Appendix H

SPHERES COMMUNICATIONS

This appendix describes the communications system for the SPHERES experiment. It acts as a reference specification for communications between the laptop and the satellites. The interface specification has three sections. The first section describes the low-level interface with the DR200x modules. Next, the basic link-level interface (packet structure, protocols) is introduced. Then the interaction between the GUI and the satellites during testing is described.

H.1 DR2000 Configuration

The DR2000 firmware was customized for SPHERES to operate in *transparent* mode. Under this mode, the DR2000 will send data without checking any contents or formats, with only three important logical values: packet size and the to/from values. The DR2000 will consider as data any bytes sent to it and send them out 'as is' except for the escape sequence $$ at the start of a packet which indicates that the data after the escape sequence is a configuration word (described below).

H.1.1 DR2000 Packet Structure

The DR2000 packet structure consists of a pre-amble that excites the receiver crystal, a 5 byte header, the body of the packet, and a 2-byte CRC. Table H.1 lists the elements of the DR2000 packet. The pre-amble is hard-coded in the DR2000 firmware. The header is cre-

ated using the configuration of the DR2000 (which is stored in FLASH, but should be updated every time a project starts for data safety). The body are the bytes received by the transmitting DR2000. The CRC is created automatically from the data. Note that the body is the only thing that must be sent to the DR2000; the header and CRC's are completely transparent to the transmitter or receiver and are never seen by the receiver. Figure H.1 depicts the packet structure.

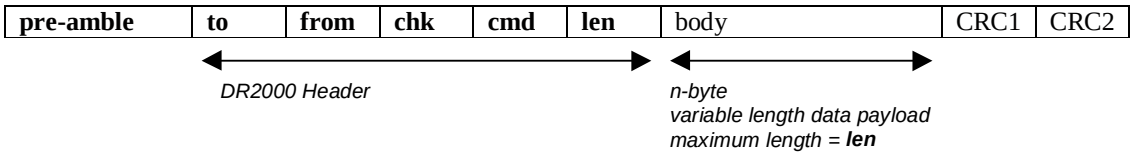

Figure H.1 DR2000 packet structure

TABLE H.1 DR2000 packet structure

Byte	Width	Name	Function
0 – 2	3	pre-amble	A hard-coded element of the firmware which sends a sequence of 0xAA and 0x55 bytes to excite the receiving satellite's crystal.
3	1	to	The intended recipient. Use 0x00 to indicate broadcast mode. Because setting the TO address takes approximately 400ms, the TO address should always be set to 0x0 (broadcast) except in special circumstances.
4	1	from	This satellite's ID. Only packets sent with the to equal to from or broadcast mode will be transferred out of the DR2000. All other packets will be discarded.
5	1	chk	8-bit checksum of the packet "body".
6	1	len	The length of the body in the packet. The maximum length is stored in the configuration, but this length may be variable in transparent mode.

TABLE H.1 DR2000 packet structure

Byte	Width	Name	Function
7 – (n+7)	n (0<n<256)	body	The body of the packet is transmitted 'as is' to the intended satellites. The body of a packet cannot start with the escape sequence $$, but it may otherwise contain any bytes.
(n+8) – (n+9)	2	CRC	A two byte CRC created out of the of the packet header and body (excludes pre-amble).

The *start of a packet* must be well understood, since an escape sequence in the middle of a packet is ignored. A packet is transmitted out of a DR2000 whenever:

- `"len"` bytes are received by the DR2000, in which case a packet is sent out immediately after calculating the CRC. Byte number "`len`+1" is considered the start of the next packet.
- There is a pause larger than 2ms between bytes. After the pause the DR2000 calculates the CRC for the small packet and updates the len byte in its actual transmission. The receiver will expect len bytes and extracts the body. Only the number of transmitted bytes are sent out of the receiver, the packet is not padded to complete the `len` specified in the receiver.

Therefore, it is important to ensure that a packet is sent continuously out of a serial port, without pauses of more than 2ms. While the receiving satellite may have timeouts that would allow it to reconstruct a packet that is divided among multiple DR2000 transmissions, there will be twice the DR2000 overhead.

H.1.2 DR2000 Commands

The DR2000 must also be configured to operate in the right mode and frequency prior to operation with the SPHERES system. This initialization occurs automatically in the SPHERES satellites, but must be performed individually in every external element (such as the communications laptop). Table H.2 lists the DR2000 commands that are pertinent to the SPHERES program; Table H.3 lists the valid hardware ID's used in SPHERES.

TABLE H.2 DR2000 configuration commands

Command	Name	Description	SPHERES
$$TOADxx	To address	Set the TO address configuration in FLASH.	`xx = 00`
$$FRADxx	From address	This satellite's address. Table 3-3 lists the valid values in detail.	Ground: `xx = 30` Satellites: `xx = 31 – 39`
$$SIZExx	Maximum packet size	The maximum length of the body before the DR2000 immediately sends out a packet	`xx = 25`
$$RFMDx	DR2000 Mode	Changes the mode of the DR2000 between transparent (x=1) and structured (x=0).	`x=1`
$$RDSPx	RF baud rate	Changes the baud rate of the RF transmissions. Always use x=0 for 57.6kbps	`x=0`

TABLE H.3 Valid satellite IDs for the `to` and `from` fields

Satellite Name	HW Address (hex)
Broadcast	0x00
Laptop/Ground	0x30
SPHERE s/n 1	0x31
SPHERE s/n 2	0x32
SPHERE s/n 3	0x33
SPHERE s/n 4	0x34
SPHERE s/n 5	0x35
SPHERE Default[a]	0x39

a. A SPHERE satellite with corrupted memory will reset its satellite ID to 0x39. The satellite must then be reconfigured with a valid ID between 0x31-0x35.

