Canadian Space Agency – Space Science and Technology **Stratos PRISM TM/TC Interface Specifications**	
Program:	Stratos
Project:	PRISM (Payload Remote Interface, Sensors Suite & Mass Memory)
Title:	Technical Note - Stratos PRISM Telemetry and Telecommands (TM/TC) Interface Specifications
Doc. Number:	**TN2018-02-PRISM**
Date of 1st release:	September 29th, 2017
Date of current release:	January 16, 2019
Author:	JF Cusson, Senior Engineer – Software & Digital Electronics James Lee, Engineer – Spacecraft Control Systems Ken Lee, FSWEP internship CSA (Mechanical Engineering Master's Thesis, McGill University)
Reviewed by:	Claude Brunet, Patrice Cote (CSA)
Approved by:	JF Cusson
Release:	2

1 SCOPE

1.1 Introduction

As part of the Engineering Capability Demonstration (ECD) program, the Canadian Space Agency (CSA) is designing and building a flight subsystem supporting science payloads onboard stratospheric balloons, to provide real-time data, such as time, gondola position and orientation, environment etc., as well as offering services like data relay to ground (i.e. forwarding to PASTIS) and mass-memory storage. This sub-system is identified as the Stratos Payload Remote Interface, Sensor Suite and Mass Memory (PRISM).

1.2 Identification

This technical note provides detailed formatting information related to the telemetry generated by the PRISM sub-system, as well as the required formatting to interact with the PRISM over telemetry and telecommands channels, and related mechanical/electrical interfaces.

1.3 System Overview

Figure 1 shows the PRISM as part of a typical setup on a stratospheric balloon gondola.

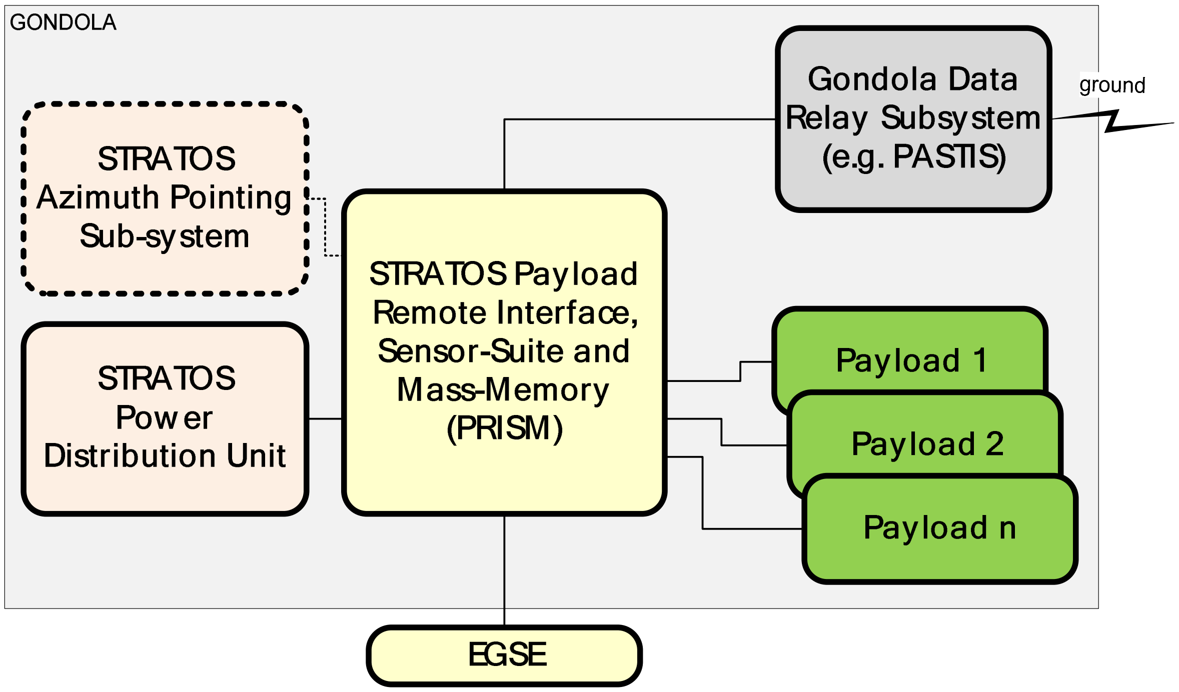

GONDOLA
Gondola Data Relay Subsystem (e.g. PASTIS)
ground
STRATOS Azimuth Pointing Sub-system
STRATOS Payload Remote Interface, Sensor-Suite and Mass-Memory (PRISM)
STRATOS Power Distribution Unit
Payload 1
Payload 2
Payload n
EGSE

