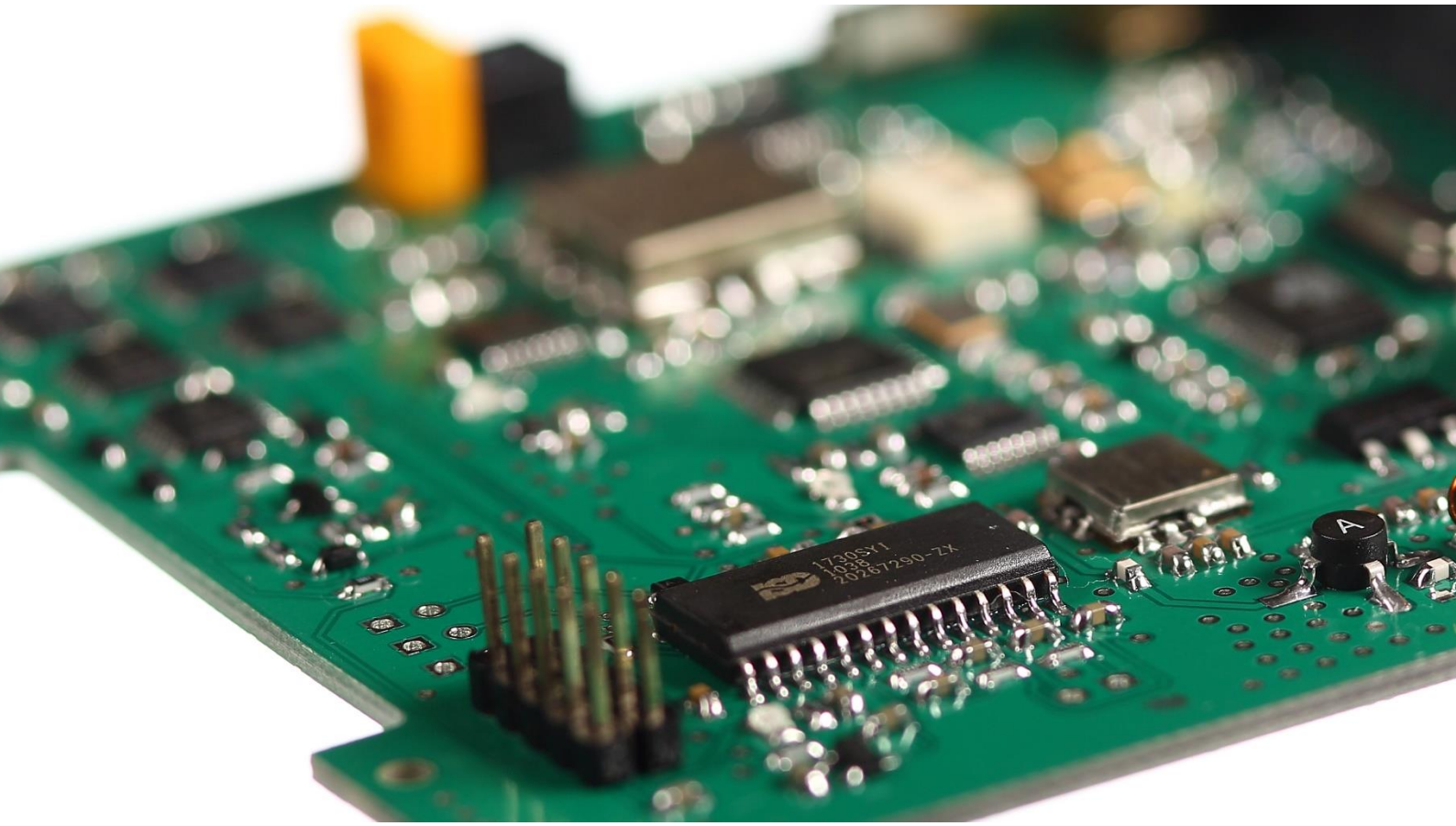
Validation date: **23 November 2017**
Prepared: **Rimantas Žičkus, Lead Software Engineer**
Revised: **Jesper A. Larsen, Lead Systems Engineer**
Approved: **Ernestas Kalabuckas, CTO**

M6P Platform Software Interface Control Document



Point of Contact: **Vytenis Buzas, CEO**
Address: **Mokslininku str. 2A, LT-08412 Vilnius, Lithuania**
Mobile: **+370 663 53355**
E-mail: **vytenis@n-avionics.com**

Table of Contents

1 Software Overview

1.1 Software Configuration

The M6P Platform (Platform) is delivered in full package software configuration. The Platform with integrated Payload is operated fully by the client. The list of all commands and operation descriptions of the satellite are provided in the form of ICDs.