These commands set the satellites to the default configuration every time they boot. All programs should do the same. After this is done once, there is no need to use these values again. As an example, the sequence used by SPHERE satellite with serial number 1 is:

```
$$RFMD1
$$TOAD00
$$RDSP0
$$SIZE25
$$FRAD31
```

H.2 Link-layer Interface

This section describes the low-level interface to the SPHERES communications system. SPHERES utilizes two communications channels (868MHz and 916MHz) operating at 56.7kbps. Nominally the 868MHz channel is used for satellites-to-laptop (STL) communication, while the 916MHz channel is reserved for satellite-to-satellite (STS) communication. Since the interface to both radios is identical, this channel assignment is strictly a matter of convention and may be reconfigured if necessary (e.g. in the event of a hardware failure). Channel bandwidth is divided between transmitting stations (e.g. satellite and laptop) using a time division multiple access (TDMA) protocol. Relative bandwidth assignments are user definable. Data packets are fixed-length and consist of a structured header and a user-defined payload.

This section first examines the packet format and then examines how this format relates to the TDMA scheme.

H.2.1 SPHERES Packet Structure

The SPHERES communication system is based around a fixed-size packet scheme. The packet structure consists of a 5-byte header and a 32-byte data section. Recent revisions to the DR2000 firmware allow variable-length packets, but most 'standard' packets to date use the common packet length. The packet structure is depicted in Figure H.2. The DR2000 adds its own header (5-byte) and CRC (2-byte) to each packet transmitted by the radio. These components are stripped off of received packets before RS-232 transmission.

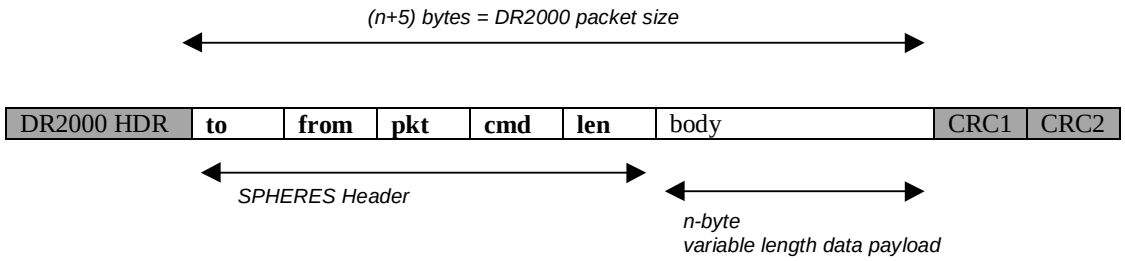

Figure H.2 SPHERES packet structure (n=32)

Important sizes are depicted by arrows in the picture. The base packet length is defined by the body of the packet. We have selected a packet-body size of 32bytes. This gives a reasonable compromise between unused capacity and header overhead. The hardware packet size, set on the DR2000, is five bytes longer than the body to account for the SPHERES packet. header. The header carries information about the packet contents and routing. The header is described in Table H.4.

TABLE H.4 SPHERES header structure

Byte	Length	Field	Description
0	1	to	This contains the receiver hardware ID as seen by each individual SPHERE satellite. Each physical satellite is assigned a unique hardware identifier in the range `0x31-0x35` (hex). Acceptable values are listed in Table 3-3. These addresses must be associated with a 'logical' identity (e.g. SPHERE1, SPHERE2, SPHERE3) in the users program. This allows us to map between logical and hardware addresses, so that any satellite can take on any task.
1	1	from	This bit contains the station ID of the transmitting satellite. The number must meet the same characteristics as described in the to field.

2	1	chk	The checksum field is an 8-bit checksum of the body of the packet, used by the SPHERES system for error detection (but not error correction). The checksum is a simple unsigned sum of the unsigned bytes of the body, truncated to 8 bits. It is calculated as follows:
			`chk = 0;`
			`for (i=0; i<len; i++)`
			`  chk += body[i];`
			`chk = chk & 0xFF;`

TABLE H.4 SPHERES header structure

Byte	Length	Field	Description

3	1	cmd	This is the command field that describes the type of packet. The command field also indicates which channel is being used (868 or 916). Each command byte is structured as follows:
			Bit Description
			0-5 command (range 0-63 decimal, 0x0-0x3F)
			6 indicates whether the packet must be acknowledged. This bit is only used for communications between the Ground station laptop and the SPHERES satellites; no other acknowledgement structure has been implemented.
			7 channel: 0 = 868, 1 = 916
			To build a command byte, one must use the following formula:
			`cmd = CHANNEL + ACK + COMMAND`
			Where:
			`CHANNEL = 0x00 for 868`
			`CHANNEL = 0x80 for 916`
			`ACK = 0x40 if ACK is required`
			`ACK = 0x00 for no ACK`
			`COMMAND = 0x00 – 0x3F`
			Currently defined command assignments are detailed in Section 6 and in the source file `commands.h`.

4	1	len	The length is checked by the receiving satellite to ensure a full packet is processed, and that a new packet is not overwritten if a short packet is transmitted. The length must be set to a value between 0-32, as a 32 byte data size is the longest the DR2000 will allow with the default settings. The default value for len in SPHERES is 32 (0x20).

To send a packet, one first generates a header according to the above structure. This is followed by the n-byte body of the packet. At the link-layer, the format of this section is arbitrary. The header and body must be sent through the serial port to the DR2000. For any standard packet type, unused bytes should still be sent using a filler character. Sending 0x0 or perhaps 0xAA for better bit balancing are good choices.