Figure 1: Typical Gondola Setup for PRISM

1.4 Document Overview

Section 2 details the mechanical and electrical interfaces, related to the data communication links used to exchange telemetry and telecommands with the PRISM. Section 3 defines the format of the telemetry/telecommands messages.

1.5 Acronyms

AD	Applicable Documents
CPU	Central Processing Unit
CSA	Canadian Space Agency
ECD	Engineering Capability Demonstration
GPS	Global Positioning System
MSL	Mean Sea Level
NED	North East Down
NMEA	National Marine Electronics Association
NTP	Network Time Protocol
PRISM	Payload Remote Interface, Sensor Suite and Mass Memory Sub-system
RD	Reference Document
SI	Système International
SW	Software
TBC	To Be Confirmed
TBD	To Be Determined

TC	Telecommands
TM	Telemetry
TMTC	Telemetry and Telecommands
UTC	Universal Time Coordinate
WGS84	World Geodetic System of 1984

1.6 Applicable Documents

AD No.	Document No.	Document Title	Rev. No.	Date
1.	CSA-STRATOS-RD-0005	Stratos Gondola System Requirements Document	TBD	TBD
2.	CSA-STRATOS-RD-0004	Stratos Gondola Equipment General Design and Interfaces Requirements (GDIR)	TBD	TBD
3.	CSA-STRATOS-RD-0012	Stratos Gondola Equipment PRISM Sub-System High-Level Requirements Specification	Rel.A	June, 2018

1.7 Reference Documents

RD No.	Document No.	Document Title	Rev. No.	Date
1.	n/a			
2.				

2 Mechanical and Electrical Interfaces

Science payloads and other sub-systems are connected to the PRISM via Ethernet or serial data links (i.e. RS-422, RS-485, RS-232).

Planned connectors available for payloads on the PRISM front panel, for telemetry/tele-commands:

- Ethernet ports, using DB9F connectors (pinout compatible with PASTIS). 10 Mbits/s.
- Serial ports, also using DB9F. Serial standard software selectable: RS232, RS422 or RS485 (slave or master). Pinout will vary depending on the selected standard.

3 Telemetry and Telecommands Formatting

3.1 General requirement

1. One general format, the same for telemetry and telecommands, so that everything could be parsed from the same input.
2. Self-contained: No need to rely on previous packet to understand the meaning of a value (e.g. no delta values).
3. A mechanism is needed to de-multiplex the data depending on its origin: on ground, we will need to route packets from the payloads to the right scientists. Therefore, a field shall allow for identification of the source.
4. Each packet needs to be time-stamped with the official on-board mission time, in addition to the time-stamp of the originated sub-system.
5. Each telemetry packet shall uniquely identify its data fields with a single identifier, such that they can be decoded easily.
6. Each command packet shall have an identifier that starts with “CMD”, and include a command count and a checksum on the value of its parameters.
7. When applicable, the checksum should be calculated and located such as to minimize the impact on the receiving onboard process.