The implemented software features allow the Customer to save integration time and reduce mission risks. They are capable of maintaining the platform and are flexible to build additional functionality and features specific to the Customer's mission. Each configuration of the satellite Platform Software (SPS) covers:

- Flight computer (FC) (includes attitude control and determination (ADCS) subsystem;
- Payload controller (PC);
- UHF radio subsystem (COMM);
- Electrical power system (EPS);
- Propulsion (engine) control unit (ECU) subsystem.

All configurable functions and telemetry of containing subsystems are accessible through CSP protocol over CAN bus during the mission and text console over UART during development. SPS API features include:

- Platform telemetry acquisition and storage;
- Platform configuration;
- Real-time clock;
- Hardware diagnostic;
- Attitude determination and control;
- Position estimation from TLE;
- Magnetic, sun and Earth horizon vectors estimation;
- UHF radio downlink/uplink;
- CSP protocol support;
- Power distribution control;
- File transfer;
- Sending of beacon packets;
- Collecting and sending of real-time telemetry packets;
- Execution of scripts;
- Configuration of SPS by NanoAvionics for each mission to include additional functions, if requested by the Customer;
- NanoAvionics' integrated Command Line Interface (CLI), which allows fast diagnostic out-of-the box, configuration and telemetry readout using a human-friendly text-command interface.

1.2 Mission Code Development

A firmware for satellite platform FC, COMM, EPS and ECU subsystems comes flashed and ready to use or in compiled image files of HEX or similar format. NanoAvionics provides full access to the Payload controller, including flashing and debugging for the Customer, to be built for a particular single-Payload mission.

The Customer is responsible for the Payload software development and implementation of NanoAvionics-provided requirements, which are listed in the following chapters.

Also, this document contains instructions for the Customer to initiate the collection of telemetry, and configuration settings of all M6P subsystems over the CSP network. The Customer's Payload mission software can request all available satellite platform data covering EPS telemetry to orientation and position in space.

If additional PC software development and access to the PC source code is needed, custom (NRE) modifications under separate agreement with the Customer are performed to meet the Customers' requirements.

Flashing and debugging of the PC requires a Serial Wire Debug (SWD) debugger. NanoAvionics recommends ST LINK/V2 as a debugger option and KEIL uVision as an integrated development environment (IDE). The STLINK/V2 debugger tool can be provided to the Customer together with the platform if required. The Customer is not limited to the use of the suggested IDE and debugging tools; nevertheless, NanoAvionics can only provide support in setting up the recommended IDE and debugger.

1.3 Attitude Determination and Control Software

NanoAvionics' Attitude Determination and Control System (ADCS) framework is a software package that implements algorithms and functions required to facilitate attitude determination and control of the satellite, employing a magnetometer, inertial sensors and sun sensors for attitude determination. Reaction wheels and magnetorquers are employed for actuation. The information below provides an overview of the different elements included in the ADCS on-board firmware:

- Orbit propagation;
- Geomagnetic field model;
- Eclipse and sun position;
- Spacecraft dynamics and kinematics;
- Sensors calibration;
- Kalman filter (position and attitude determination);
- Control algorithm;
- B-dot for detumbling.

The ADCS code will start up in detumbling mode until upload of the satellite two-line element (TLE) is performed during the satellite commissioning or acquisition of the GPS signal, before initiating more advanced control modes. The code under this framework will be tailored and improved by NanoAvionics to support the specific mission tasks, including specific modes to reach optimal capabilities of the ADCS according to the particular mission case. The control modes are distinguished as following:

- Detumble mode;
- Velocity vector pointing mode;
- Nadir pointing mode;
- Sun maximum power tracking;
- Earth target tracking according to geodetic coordinates;
- User supplied (ECI) vector tracking.

Detumbling is performed by use of magnetorquers, which interact with the Earth's magnetic field to create a torque. This will eventually slow the spacecraft's angular velocity and damp oscillations down to a point where a transition to a more advanced attitude control mode is made. Determination is performed by collecting data from the following sensors:

- NanoAvionics Inertial and magnetic sensors system;
- Sun sensors;
- Gyros;
- GPS.