Inter-packet time must be carefully controlled since the DR2000 does not provide any flow-control information. If packets are sent to the DR2000 too rapidly, data loss will result. There are four stages to each packet transmission (Figure H.3) First, the packet is sent serially from the TX-computer (DSP or CPU) to the TX-DR2000. The packet is then copied to an internal transmit buffer, where the DR2000 header and CRC are added. Next

the packet is transmitted, via wireless, from the TX-DR2000 to the RX-DR2000. The last step involves the serial transfer from the RX-DR2000 to the RX-Computer.

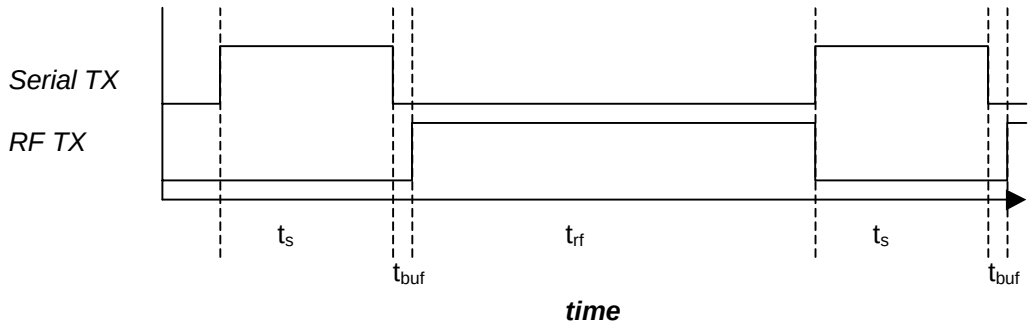

Figure H.3 Packet transmission sequence

To ensure that transmissions do not overlap there must be a minimum time-separation between packets, to ensure the DR2000 buffers are not overwritten. The minimum separation between the start of two packets is:

$$t_{min} = t_s + t_{buf} + t_{rf} \tag{H.1}$$

where t_s is the transmission time between the SPHERES avionics and the DR2000 over the standard UART line, t_{buf} is the buffering time of the DR2000 and t_{rf} is the RF transmit time.

The UART transmission time of an n byte packet is:

$$t_s = \frac{10(n+5)}{115200} \tag{H.2}$$

Each byte sent over a standard UART line is added one stop bit and one start bit, for a total of 10-bits per transmitted byte. The SPHERES header is five bytes long, which must be transmitted over the RS232 line to send the packet. The RS232 line is operated at 115.2kbps. For a packet where n=32 t_s=3.21ms.

The measured buffering time is t_{buf} = 600μs.

For a packet with n bytes in the body, the RF transmission time can be calculated as follows:

$$t_{rf} = \frac{14(n + 15)}{57600} \quad \text{(H.3)}$$

There are a total of 15 header bytes (5 SPHERES header bytes, 3 pre-amble bytes of the DR2000, 5 DR2000 header bytes, and the 2-byte CRC), therefore the (n+15). Further, each byte is converted to a 12-bit word for bit balancing purposes, and added a stop and a start bit, making each word 14-bits long. The RF transmission frequency is 57.6kbps = 57600 bits per second. For the standard SPHERE package where n=32 t_{rf} = 11.42ms.

Therefore, for the SPHERES standard packet, to total time between packet starts must be at least:

$$t_{min} = 3.2ms + 0.6ms + 11.42ms = 15.22ms \quad \text{(H.4)}$$

H.2.2 Time-Division Protocol

Access to the two RF channels must be managed to ensure that transmitting stations do not inadvertently transmit at the same time. Simultaneous transmissions would cause loss of data and degrade the overall productivity of the SPHERES experiment. We have chosen a time-division multiple access scheme for channel management. In such a scheme each station has an assigned time-interval (or window) in which it is permit-ted to transmit. At other times, the station remains in receive mode and must store outgoing packets until its next transmit window. Since most of the communications traffic in our experiments are expected to be predictable and periodic, TDMA is a good access scheme that ensures fair and efficient access to the radio channels.

The SPHERES TDMA scheme is depicted graphically in Figure H.4. The basic unit of time is called a frame. During each frame, all stations are given a chance to transmit. A frame begins with the receipt of a synchronization packet from the laptop. Within a frame,

stations are assigned a transmit window by specifying a start time and a stop time. The frame timers on each satellite will not reset until the next synchronization packet is received.

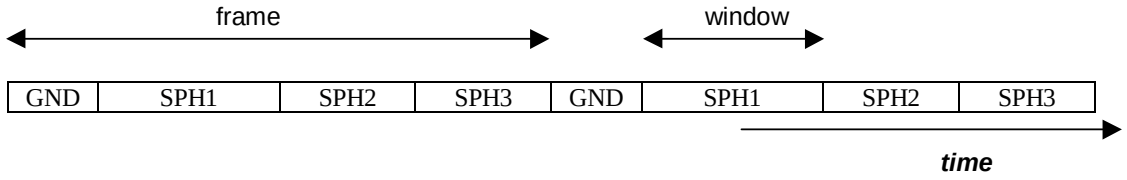

Figure H.4 Time Division Multiple Access scheme

We adopt this synchronization scheme because the we have limited control over the timed behavior of the laptop. Moreover, in tests of the standard windows timer routines, we have observed temporal jitter of up to about 20ms. Making the laptop the master time reference rather than a slave, eliminates the need to worry about synchronizing the laptop to the internal satellites time. Because the laptop dictates the frame size by sending the synchronization packets, and because the laptop software cannot change due to NASA regulations, the frame size has been fixed to 200 ms. Table H.5 lists the standard 200ms frame specification for the STL channel; Table H.6 lists the STS frame.

