# HyperNAV Firmware Documentation

The source code is in Perforce at Firmware\Radiometers\Hyperspectral\Source\HyperNAV. There are two subdirectories for the firmware for each processor:  
 \Controller\Source\HyperNAV\_Controller\src  
 \Spectrometer\Source\HyperNAV\_Spectrometer\src

Compilation is done within Atmel Studio, version 6.2 as of 2015-09-21.  
The Atmel Solution files are in the respective Source directories specified above.

There is one feature not properly done: Data exchange between Controller and Spectrometer is via packets. The packets are defined in the data\_exchange\_packet.h file which must be present in both solutions. There are currently two copies of that file, and changing that file in one solution requires changing it in the other one as well.

## Overall Structure Common for Controller and Spectrometer

Files that serve similar/analogous purposes in each firmware have matching names, e.g.:  
 main.controller.c  
 main.spectrometer.c

Each firmware has a setup & command task, a task monitor, and a data exchange task.

### The main() Function

Files  
 main.controller.c  
 main.spectrometer.c

The code in the main.\*.c files is fairly brief:

1. Initialize board and firmware (interrupt handling, watchdog, power management). Not properly done yet. Requires hardware understanding (Scott).
2. Create all tasks that will run on the respective board, and start the task scheduler.

### Task Definitions and Task Control

Files

tasks.controller.c  
 tasks.controller.h  
 tasks.spectrometer.c  
 tasks.spectrometer.h

The task.\*.h file defines for each task that will run the following:  
 HNV\_<the-task>\_NAME   
 HNV\_<the-task>\_STACK\_SIZE  
 HNV\_<the-task>\_PRIORITY  
 HNV\_<the-task>\_PERIOD\_MS  
The file also declares two global variables:  
 gHNV\_<the-task>\_Handler  
 gHNV\_<the-task>\_Status

In task.\*.c, the Task Monitor periodically checks on the Status of all tasks, and triggers a watchdog reset if either task falls for more than ?? seconds into an unknown status.

Each task has the same basic interface and code principle. There are <the-task>.h file specifying the interface and the <the-task>.c file for the implementation.   
 <the-task>\_createTask() allocate the task and internal variables  
 <the-task>\_resumeTask() start execution of the task   
 <the-task>\_pauseTask() task will be asleep, only periodically wake up to check is started  
 <the-task>\_task() a static function performing the task specific functionality

|  |  |  |
| --- | --- | --- |
| **Task Name / Description** | **Controller** | **Spectrometer** |
| Setup and Command Handling | Always running | Always running |
| Task Monitor | Running as soon as startup done | Running as soon as startup done |
| Data Exchange | Start ASAP | Start ASAP |
|  |  |  |
| Stream Output | Start/stop | N/A |
| OCR-504 Data Collection | Start/stop | N/A |
| Profile Manager | Start/stop | N/A |
| USB Virtual Com Port | Start/stop (not coded yet) For Bootloader Access | N/A |
| USB Mass Storage | Start/Stop (not coded yet) For data offload | N/A |
|  |  |  |
| Data Acquisition | N/A | Start/Stop |
| Profile Processor | N/A | Start/stop Includes Modem I/O |
|  |  |  |
|  |  |  |
|  |  |  |

### Setup and Command Task

Files  
 setup.controller.c  
 setup.controller.h  
 SystemAPI/hypernav.sys. controller.c  
 SystemAPI/hypernav.sys. controller.h  
 SystemAPI/power.c  
 SystemAPI/power.h  
 config.controler.c  
 config. controler.h   
 command.controller.c  
 command.controller.h  
  
 setup.spectrometer.c  
 setup.spectrometer.h  
 hypernav.sys.spectrometer.c  
 hypernav.sys.spectrometer.h  
 power.spectrometer.c  
 power.spectrometer.h  
 config.spectrometer.c  
 config.spectrometer.h  
 command.spectrometer.c  
 command.spectrometer.h

Initially, only the setup and command task runs; all other tasks are created but are in their initial (paused) state.