The sensors' data is fused by the Kalman filter, determining the spacecraft's position and attitude. An obtained attitude quaternion is used as input data for attitude control algorithms, which drive the reaction wheels to achieve the required pointing direction. Magnetorquers are also used to desaturate the reaction wheels.

1.4 Coordinate Reference Frames

- **ECI.** Earth-centered inertial (J2000.0 - Earth's mean equator and equinox at 12:00 terrestrial time on 1 January 2000). Axis:
  - X: is aligned with the mean equinox;
  - Z: is aligned with the Earth's spin axis or celestial North Pole;
  - Y: chosen so that the (X, Y, Z) trihedral is orthogonal.
- **ECEF.** Earth centered, Earth fixed. Axis:
  - X: intersects the sphere of the Earth at 0 degrees latitude (equator) and 0 degrees longitude (Greenwich);
  - Z: is aligned with the Earth's spin axis;
  - Y: chosen so that the (X, Y, Z) trihedral is orthogonal.
- **BCF.** Body centered, body fixed. Satellite-fixed reference frame. Origin in center of mass. Axis:
  - Z: forward, parallel to 300 mm satellite edge, in direction opposite to propulsion exhaust face;
  - X: up (when tuna can with battery is high), parallel to 200 mm satellite edge;
  - Y: right, chosen so that the (X, Y, Z) trihedral is orthogonal.
- **LVLH.** Local vertical, local horizontal. Local orbital frame. Origin in center of mass. Axis:
  - Z: points to Earth's center;
  - Y: normal to the orbit plane, opposite of the orbit angular momentum vector;
  - X: completes the right-handed orthogonal system and is positive in the vehicle's direction of motion.
- **TNW.** Tangential normal omega (Greek omega denoting the axis of angular momentum). Local orbital frame. Origin in center of mass. Axis:
  - X: collinear to absolute orbital velocity;
  - Y: normal to the orbit plane, same direction as the orbit angular momentum vector;
  - Z: chosen so that the (X, Y, Z) trihedral is direct (Points to direction close to nadir).

1.5 Script Definitions

Mission controller translates script command into target quaternion and reference frame. It passes this information to low-level ADCS control module. Mission controller script input can be described as follows:

Input = *adcs CommandStartTime CommandEndTime* [*AttitudeDefinition*];

adcs is a command that initiates the ADCS script procedure that is followed by these parameters:

- **CommandStartTime.** This is a parameter that defines the command start time in UTC unix time floating-number format.
- **CommandEndTime.** This is a parameter that defines the command end time in UTC unix time floating-number format.

AttitudeDefinition is a parameter group that is defined by the required coordinate reference frame and is changed in the mission controller script input by either:

- AttitudeECI = *PointECI* **Quaternion**
- AttitudeLVLH = *PointLVLH* **Quaternion**
- AttitudeTNW = *PointTNW* **Quaternion**
- AttitudeGEO **=** *PointGEO* **Geolocation VectorVBF**
  - GeoLocation = ***GeodeticLatitudeDeg LongitudeDeg AltitudeAboveGeoidMeters***

Quaternion is defined by quaternion floating numbers (r, i, j, k), VectorVBF is defined by coordinate floating numbers (x, y, and z) and GeoLocation has the format of geodetic-position floating numbers.

The following are the examples to better represent and to help understand the mission controller command:

- ***adcs 1577836800 1577836920 pointECI 1 0 0 0*** (2020-01-01T00:00:00Z set satellite attitude with BCF ref frame matching ECI ref frame. Keep pointing for 2 min.)
- ***adcs 1577836800 1577836920 pointECI 0.707106 0 0 -0.707106*** *(*Same timing. Set satellite attitude with y-axis aligned with the mean equinox, z-axis aligned with celestial North Pole)
- ***adcs 1577836800 1577836920 pointLVLH 1 0 0 0*** *(*Same timing. Set satellite attitude with BCF ref frame matching LVLH ref frame)
- ***adcs 1577836800 1577836920 pointGEO 54.6872 25.2797 100 0 0 1*** (Same timing. Point satellite z-axis to Vilnius geodetic location: 54.6872°N, 25.2797°E, 100 m above sea)

1.6 CubeSat Space Protocol (CSP)

The majority of the Platform subsystems communicate using the CSP – an efficient, open-source protocol stack developed specifically for network-centric nano-satellite systems. CSP is a small network-layer delivery protocol designed for nano-satellites. The protocol is based on a 32-bit header containing both network and transport layer information. The implementation is written in C and is ported to run on any thread-based operating system, such as FreeRTOS, Linux or Windows.