TABLE H.5 STL standard frame specification (200ms frame)

Number of Stations	Station	Length (ms)	Start (ms)	Stop (ms)
4	SPH1	53	0	53
	SPH2	53	53	106
	SPH3	54	106	160
	GND	40	160	200
3	SPH1	80	0	80
	SPH2	80	80	160
	GND	40	160	200
2	SPH1	160	0	160
	GND	40	160	200

TABLE H.6 STS standard frame specification (200ms frame)

Number of Stations	Station	Length (ms)	Start (ms)	Stop (ms)
3	SPH1	66	0	66
	SPH2	66	66	132
	SPH3	68	132	200
2	SPH1	100	0	100
	SPH2	100	100	200
1[a]	SPH1	200	0	200

a. STS use during single satellite operation is useful if there is a DR2000 that is listening on that channel in order to download telemetry data.

We make the following notes about the frame specification process:

- The satellite ID's are the logical identifications of the SPHERES satellites, and not the hardware ID (i.e., SPHERE1, SPHERE2, etc., rather than hardware ID 0x31, 0x32, etc.).
- To completely describe the frame structure we need to specify the frame length, and start and stop times for each station. These times are offsets from the start of a frame and are measured in units of ms.
- After a frame expires, the next frame will not begin until the next synchronizing packet is received. This satisfies the safety related requirements that will disable the satellites if it loses communication with the laptop.
- Users and guest scientists can specify the frame structure for their experiments. This can change from test to test and is specified in the GUI-ini file provided by the user.
- Stations need not receive equal windows.
- The ground (laptop) station must be the last window on the STL channel. This is to allow for frame synchronization (see below). This window must be at least 40 ms long. If the laptop has additional data to transmit, it should first transmit the extra packets, and then send the synchronization.
- Users are solely responsible for ensuring that their window divisions are consistent. As a matter of convention, users may freely configure the STS channel but should employ the standard STL assignments in most circumstances.

H.2.3 Packet Acknowledgement

The SPHERES core operating system implement a simple packet acknowledgement scheme dedicated to ensure commands from the laptop are received and executed syn-

chronously between multiple satellites. This packet acknowledgement is implemented by using the `cmd` field of the SPHERES header and a byte in the state-of-health (SOH) packet (described below), which is sent to the laptop. While other satellites also see the SOH, and could conceivably use the same system to require inter-satellite acknowledgements this is not recommended. First, the state-of-health packet is only transmitted via STL, therefore only STL acknowledgements are possible. Second, the SPHERES core software does not contain functions that process an acknowledgement, and therefore the user would still need to implement special procedures. Therefore, any other type of packet acknowledgement is the responsibility of the guest scientist.

Figure H.5 illustrates the packet acknowledgement sequence. In order to enable packet acknowledgements bit 7 of the `from` field in the header must be set. When a SPHERE satellite receives a packet that requires an acknowledgement, it will set its bit field (given its logical satellite ID) to 1 in the SOH, and request that the SOH be sent immediately. The SOH will be sent to the laptop in the satellite's next TDMA window. Because the laptop is the only unit expected to send packets that require acknowledgement and because that packet causes the TDMA synchronization, the implemented scheme requires that a satellite reply with the acknowledgement in the same frame. If the laptop does not receive an acknowledgement in the same frame, it will retransmit the command. The laptop tracks the number of acknowledgements expected, and will retransmit the command to all satellites until it receives the required number of replies - all satellites always retransmit the acknowledgements. Note that in this process, there is no memory of the original packet, but rather a new one is created until the satellites respond correctly.

Because a packet acknowledgement is intended to synchronize the SPHERES test times, every time that a SPHERE gets a command to start a test it should request for an acknowledgement. The satellite, in turn, will always reset the test time when a command to start a test is received, even if a test is already in progress. In the case where the satellite is already running a test, the satellite will terminate that test immediately, and restart a test with the new test command received. This behavior ensures that all satellites will be run-

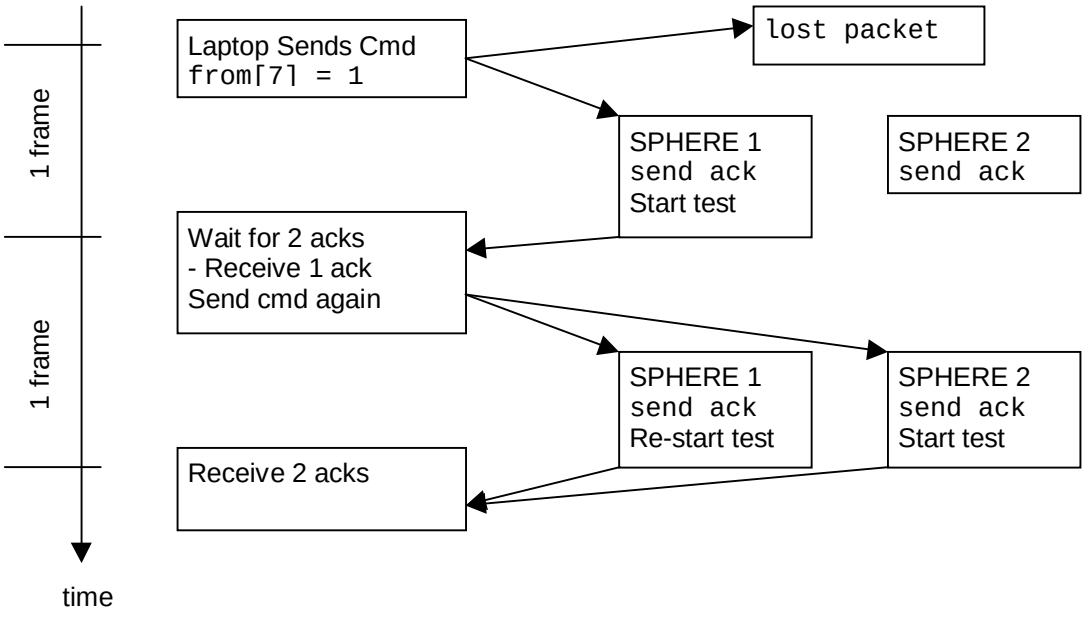

Figure H.5 Packet acknowledgement sequence example (1 lost packet)

ning the same test with the same time (within 1ms of each other) once the acknowledgement process is complete.