3.2 General Formatting

Telemetry and telecommands packets will be formatted as an ASCII string containing printable characters (i.e. codes 0x20 to 0x7E inclusively, expressed here in hexadecimal) listing comma separated values (CSV), with the following conventions:

1. **PACKET PROVENANCE (SOURCE) – *PRISM ONLY, SCIENCE PAYLOAD TO LEAVE EMPTY*:** String will be prefaced with a keyword as a “magic cookie” to identify the stream. IMPORTANT: This identifier will be generated ONLY by the PRISM, any subsystem or science payload sending a packet to the PRISM shall leave this field empty (i.e. the packet will start with a comma). The PRISM will then populate it with an identifier related to the interface port used, e.g. `CHAN-2` if the packet was received on the second Ethernet port. When the packet will be generated by the PRISM itself, the SOURCE will be `PRISM` (TBC);

2. **MISSION TIME STAMP - *PRISM ONLY, SCIENCE PAYLOAD TO LEAVE EMPTY*:** A date-time (UTC) field will follow, formatted ISO-8601 *EXCEPT FOR THE "T" DELIMITER to be replaced by a space*, as: `yyyy-mm-dd hh:mm:ss.sss`. The time-zone indicator (i.e. training "Z") is to be omitted, ***as all timestamp shall be interpreted as UTC***, and NOT local. The MISSION TIME STAMP will be filled by the PRISM itself, from its onboard clock synchronized from GPS UTC. Any subsystem or science payload generating telemetry destined to the PRISM shall leave this field empty.

3. **SUB-SYSTEM TIME STAMP:** A date-time (UTC) field will follow, formatted ISO-8601 *EXCEPT FOR THE "T" DELIMITER to be replaced by a space*, as: `yyyy-mm-dd hh:mm:ss.sss`. The time-zone indicator (i.e. training "Z") is to be omitted, ***as all timestamp shall be interpreted as UTC***, and NOT local. Filled by the sub-system/science payload generating the packet. Having a MISSION TIME versus a SUB-SYSTEM TIME will allow synchronization of data relative to its generator system (e.g. science payload) or more globally with all other systems interfacing with the PRISM, which maintains an official mission time. Note that in the case of a packet generated by the PRISM itself, the SUB-SYSTEM TIME might be left empty, because it is identical to the MISSION TIME.

4. **PACKET ID (i.e. the "leading identifier or packet ID")**: The next item in the packet will be the PACKET ID, which uniquely identifies the content of the packet;

5. **SOFTWARE ID**: Some telemetry and all commands include this field. It uniquely identifies the software to which the command is destined to, or the software that generated the telemetry.

6. **LIST FORMAT:** Values will be comma separated;

7. **BINARY DATA:** Binary data dumps (e.g. parts of pictures or memory dumps) shall be text encoded using the Base64 representation (radix-64);

8. **VALUE REPRESENTATION:** Data values other than date or binary dumps will be in any format acceptable to the ANSI C string-to-double function strtod(3). "`inf`" and "`nan`" are acceptable.

9. **LEADING ZEROS:** Except when noted otherwise, numeric values will not be represented with leading zeros (e.g. "`82`" and NOT "`0082`"). A notable exception is within the time stamps, where month, day, hours etc… all must use leading zeros to keep the length of the time-stamp standard (e.g. "`2018-03-20 09:30.04.001`")

10. **EMPTY FIELDS:** Field not supplied or not available will be left empty (i.e. continuous commas);

11. **TERMINATION:** String will be terminated by \r\n (carriage return – ASCII hexadecimal `0x0D`, line feed – ASCII `0x0A`);

Note that the tele-commands adhere (i.e. is compatible with) this format, but includes more fields in their standard header.

The complete and exact list of items composing a packet shall be unique per its packet ID, for any packet generated by the PRISM. In general, packets create by other subsystems and science payload should also adhere to this specification, to simplify decoding and displaying.

Example 1:

```
PRISM,2017-04-08 03:12:26.908,,GPS1,MODE_AIR,31235.0,2347.97,S,13352.96,E,558.4,M
```

Where :

- `PRISM` is the SOURCE, in this case specifying that the PRISM generated the telemetry contained in this packet
- `2017-04-08 03:12:26.908` Is the MISSION time stamp;
- The SUB-SYSTEM time stamp is not provided, because identical to the MISSION time (since the PRISM itself generated this telemetry packet).
- `GPS1` is the packet ID;
- The rest is a list of telemetry values, specific to this packet; and
- The string is terminated by 2 characters, 0x0D and 0x0A (non printable).