The CSP enables distributed embedded systems to deploy a service-oriented network topology. The layering of CSP corresponds to the same layers as the TCP/IP model. The implementation supports a connection-oriented transport protocol (Layer 4), a router-core (Layer 3) and several network-interfaces (Layer 1–2).

Key features include:

- Simple API similar to Berkeley sockets;
- Router core with static routes. Supports transparent forwarding of packets, e.g. over space link;
- Support for both connectionless operation (similar to UDP) and connection-oriented operation (based on RUDP);
- Service handler that implements ICMP-like requests, such as ping and buffer status;
- Support for loopback traffic. This can e.g. be used for inter-process communication between subsystem tasks;
- Optional support for broadcast traffic if supported by the physical interface;
- Optional support for promiscuous mode if supported by the physical interface;
- Optional support for encrypted packets with XTEA in CTR mode;
- Optional support for HMAC-authenticated packets with truncated SHA-1 HMAC.

The M6P Platform uses CSP version 1.4 and source code is available at *https://github.com/libcsp/libcsp*. CSP protocol configuration used in the M6P Platform:

CSP feature	Is supported
CRC32	✓
HMAC	-
RDP	-
QOS	-
UDP	✓
XTEA	-

1.7 M6P Platform Network

The M6P Platform, from a software point of view, is like a network, where all network nodes are connected and can communicate with each other freely. The network itself is split into ground and space segments. The ground segment contains ground-station equipment and mission control software. The space segment has nodes of:

- Flight computer (FC);
- Payload controller (PC);
- UHF radio (COMM);
- Electrical power system (EPS);
- Propulsion (engine) control unit (ECU).

In order to achieve network-centric communication between Platform subsystems, CSP protocol is used. Each node has a unique CSP identification code, and packets with specified destination CSP IDs are automatically routed through the network.

The same approach is applied to packets sent from or to the ground station. COMM acts as a bridge or link from the ground segment to the space segment network (satellite internal network). It accepts CSP packets where CSP ID is specified to any CSP ID from the ground segment network. Those packets are packed into radio protocol, sent via UHF to the ground segment, unpacked and sent into the ground segment network. Therefore, from the network point of view, the radio link is not visible. CAN bus is used as a physical layer for the space segment network.

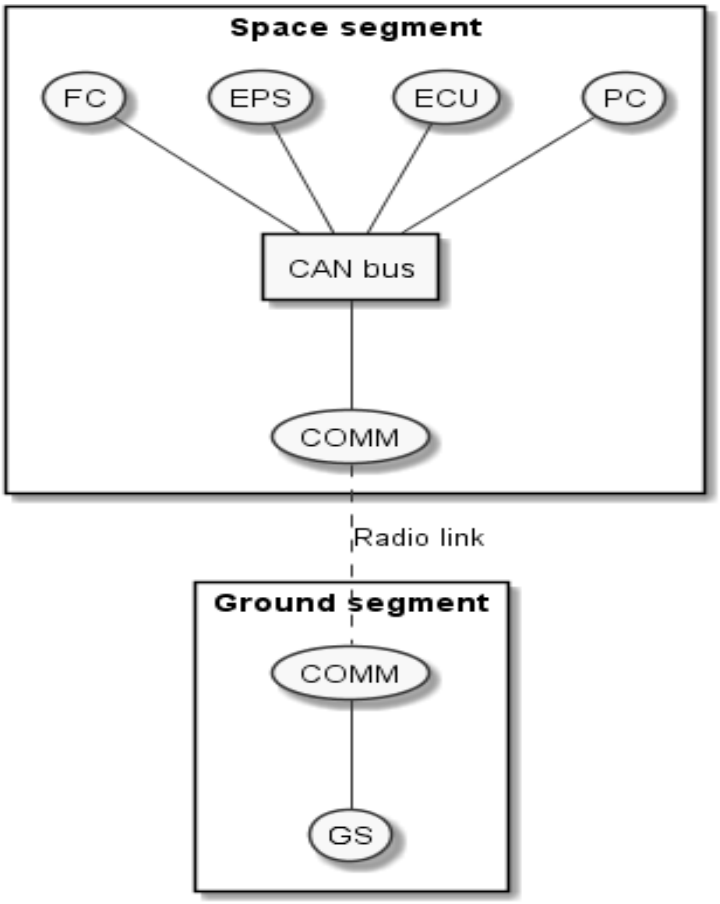


Figure 1. M6P Platform Network

Space segment:

Node	CSP ID
Primary COMM	1
Secondary COM	2
Flight computer (FC)	3
Electric power system (EPS)	4
Engine control unit (ECU)	5
Payload controller (PC)	6
Reserved 1	7
Reserved 2	8
Reserved 3	9
Reserved 4	10
Reserved 5	11
Payload*	12-15

* If payload requires integration into the main M6P network. All CSP packets to payload will be routed through the PC.

Ground segment:

Node	CSP ID
Main ground station	16
Ground station for payload serving	17–20

The CSP network configuration **cannot be changed** during software runtime.

1.8 Request-reply Communication Pattern

Request-reply is one of the basic communication methods used to communicate among subsystems. Request-reply has two participants:

- Requestor sends a request message and waits for a reply message;
- Replier receives the request message and responds with a reply message.

This is a simple, but effective messaging pattern that allows two applications to have two-way communication over a channel. This pattern is used on top of other protocols, like CSP, and allows for Client-server architecture. All requests and replies (having a byte order of **little endian**) are packed into CSP packets.

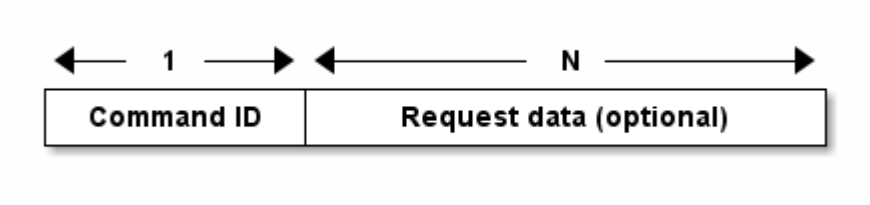


Figure 2. Request packet structure and each field size in bytes

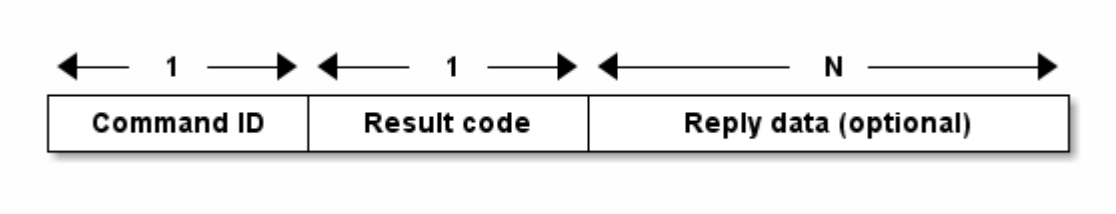


Figure 3. Reply packet structure and each field size in bytes

Request and reply data fields are optional and their size N can be equal to zero.

1.8.1 Command ID

Command ID takes 1 byte. All available commands are listed in the following chapters.

1.8.2 Result Codes

Result code takes 1 byte and in most cases is equal to 0 (false), with the condition of failed execution command, or equal to 1 (true) if execution was successful. Other possible values depend on specific commands and are described in the following chapters.

2 Payload

2.1 Integration into the M6P Network

Due to the use of the CSP protocol, it is required that the CSP be supported by the Payload. Payload CSP ID values range from 12 to 15. A full list of all M6P network node CSP IDs can be found in the *M6P Platform Network* section.

The Payload is integrated into the main M6P network through the Payload controller (PC) and has the ability to communicate with all other subsystems of the Platform. The PC acts as a bridge between two CAN buses and passes CSP packets between them. The request-reply pattern is used for all communications.