H.3 Application Layer Interface

This section describes the different messages that are exchanged between the SPHERES satellites and the laptop base station. All other packages are not interpreted by either the satellites or the laptop in a standard configuration (the user may create message receiver procedures for the satellites, but that is at a higher level interface and is part of the Guest Scientist Program interface, not of the basic communications interfaces).

H.3.1 Transmitted Messages

The laptop must generate two types of packets: General-Purpose Commands (GPC), and Initialization Packets.

H.3.1.1 General Purpose Commands

Figure H.6 presents the contents of a general purpose command packet. These packet act as frame synchronization for the satellites, control test execution, and resets. The laptop broadcasts this packet every 200ms, creating the 200ms TDMA frame size specified above. Because the laptop transmission frequency is fixed, the frame size is restricted to 200ms.

Header:

```
to   = 0x00
from = 0x30
chk  = <checksum>
cmd  = COMM_CMD_GENERALCMD (base + 0x21 = 0x41)
len  = 0x20
```

0	1	2	3	4	5	6	7
Run Time cmd	Run Time unit	unused		Test Number SPHERE 1		Test Control SPHERE 1	

8	9	10	11	12	13	14	15
Test Number SPHERE 2		Test Control SPHERE 2		Test Number SPHERE 3		Test Control SPHERE 3	

16	17	18	19	20	21	22	23
Test Number SPHERE 4		Test Control SPHERE 4		Test Number SPHERE 5		Test Control SPHERE 5	

24	25	26	27	28	29	30	31
Reset SPH s/n 1	Reset SPH s/n 2	Reset SPH s/n 3	Reset SPH s/n 4	Reset SPH s/n 5	STL Sync	STS Sync	Enter Boot Load

Figure H.6 General purpose command structure

The bit fields for the different command types are described in Table H.7.

Starting & Stopping a Test

Sending a start or stop test are mutually exclusive; the SPHERES will check that if a test start command is sent the stop is not present, or vice versa; if both commands are present the packet will be ignored.

TABLE H.7 General purpose command structure details

Byte	Size (bits)	Field	Type	ID[a]	Description
0	1 (8)	run time cmd	unsigned char	LOG	This field is used to send the satellites commands during tests for manual operations of the satellites. This field includes a one-byte `unsigned char` command that will be passed on to the SPHERE controller software even while a test is already in progress.

1	1 (8)	run time satellite	unsigned char	LOG	This field indicates the logical ID of the commanded satellite. Any combination of bits is allowed. The bit fields are:
					`Bit Description`
					`0 SPHERE 1`
					`1 SPHERE 2`
					`2 SPHERE 3`
					`3 SPHERE 4`
					`4 SPHERE 5`
					`5-7 unused`

2-3	2 (16)	unused	-	-	-
4-5	2(16)	test number SPH1	unsigned short	LOG	This two-byte, unsigned integer specifies the test number to start. This field is ignored unless the Start Test bit is set.

6-7	2(16)	test control SPH1	unsigned short	LOG	The test control field is used to start and stop tests as well as to command synchronization of the satellites. All unused bits are reserved for future use.
					`Bit Description`
					`0 reserved`
					`1 Start Test`
					`2 Stop Test`
					`3 Synchronize SPH time`
					`4-15 unused`

8-9	2(16)	test number SPH2	unsigned short	LOG	See above.
10-11	2(16)	test control SPH2	unsigned short	LOG	
12-13	2(16)	test number SPH3	unsigned short	LOG	
14-15	2(16)	test control SPH3	unsigned short	LOG	
16-17	2(16)	test number SPH4	unsigned short	LOG	
18-19	2(16)	test control SPH4	unsigned short	LOG	

TABLE H.7 General purpose command structure details

Byte	Size (bits)	Field	Type	ID[a]	Description
20-21	2(16)	test number SPH5	unsigned short	LOG	
22-23	2(16)	test control SPH5	unsigned short	LOG	

24
 	
25
 	
26
 	
27
 	
28
	1(8)
	
	1(8)
	
	1(8)
	
	1(8)
	
	1(8)
		reset SPH 0x31
		
		reset SPH 0x32
		
		reset SPH 0x33
		
		reset SPH 0x34
		
		reset SPH 0x35
			unsigned char
			
			unsigned char
			
			unsigned char
			
			unsigned char
				HW
				
				HW
				
				HW
				
				HW
					This is a byte field to initiate reset of the satellite. Because resets can cause loss of data or resources, they must be repeated several times in a row before a satellite will process them. The following commands are supported:
					`Bit Description Repeat`
					`0 Vent Tank 2`
					`1 Soft Reset 2`
					`2 Hard Reset 2`
					`3 916 Reset 2`
					`4 868 Reset 2`
					`5 Tank count 2`
					`6-7 unused`

29
 	
30
	1(8)
	
	1(8)
		STL Sync
		
		STS Sync
			unsigned char
			
			unsigned char
				LOG	Synchronize / Enable the corresponding communications channel. This byte must be received to synchronize every frame and in turn enable communications on this channel. The communications channel will not transmit after their initial window until a new Sync bit is received. Each bit is used for a logical SPHERE satellite:
					`Bit Description`
					`0 SPHERE 1`
					`1 SPHERE 2`
					`2 SPHERE 3`
					`3 SPHERE 4`
					`4 SPHERE 5`
					`5-7 unused`

TABLE H.7 General purpose command structure details

Byte	Size (bits)	Field	Type	ID[a]	Description

31	1(8)	Enter Boot Load	unsigned char	HW	This command forces the satellite with corresponding hardware ID to enter boot loader mode. Once the command is received the satellite will not exit boot loader mode until it has been re-programmed with a new program. One bit is assigned to each satellite; any combination of bits may be used, since the boot loader identifies which satellite is being programmed. Note that due to its potential effect to erase the current program, the command must be received 3 times in a row take effect:
					`Bit Description Repeat`
					`0 SPH HW ID 0x31 3`
					`1 SPH HW ID 0x32 3`
					`2 SPH HW ID 0x33 3`
					`3 SPH HW ID 0x34 3`
					`4 SPH HW ID 0x35 3`
					`5-7 unused`

a. The ID field specifies whether the command is destined for the HW - hardware ID or the LOG - logical ID.

