Hello Evan,

This document is for you to quickly get up to speed on OSPRE (team name) software design. This document contains a general description of the OSPRE system, a method for navigating our design, and the major design questions we would like to discuss with you.

**General Information:**

* OSPRE code will be written in C/C++
  + Each C++ class will have its own header file and implementation file
* OSPRE will be running on a Linux OS
* OSPRE processes will be split between Zed Board (Simulated Spacecraft) and onboard processor
* OSPRE will be running on multiple processes
  + OSPRE will consist of 5 processes ( Camera Controller, Image Processing, GNC, SCComms, and WatchDog)
  + There will be an additional external process that we will also build called S/C which will run on the Zed Board and act as the Spacecraft
* All OSPRE processes will be event driven server processes
* All communication will occur over TCP/IP non-blocking sockets
* OSPRE will not be multi-threaded

For further details on OSPRE Processes see: Process\_Definition.docx

**Description of OSPRE Server Process:**

* All processes will listen at a defined port number for new connection requests
* Each process will have a defined number of connections
  + All processes will be accepting connections
  + Processes will connect to each other in a predefined manner
  + Only WatchDog will be written to support N connections
* All processes will be event driven and will queue time events as needed

Main Server Loop Example (Pseudo Code):

init();

while (1) {

select() with timeout

if statement // Handle Error Cases

if statement // Handle Timeouts

if statement // Handle Data Events

}

cleanup();

1. For further details on OSPRE Server Process Model see: OSPRE\_Server\_Model.docx
2. For more information on Class Abstraction see: Class\_Abstraction.h
3. For working code examples see: server.c, client.c

To run code:

gcc server.c –o server

gcc client.c –o client

Terminal Window 1: ./server (Port number between 5000 and 30000)

Terminal Window 2: ./client localhost (Same Port Number) (Message in single quotes)

**Startup:**

* When the OSPRE sensor package receives power, the microprocessor onboard will turn on and boot up Linux OS. On startup, a script will be run to start our OSPRE system.

For further details see: Startup\_Protocal.docx

**Shutdown:**

For further details see: Shutdown\_Protocal.docx

**Running Modes:**

* Normal Operation:

For further details see: Normal\_Run.docx, SoftwareModel2.pdf, and Message\_Protocals.docx

* Sleep Mode:
  + During sleep mode, processes will remain turned on in sleep cycles occasionally sending their status to WatchDog

For further details see: Message\_Protocals.docx, SoftwareModel2.pdf, Sleep\_Run

**Error Cases:**

See Error\_Cases.txt

**Design Questions:**

**How should the Spacecraft send OSPRE data messages?**

Option 1:

OSPRE sends a request to the S/C asking for input values at a certain point in time. (Request / Response)

Good:

* The Spacecraft only has to send a minimal amount of messages so the Spacecraft CPU is free to do other things

Bad:

* Makes OSPRE message system slightly more complicated (More Message Types, More message passing)

Option 2:

Spacecraft automatically sends updates of OSPRE input values when they change

Good:

* OSPRE sends less internal messages, loose request data message

Bad:

* S/C is sending OSPRE many more messages then OSPRE is using.

**How will we know if we can take pictures or not? (This is a question because we are ignoring lost in space scenario because it is out of scope)**

Option 1:

We are always taking pictures and Image Processing is determining if there is a body in the image. Earth / Moon determined by object angular diameter size at known distance.

Good:

* Eliminates need for S/C to tell OSPRE if it can take pictures or not

Bad:

* Adds a ton of complexity to Image Processing and risk to our project

Option 2:

Spacecraft sends OSPRE a message saying it can take pictures now and that it can no longer take pictures now.

Good:

* Simplest approach

Bad:

* Breaks encapsulation because the spacecraft now needs to know details about OSPREs camera
  + OSPRE should be determining if it can or cannot take a picture

Option 3:

Spacecraft sends OSPRE pointing information and angular velocity and Camera Controller determines if it can take a picture. (Uses camera FOV, Exposure time, and pointing information)

Good:

* Keeps encapsulation in the OSPRE system
* Keeps encapsulation within the Camera Controller

Bad:

* Requires extra input to OSPRE system
* Added Complexity