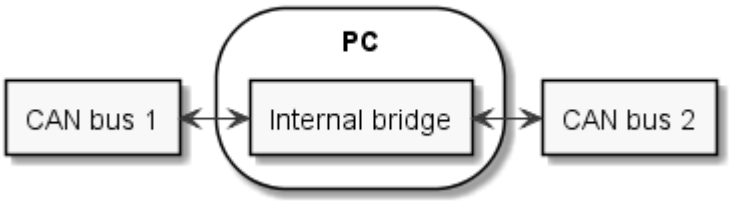


Figure 4. The PC acting as a bridge between two CAN buses

This approach also allows direct communication with the Payload from the ground station using UHF radio. The PC routes CSP packets to the Payload CAN bus and back to the main network CAN bus on which COMM with UHF radio is connected.

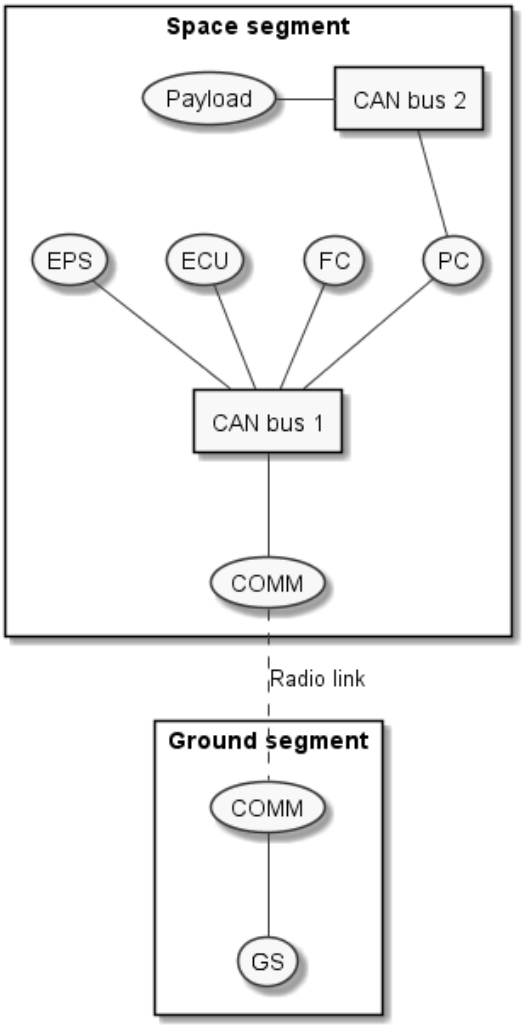


Figure 5. M6P CSP network with integrated Payload

2.2 Payload Data Transfer Using S-band Transmitter

The Payload does not have direct access to the S-band transmitter. The PC is responsible for data collection from the integrated Payload and transfer to the S-band transmitter. More detailed information about session configuration and the data transfer process can be found in the *Payload Controller* chapter.

2.3 Payload Development Continuous Integration for Testing

The Customer can fully integrate the Payload development process into the main M6P platform development cycle. The aim is to provide a TCP/IP connection that can be used as a link for CSP or other protocol packets, giving the ability to access the main M6P CSP network or other M6P software remotely.

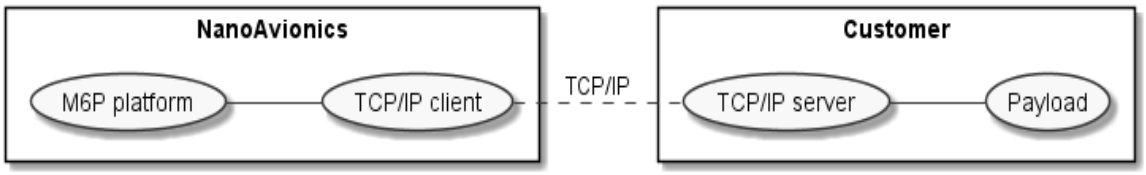


Figure 6. Payload integration into M6P platform during software development and testing phase

The Customer must provide a TCP/IP server IP address and port. NanoAvionics has a TCP/IP client that connects to the Customers server, and the established connection is used for command-and-control CSP packets.

CSP packet format and packet framing, sent over TCP/IP connection, has the same format as CSP packets sent over UART interface using KISS framing. Examples of using CSP and KISS framing can be found in standard CSP library.

3 Payload Controller

3.1 Data Collection from Payload

3.1.1 Polling and Buffering

As mentioned above, the Payload controller (PC) is responsible for data collection from the integrated Payload and transfer to the S-band transmitter. Sessions describe time slots when Payload data is transferred to the S-band transmitter. Two approaches to data collection are available:

- Data polling. The PC acts as master and is responsible for the initiation of session with the Payload. During that session, the PC polls the Payload with specific commands for data and transfers it to the S-band transmitter.
- Data buffering. If the Payload does not have the ability to buffer its own data, then the PC internal file system can be used to store data over time. During the session, data is transferred from the internal PC memory to the transmitter, and no additional Payload polling is performed.