All packets that include a test start or test stop must request an acknowledgement as described in Section H.2.3. Therefore, a start/stop test should be re-sent until the satellite acknowledges its receipt. Note that the satellite will acknowledge the packet as soon as its received, regardless of whether the start/stop test is valid at the time the command was received. The satellite sending out the start/stop test is responsible to check the SOH packet to determine if a test actually started or not. For example, if the satellite is currently in "idle" mode and a "test start" command is sent, the satellite will acknowledge the packet as received, but a test will not start because the "enable" button has not been activated.

H.3.1.2 Initialization Packets

Initialization packets are sent to the satellites so that they know the locations of the beacons and the current temperature. Each satellite stores this information in the onboard flash memory. When a satellites boots, it reads the flash and processes the beacon locations saved there. Since the satellites has no way of knowing the age of this data, it sets a

bit in its state-of-health packets to signify that it is using old beacon data and has not received new data since power-on. If the GUI sees that this bit is set, it can generate extra initialization packets for the satellite. Such initialization is optional from the satellite's point of view - it will operate correctly without beacon initialization.

For simplicity, we have adopted the simple but somewhat redundant packet structure presented in Figure H.7 and described in Table H.8.

Header:

```
to   = <any valid address>
from = 0x30
chk  = <checksum>
cmd  = COMM_CMD_BEACON_INFO (base + 0x26 = 0x46)
len  = 0x20
```

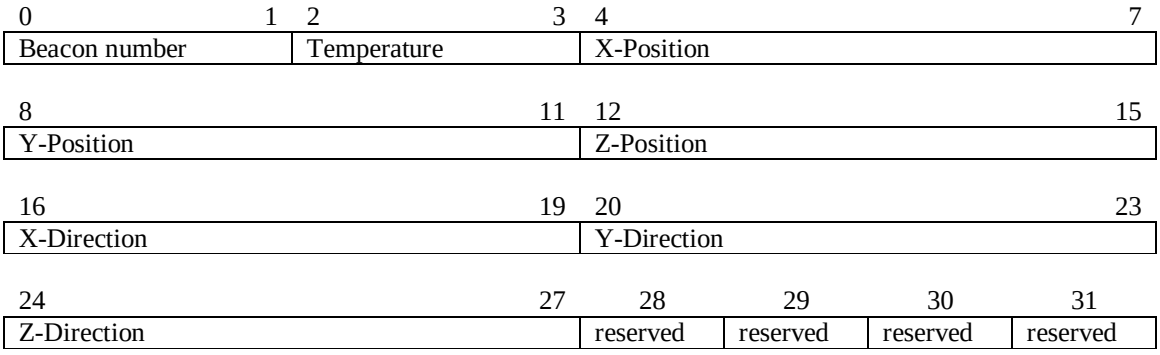

Figure H.7 Initialization packet structure

TABLE H.8 Initialization packet structure details

Byte	Size (bits)	Field	Type	Description
0-1	2(16)	beacon number	unsigned short	This value selects the beacon that the packet applies to. Valid numbers are from 1-6.
2-3	2(16)	temperature	unsigned short	The current temperature specified in tenths of a degree C. (i.e. 22.0°C = 220).

TABLE H.8 Initialization packet structure details

Byte	Size (bits)	Field	Type	Description

4-7	4(32)	X-position	float	4-byte IEEE floating point that specifies the 3D position of the beacon in meters. The value should be cast into an integer in order to transmit it correctly; e.g.:
				`float xPos = 3.187;`
				`int xVal;`
				`xVal =`
				`*((unsigned int *) &xPos);`

8-11	4(32)	Y-position	float	
12-15	4(32)	Z-position	float	
16-19	4(32)	X-direction	float	4-byte IEEE floating point that specifies the three element unit vector direction of the beacon.
20-23	4(32)	Y-direction	float	
24-27	4(32)	Z-direction	float	
28-31	4(32)	unused	-	-

By convention, unused beacons should specify zeros for the direction vector. The direction vector of active beacons should be normalized (i.e. have unity magnitude).

Determination of Beacon Position and Direction

Definitions

- *D* = distance of beacon to seat track
- *P(i)* = (x_i, y_i, z_i) = position of each used seat track hole location with respect to the ISS
- *B(i)* = (x_i, y_i, z_i) = corrected position of each beacon transmitter element
- *q(i)* = (x_{dir-i}, y_{dir-i}, z_{dir-i}) = unit vector direction of each beacon

The following data must be known in order to solve the equations:

- Number of beacons in use
- *P(i)* for each beacon in use
- Gold and black angles for each beacon, from here on referred to as $gold_i$ and $black_i$

Steps to determine position & direction

1. Location Correction

Correct the location of the beacon transmitter itself by adding the offset of the transmitter from the seat track. The correction depends on the location of the seat tracks as per Table H.9

TABLE H.9 Corrections and seat track locations

Track	Correction
Overhead	$Bx_i = Px_i$ $By_i = Py_i$ $Bz_i = Pz_i + D$
Starboard	$Bx_i = Px_i$ $By_i = Py_i + D$ $Bz_i = Pz_i$
Deck	$Bx_i = Px_i$ $By_i = Py_i$ $Bz_i = Pz_i - D$
Port	$Bx_i = Px_i$ $By_i = Py_i - D$ $Bz_i = Pz_i$

This calculates all the *B*(i)'s.