Setup will  
 initialize all hardware components  
 start the task manager  
 retrieve the configuration settings  
 perform configuration specific setup  
 enter command and control system

The command and control system never terminates.  
 First, select tasks are resumed (from their initial paused state).  
 Then, an infinite loop executes, responding to incoming requests  
 [Controller and Spectrometer] Check for incoming packets (from other tasks)  
 [Controller] If in profiling, check for profiling commands  
 [Controller] If in continuous acquisition, check for interrupt to enter command mode  
 [Controller] If in command mode, check for commands  
 [Spectrometer] Check for input via maintenance port

### Inter-Task Communication

Files

data\_exchange\_packet.c  
 data\_exchange\_packet.h  
 data\_exchange.controller.c  
 data\_exchange.controller.h  
 data\_exchange.spectrometer.c  
 data\_exchange.spectrometer.h

Tasks communicate via packets. The packet format is defined in data\_exchange\_packet.h.

A packet consist of the from and to tasks (addresses), a packet type, and the packet data. If the data size is small (e.g., commands), data are included by value in the packet. Large data volumes (e.g., a measured spectrum) are passed by addresses to data locations. Then, care must be taken to re-use that data location only after the receiving task has consumed the data. The receiving task will then send a ‘Data\_Release’ packet back to the data owner, so the memory location can be used again. Under this memory management model, a task will maintain a set of pointers to memory, and keep track of which pointers are in packet transfers, and which pointers have been released by the receiving task.

Task communication can be between tasks that run on the same board or between tasks that run on different boards. All communication is performed via the data\_exchange\_packet\_router() function contained in data\_exchange\_packet.c. That function will determine how to route the packet.

If the destination tasks run on the same board, the packet is delivered directly via a public function named <the-task>\_pushPacket(). This function accepts a packet, and places it onto a queue that is private to that task. In its regular operation, the task will check for incoming packets, and handle the commands or data contained therein.

If the restination task runs on the other board, the packet is delivered to the data\_exchange\_<this-board>\_pushPacket\_viaSPI() function. The packet will then be received by the data\_exchange.<other-board>\_task, and that function will push the packet via data\_exchange\_packet\_router() to the appropriate local task.

## Controller

### Stream Log

Files  
 stream\_log.c  
 stream\_log.h

This task will receive data packets from data acquisition tasks (HyperNAV radiometers, OCR-504), generate data frames, and both stream the frames to serial output and log the frames to eMMC.

### TODO: OCR-504 data Collection

…

### TODO: Profile Manager

In autonomous profiling, receive data from data\_acquisition task, generate frames, and save frames to eMMC.

In transfer mode, package profile into transfer packets, retrieve respective frames from eMMC, send to profile processor for compression and encoding.

### Other files

command.errorcodes.h

config.errorcodes.h

crc.c

crc.h

datalogfile.c

datalogfile.h

errorcodes.h

extern.controller.c

extern.controller.h

filesystem.c

filesystem.errorcodes.h

filesystem.h

frames.c

frames.h

info.c

info.h

io\_funcs.controller.c

io\_funcs.controller.h

version.controller.h

wavelength.c

wavelength.h

## Spectrometer

### Data Acquisition

Files  
 data\_acquisition.c  
 data\_acquisition.h

Currently supports data acquisition based on start and stop commands (i.e., continuous commanded).

To do: Support single frame collection for at-depth points.

### TODO: Profile Processor

Receive frames to be packed into transfer packet from Profile Manager.

Send transfer packet via modem.

Receive commands via modem, and pass them on to controller command task.

### Other Files

command.errorcodes.h

errorcodes.h

extern.spectrometer.c

extern.spectrometer.h

info.spectrometer.c

info.spectrometer.h

io\_funcs.spectrometer.c

io\_funcs.spectrometer.h

system.c

system.h

version.spectrometer.h