Example 2:

In this example, we'll consider a science payload sending telemetry to the PRISM via an Ethernet connection (connected to physical port #2). Originally, the science payload would generate a packet similar to the following, and send it to the PRISM:

```
,,2017-04-08 05:10:02.003,SPECTR,0,33.8,,,44.0,V,35.007
```

Where :

- There are 2 empty field at the start of the packet;
- `2017-04-08 03:12:26.908` Is the SUBSYSTEM (i.e. science payload) time stamp;
- `SPECTR` is the packet ID;
- The rest is a list of telemetry values, specific to this packet (note that some fields are empty, specifying that these values were not available); and
- The string is terminated by 2 characters, 0x0D and 0x0A (non printable).

The PRISM, upon receiving the packet on channel #2 (e.g. Ethernet port #2), adds the SOURCE (i.e. `CHAN-2`) and the MISSION TIME in the first two empty fields, which leads to the following packet being relayed to the ground and saved in the PRISM mass memory:

```
CHAN-2,2017-04-08 05:10:00.900,2017-04-08 05:10:02.003,SPECTR,0,33.8,,,44.0,V,35.007
```

By adding the SOURCE, the ground equipment will be able to efficiently demultiplex telemetry per science team and other subsystems.

3.3 SOURCE (SRC) Field

Possible values:

- “**SWNAV**”: From the SWNAV process executing on NAVEM computer
- “**PRISM**”: AHR0 and POS0 multicast messages to instruments from the SWNAV process executing on NAVEM computer
- “**SW_EM**”: From the SWEM process executing on NAVEM computer
- “**IOCTL**”: From the I/O Controller
- “**SWCDH**”: From the CDH process executing on the CDH computer
- “**GPS01**”: From the NovAtel GPS receiver, as collected by SWCDH
- “**UPLNK**”: Reserved for uplinked packets
- “**PLDxx**”: (xx=00 to 99) Reserved for payloads serviced by the PRISM
- “**AUXxx**”: (xx=00 to 99) Reserved for peripherals connected to the PRISM

Note that the source field has a fixed length of 5 characters, for alignment purposes.

For backward compatibility with AUSTRAL2017 campaign, the following values shall NOT be used for PACKET SOURCE:

- DAM_RTCLK
- DAM_HKPNG
- DAM_TEMPS
- DAM_HARDW
- DAM_UBLOX
- DAM_NOVATEL
- DAM_TC
- DAM_IMG
- DAM_IM2
- DLOG_CAM
- BM1_ADU
- BM2_ADU

- BM1_TLM
- BM2_TLM
- BM1_HKP
- BM2_HKP
- ACK
- NACK

3.4 SOFTWARE IDENTIFIER (SW-ID) Field

Possible values:

- “**SWNAV**”: The SWNAV process executing on NAVEM computer
- “**SW_EM**”: The SWEM process executing on NAVEM computer
- “**IOCTL**”: The main software of the I/O Controller
- “**SWCDH**”: The CDH process executing on the CDH computer
- “**GPS01**”: The NovAtel GPS receiver

Note that the software identifier field has a fixed length of 5 characters, for alignment purposes.

3.5 Telemetry Formatting

3.5.1 Periodic and One-Time (OT) Telemetry Format

```
[src],[m-time],[ss-time],[tm-id],[field 1],[field 2],…,[field n]
```

Note: "**EVENT**", "**ACK**" and "**NACK**" are reserved telemetry identifiers (i.e. `[tm-id]` cannot be set to these values for general periodic and one-time telemetry).

3.5.2 Event Telemetry

```
[src],[m-time],[ss-time],EVENT,[event string]
```

3.5.3 Command Acknowledgment

```
[src],[m-time],[ss-time],ACK/NACK,[sw-id],[cnt],[cmd-id],[command parameters/message]
```

Where:

- **`src`**: The source, relative to the PRISM, of the packet. INSERTED BY THE C&DH. See "SOURCE Field" above, for possible values.
- **`m-time`**: The mission time. INSERTED BY THE C&DH
- **`ss-time`**: The sub-system time.
- **`ACK/NACK`**: Fixed identifier. ACK for a positive acknowledgment, NACK otherwise.