2. Move the frame origin to center of beacon area

   First determine the total expansion of the beacon area; for all beacons find:

$$x_{max} = \max(Bx_i) \tag{H.5}$$

$$x_{min} = \min(Bx_i) \tag{H.6}$$

$$y_{max} = \max(By_i) \tag{H.7}$$

$$y_{min} = \min(By_i) \tag{H.8}$$

$$z_{max} = \max(Bz_i) \tag{H.9}$$

$$z_{min} = \min(Bz_i) \tag{H.10}$$

   Now determine the 'center' of the beacon area:

$$x_{gc} = (x_{max} + x_{min}) / 2 \quad \text{(H.11)}$$

$$y_{gc} = (y_{max} + y_{min}) / 2 \quad \text{(H.12)}$$

$$z_{gc} = (z_{max} + z_{min}) / 2 \quad \text{(H.13)}$$

Now 'center' the beacon locations using the general center locations; for all beacons:

$$B'(i) = (x'_i, y'_i, z'_i) = \quad \text{(H.14)}$$

$$x'_i = x_i - x_{gc} \quad \text{(H.15)}$$

$$y'_i = y_i - y_{gc} \quad \text{(H.16)}$$

$$z'_i = z_i - z_{gc} \quad \text{(H.17)}$$

3. Determine direction with unit vectors

   The vector overhead depends on the beacon location as specified in Table H.10.

TABLE H.10 Unit vector determination

Track	Correction
Overhead Deck	$x_{dir\text{-}i}$ = -sin($gold_i$) * sign(x'_i) $y_{dir\text{-}i}$ = -cos($gold_i$) * sin($black_i$) * sign(y'_i) $z_{dir\text{-}i}$ = -cos($gold_i$) * cos($black_i$) * sign(z'_i)
Starboard Port	$x_{dir\text{-}i}$ = -sin($gold_i$) * sign(x'_i) $y_{dir\text{-}i}$ = -cos($gold_i$) * cos($black_i$) * sign(y'_i) $z_{dir\text{-}i}$ = -cos($gold_i$) * sin($black_i$) * sign(z'_i)

where sign(#) is defined as:

$$\text{if } (\# < 0) = (-1) \quad \text{(H.18)}$$

$$\text{if } (\# >= 0) = (1) \quad \text{(H.19)}$$

4. Transmit Packets

   A total of "i" packets will be transmitted, one for each beacon, the packet should be composed of the bytes described in Table H.11.

TABLE H.11 Beacon location packet structure

Byte	Size (bits)	Field	Type	Value
0-1	2(16)	beacon number	unsigned short	i
2-3	2(16)	temperature	unsigned short	(temp * 10)
4-7	4(32)	X-position	float	B'(i) = x'_i
8-11	4(32)	Y-position	float	y'_i
12-15	4(32)	Z-position	float	z'_i
16-19	4(32)	X-direction	float	$\theta(i) = x_{dir-i}$
20-23	4(32)	Y-direction	float	y_{dir-i}
24-27	4(32)	Z-direction	float	z_{dir-i}

To store the `float` variables into a binary `char` packet the following cast can be used:

```
float x[5], y[5], z[5];
int x_int[5];
unsigned char packet[32];

x[i] = <x final pos of beacon i>;          // float
x_int[i] = *((unsigned int *) &x[i]);      // int
memcopy (&packet[i*4], y[i], sizeof(int)); // unsigned char
```

H.3.2 Received Messages

The laptop must archive all incoming data from the radio to a binary file. Post processing will be used to extract telemetry and experiment results. Although the laptop is free to ignore the majority of this traffic, it must perform some preliminary packet parsing to decode the state-of-health of each active satellite.

H.3.2.1 Packet Parsing

Packets are nominally 37 bytes long (including header), but loss of individual bytes has been observed. Therefore, the laptop should use simple pattern matching on the incoming

data to look for the header, which presents the best pattern to match. Specifically, look for the following byte sequence presented in Table H.12.

TABLE H.12 Packet parsing matching sequence

Header	Value Match
TO	0 0x30-0x39
FROM	0x30-0x39 0xB0-0xB9
PKT	<any>
CMD	<specific cmd>
LEN	0x25

Using only the `TO`, `FROM`, and `LEN`, fields the probability of a false positive match in a random data stream of data is about $2x10^{-6}$. If we consider that we only need to parse the state-of-health packets, we can also match by `CMD` field, and the `TO` field will always be broadcast. This will drop the probability of false matching to about $4x10^{-9}$. This low figure indicates that direct, localized pattern matching (i.e. without memory of past PKT fields, or when the last header was found), should provide sufficient performance.

H.3.2.2 State-of-Health Packets

State of health (SOH) packets are generated by each active satellite at 1Hz. These are added to the onboard message transmit queues, so actual reception time of these messages may vary depending on the current traffic levels. The packets are formatted as presented in Table H.13.

TABLE H.13 State of Health packet structure details

Byte	Size (bits)	Field	Type	Description
0-3	4(32)	Time Stamp	unsigned int	The time is measured in ms. The time is referenced to either the power-on time, or the last time a time-synchronization command was received.
4-7	4(32)	Program ID	unsigned int	This is a unique identifier that is associated with each program file.

