdue.edu>
Date: July 6, 2009 8:03:49 AM GMT-04:00
To: <jhaase@purdue.edu>
Subject: **RE: new questions**
Reply-To: <wyss@purdue.edu>

Replies to question in document:

(3) No it is not normal. Something destroyed the configuration. You can always get back to the original configuration.

cp $DEFAULT_CONFIG_STATE_FILE $CONFIG_STATE_FILE

(4) NEVER use the -n option. That is basically for the installation procedure (n stands for new). It erases all configuration information. In normal practice -d is used to specify which one of the receiver's configurations you are changing, followed by the parameters. Turning receiver 1 off is

changeConfig -d 1 -P 0

(small p changes the serial port)

(5) The network_cfg copied says do no networking, so it will never read the ifcfg-eth0. The original network_cfg is copied to the installation directory on initial installation and can be restored if necessary.

From: Phillip Wyss <wyss@purdue.edu>
Date: July 6, 2009 6:49:50 AM GMT-04:00
To: "Haase, Jennifer S" <jhaase@purdue.edu>
Subject: **Re: questions**

On Jul 6, 2009, at 2:05 AM, Haase, Jennifer S wrote:

1) In your install notes you say:
Some of these are settable using CNES TC TEXT commands. Running system modules are not notified when making these changes locally.

So I assume that if the configuration parameters are changed using CNES TC TEXT, then the running system modules are stopped and restarted automatically so the changes take effect? Is that true?

The modules are either notiified and the changes are made on the fly (position sample rate) or the coordinator stops and restarts the module (e.g. application mode) This is done when the TEXT@finish command is sent after configurations are made.

2) How do I set up a partition on a jump drive to use for the data transfer/archive in production mode?

I have a memory stick that is 476 Mbytes
On MacOS they have newfs to replace mkfs, but it doesn't say anything specific about FAT .
I have attached my manual pages for fdisk an newfs
Could you please tell me your best guess of what exactly the commands are for making the partition usng fdisk?
you said:

fdisk
mkfs -c vfat sda1 sda2
or something like that.

fdisk is available in the production environment. After inserting the thumb drive and restarting the system (this ensures the naming of the pseudo scsi unit)

fdisk /dev/scsi/host0/bus0/target0/lun0/disc

type m <return> for help

create new partition table
create new partition, primary, partition 1, first cylinder, about 1/4 the default last cylinder.
create new partition, primary, partition 2, default first and last cylinder
write new partition table
exit

This will create a partition thumb drive 1/4 of which is used for production, 3/4 for archive. This is appropriate depending on the size of the thumb drive.

I used Knoppix for this, so not sure how to proceed. Need to experiment a little.

fdisk (2 primary partitions)
mkdosfs -F 32 /dev/sd..... (whatever device name)
put them in mac and gave them titles