- **sw-id**: The identifier of the software that processed the command that is acknowledged. See the “SOFTWARE IDENTIFIER” section above, for possible values.
- **cnt**: Command counter. The source process should increment this counter. The goal is to have a different value of this counter for several successive commands such that acknowledgments can be identified, not necessarily to verify the sequence.
- **cmd-id**: Identification of the command. Example: “PING”.
- **Command parameters/message**: An optional field. Could be empty or could be replaced by a generic message (e.g. provide the software version in case of a PING, or the cause of a negative acknowledgment NACK).

3.6 Command Formatting

```
[src],[m-time],[ss-time],CMD,[sw-id],[cnt],[chk],[cmd-id],[command parameters…]
```

Where:

- **src**: The source, relative to the PRISM, of the packet. INSERTED BY THE C&DH. See “SOURCE Field” above, for possible values.
- **m-time**: The mission time. INSERTED BY THE C&DH
- **ss-time**: The sub-system time.
- **CMD**: Fixed identifier. Note that in the event that the command format would change, this identifier could have a version number attached to it (e.g. CMD2, meaning “this is a command, with format #2”). So it is compulsory for the receiving process to compare with the full string (i.e. receivedCommand.equals(“CMD”)) instead of verifying only its start (i.e. receivedCommand.startsWith(“CMD”)).
- **sw-id**: The identifier of the software to which this command is destined to. A process should ignore commands with another destination than itself. See the “SOFTWARE IDENTIFIER” section above, for possible values.
- **cnt**: Command counter. The source process should increment this counter. The goal is to have a different value of this counter for several successive commands such that acknowledgments can be identified, not necessarily to verify the sequence.
- **chk**: Checksum. See “COMMAND CHECKSUM” section below for more information.
- **cmd-id**: Identification of the command. Example: “PING”.

- **`Command parameters`**…: comma separated values, providing command parameters.

Note that the full command packet string will be terminated by the two characters CR-LF.

3.7 Command Checksum

- Calculated by adding the byte value of each character (e.g. “PING” = 80+73+78+71 = 302).
- Calculated on the substring AFTER the 6-fields header “`[`*`src`*`],[`*`m-time`*`],CMD,[ss-time],[cnt],[chk],`”
- Taking into consideration ONLY ASCII character, i.e. not including the line termination (carriage return-line feed) nor any null character. ***Also excluding SPACE character (ASCII decimal 32)***.
- See appendix : Calculating Checksum for the actual algorithm.

4 Telemetry and Telecommands Procedures

4.1 Periodic Telemetry

Sent at a specific rate, continually. Can be turned OFF. Controlled by setting parameters in the generating software. The packet identifier determine the content of each telemetry packet.

4.2 One-Time (OT) Telemetry

Sent upon request, or triggered by specific events. Again, the packet identifier determine the content of each telemetry packet.

4.3 Event Telemetry

An event telemetry only provide a message string, and does not contain specific comma separated fields after the packet header. The messages will be displayed in a special window on ground, and are expected to require the attention of the operator. An event message is not necessarily associated with an anomaly.

Example:

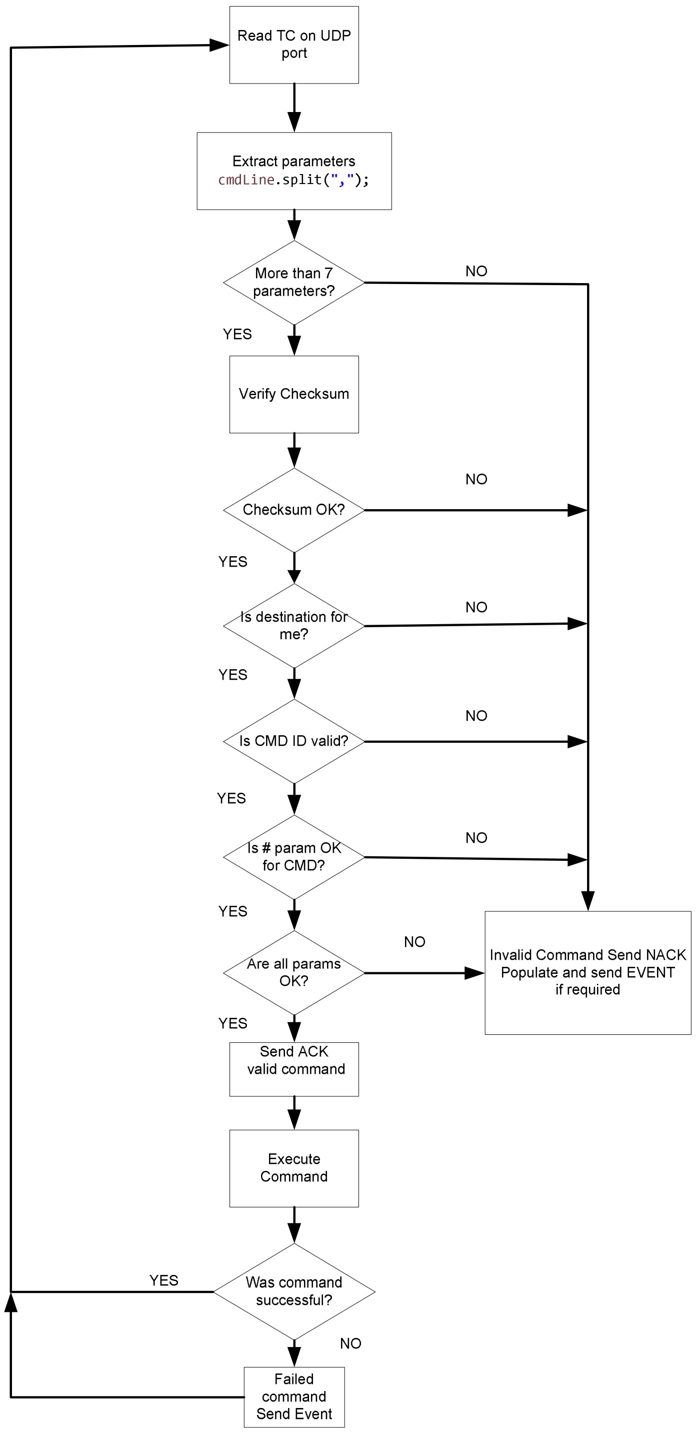
```
SWCDH,2017-04-08T05:10:00.900,,EVENT,FLIGHT PHASE changed from  ASCENT to CEILING
```

4.4 Command Acknowledgment

See the “COMMAND FLOW” section below.

4.5 Command Flow

Each command received will be decoded and validated. Upon the end of the command validation process, a proper ACK or NACK reply will be sent back. If there is an issue while executing the command, an EVENT telemetry might be generated to report it.



APPENDIX: CRC Calculation

Note: In the 2018 implementation, there were two different way of calculating the CRC. SW-CDH and SW-NAV did not use the way described in this document, using an unsigned byte to keep the CRC result between 0-255, while SW-EM did not restrict. The CRC will be standardized in future releases of the system.

SW-NAV CRC Calculation:

```
/*========================================================================*/
/*========================================================================*/
/**
 * calculateChecksum()
 *
 * This method calculates the telemetry message checksum, used to verify telecommands
 *
 * @param commanId Sring Command name
 * @param params String[] array of command parameters
 * @return byte checksum the computed checksum
 * */
public  byte calculateChecksum( String commandId, String[] params ) {
/*========================================================================*/
    byte checksum = 0;
    byte[] b = commandId.getBytes();
    if( b != null ) {
        for( int j=0; j<b.length; j++ ) {
            int byteValueUnsigned = (int)(b[j]) & 0xFF; //probably not necessary
            if( (byteValueUnsigned > 32) && (byteValueUnsigned < 127) ) {
                checksum += b[j];
            }
        }
    }
    for( String param : params ) {
        b = param.getBytes();
        if( b != null ) {
            for( int j=0; j<b.length; j++ ) {
                int byteValueUnsigned = (int)(b[j]) & 0xFF; //probably not necessary
                if( (byteValueUnsigned > 32) && (byteValueUnsigned < 127) ) {
                    checksum += b[j];
                }
            }
        }
    }
    return( checksum );
}
```