TABLE H.13 State of Health packet structure details

Byte	Size (bits)	Field	Type	Description
8-11	4(32)	Tank Usage	unsigned int	This is the aggregate firing-time of the thrusters on the satellite (thruster-milliseconds). This value is divided by the expected total gas availability to provide a percentage estimate of the amount of remaining propellant.
12-15	4(32)	Test Time	unsigned int	Time in ms since the start of the current test.
16-19	4(32)	Maneuver Time	unsigned int	Time in ms since the start of the current maneuver.
20	1(8)	Last Test Result	unsigned char	Indicates the manner in which the last test completed. The following codes are defined: Bit Description 0 No data / test in progress 1 Normal / OK 2 Stop via hardware Enable button 3 Stop via communications Stop Test 4 Stopped due to communications failure 5 Unknown Test 6 Test Timeout 9 Satellite Not Enabled 10-255 User defined
21	1 (8)	Temperature	unsigned char	The temperature set in the SPHERE metrology section. The temperature is specified in tenths of a degree C. (i.e. 22.0°C = 220).
22-23	2(16)	IR Rcv Counter	unsigned short	The number of IR pulses observed since the last reset.
24-25	2(16)	Test Number	unsigned short	Currently running test number. A value of 0 indicates that no test is running
26-27	2(16)	Maneuver Number	unsigned short	Current maneuver number. This value is zero if no test is running.
28	1(8)	Status	unsigned char	This is a bit-field description of the SPHERES status: Bit Description 0 Battery status (1 = OK, 0 = Low) 1 STS Enabled 2 STL Enabled 3 Using old beacon data 4-7 Unused

TABLE H.13 State of Health packet structure details

Byte	Size (bits)	Field	Type	Description

29	1(8)	Operating Mode	unsigned char	This describes the current operating mode of the satellite. The whole byte is used to identify the mode:
				`Value Description`
				`0 Idle`
				`1 Transition (enable button pressed)`
				`2 Position Hold`
				`3 Running a test`
				`4 Suspend (due to lack of RF Sync)`
				`5-255 Unused`

30	1(8)	Satellite Role	unsigned char	Indicates the current logical role performed by the satellite identified in the header by its hardware ID. The SPHERES interpret this value to establish the connection between logical address or role (i.e. SPHEREn) and the hardware address (i.e. the FROM field in the header).
				Must be one of the following values:
				`Value Description`
				`1 SPHERE 1`
				`2 SPHERE 2`
				`3 SPHERE 3`

31	1(8)	Acknowledge-ment	unsigned char	This byte is set to 1 if the satellite is acknowledging a command received in the previous General Command Packet received. It is zero if this is not an acknowledgement.

H.3.2.3 Background Telemetry

Active satellites automatically and periodically broadcast their onboard state estimates. It is unlikely that the GUI will have to parse these packets, but the packet definition is included in Figure H.8 and Table H.14 for completeness.

The telemetry packet use 16-bit short integers to transmit all the data, therefore these numbers are scaled using the factors presented in Table H.15. Multiply the received short integer by its factor to obtain the original values.

Header:

```
to   = 0 (Broadcast)
from = <unit hardware ID>
chk  = <checksum>
cmd  = COMM_CMD_TELEMETRY (base + 0x1B = 0x3B)
len  = 0x20
```

0 — 2	3	4 — 5	6 — 7
Time Stamp	Unit role	X-Position	Y-Position
8 — 9	**10 — 11**	**12 — 13**	**14 — 15**
Z-Position	X-Velocity	Y-Velocity	Z-Velocity
16 — 17	**18 — 19**	**20 — 21**	**22 — 23**
ε_1 quaternion	ε_2 quaternion	ε_3 quaternion	η quaternion
24 — 25	**26 — 27**	**28 — 29**	**30 — 31**
ω_x angular velocity	ω_y angular velocity	ω_z angular velocity	unused

Figure H.8 Telemetry packet structure

TABLE H.14 Telemetry packet structure details

Byte	Size (bits)	Field	Type	Description
0-2	3(24)	time stamp	unsigned int	Current satellite time in ms. Note the most significant byte of the unsigned integer must be masked off to allow the role to be stored there.
3	1(8)	satellite role	unsigned char	The current role of the satellite as described in Table 6-4.
4-5	2(16)	X-position	short	The position of the satellite with respect to the global frame. A 16bit signed short indicating the value. The signed short is created by dividing the original float using the factors presented in Table 6-5.
6-7	2(16)	Y-position	short	
8-9	2(16)	Z-position	short	
10-11	2(16)	X-velocity	short	The velocity of the satellite with respect to the global frame, scaled as above.
12-13	2(16)	Y-velocity	short	
14-15	2(16)	Z-velocity	short	

TABLE H.14 Telemetry packet structure details

Byte	Size (bits)	Field	Type	Description
16-17	2(16)	ε_1 quaternion	short	The quaternion components that describe the attitude of the satellite with respect to the global frame, scaled as above.
18-19	2(16)	ε_1 quaternion	short	
20-21	2(16)	ε_1 quaternion	short	
22-23	2(16)	η quaternion	short	
24-25	2(16)	ω_x rate	short	The angular velocities of the satellite with respect to the global frame of reference, scaled as above.
26-27	2(16)	ω_x rate	short	
28-29	2(16)	ω_x rate	short	
30-31	2(16)	unused	-	-

TABLE H.15 Telemetry data conversion factors

Element	Units	Maximum (`0x7FFF`)	Function	Factor
Position	m	3.5m	3.5[m] / 32767[b] =	.00010681 [m]/bit
Velocity	m/s	1.0m/s	1.0[m] / 32767[b] =	.000030519 [m/s]/bit
Quaternion	-	1.0	1.0 / 32767[b] =	.000030519 [ ]/bit
Angular Velocity	rad/s	1.5rad/s	1.5[rad/s] / 32767[b] =	.000045778 [rad/s]/bit