The amount of data transferred can be limited by session duration or by setting a maximum number of packets with known size. The data collection type (polling or buffering) is preconfigured for each session and cannot be changed during runtime. For rideshare missions, Payload does not have direct access to session configuration, but additional options can be discussed under agreement with the Customer.

3.1.2 Session State

The PC has one particular session configuration for each Payload, and each session has three possible states. The states are shown in the table below:

Table 1. Payload controller session state

Session State	Value	Description
NOT ACTIVE	0	Not configured or finished
PENDING	1	Waiting for configured time point to start session
ACTIVE	2	Session is active

Only one session can have an ACTIVE session state at any particular time.

3.1.3 Setup of the Session

The PC session, for Payload serving, must be initialized by the SETUP command using CSP protocol. The SETUP command must contain the following:

1. Session ID that is used for setup. Each Payload has its own dedicated session ID.
2. Session start UTC timestamp. Describes when session should be started. The timestamp value should be synchronized with internal M6P time as a reference point. If the time has already passed or is equal to zero, the session will be started immediately if there are no sessions in ACTIVE state at that time.
3. Session duration, which is measured in seconds. During that period the PC will collect data from the identified Payload and will push data to the S-band transmitter. Minimum session duration time is 5 seconds and maximum time is 900 seconds (or 15 minutes).
4. Maximum number of data packets to be transferred during the session. If the value is equal to zero, then the property is ignored.

5. Size of data frame for data reading from the Payload. Minimum value = 1. Maximum value = 254.
6. Payload synchronization message. This message is sent to the Payload **only once** per session to mark down that the data collection is going to start. For example, it can be used to inform the Payload what data and information needs to be transferred to the ground station during the session, or to pass certain types of information to the Payload itself. The customer is responsible for providing the content of the synchronization message, which can also be empty if required. The maximum length of the synchronization message is 128 bytes.

Table 2. SETUP command example

SETUP Request Data	Description
uint8_t sessionId = 1	Dedicated Payload session ID equal to 1
uint32_t timestamp = 1577836800	Session starts at 2020-01-01 00:00:00
uint16_t sessionDuration = 60	Session duration is 60 seconds
uint16_t maxNumOfPackets = 30	30 packets to be sent during 60-second session
uint8_t pollFrameSize = 64	Each packet will have a size of 64 bytes
uint8_t syncMessage[N] = 'Send telemetry'	The message that is sent to the payload to mark the start of data collection

Only one Payload can be served at a time. If a new SETUP command time configuration overlaps with any other Payload session time configuration, it is discarded.

When a command is received and passes through PC validation, the session enters PENDING state, or ACTIVE state if configured to start immediately. Any session in PENDING state can be reconfigured only if an ABORT command is executed for that session.

3.1.4 Polling

The Payload must support request-and-reply packet format for synchronization and polling messages. A reply result code from the Payload is ignored but must be sent. CSP IDs and destination ports for each Payload sync and polling must be known in advance and cannot be changed during runtime. Synchronization and polling must be performed on the same CSP port.

Before data collection, a SYNC command containing a configured synchronization message for the Payload is sent. The Payload must send a reply with the synchronization message as reply data in order to acknowledge and start data transfer. If acknowledgement is not received, the session is then aborted.

The PC periodically polls the Payload for data packets using the POLL command. Each poll command contains the requested data packet size, which is equal to the size value received with the SETUP command. Reply data size cannot exceed that value but can be less. If the reply data size is equal to zero, no data is transferred to the S-band transmitter to be downloaded to the ground station, but the packets counter number is increased. The PC does not analyze data and serves as a bridge to the S-band transmitter. It is required for the Payload to have data framing implemented inside the packets if needed.

If the Payload does not send a reply to the poll message, then the request is repeated two more times. If all three attempts fail, the transferred data packets counter will still be incremented in order to prevent infinite polling.

3.1.5 Buffering

If the Payload does not have the ability to buffer its own data, then the PC's internal file system can be used to collect data over time. The total available memory size is 256 MB with system information overhead included, with the possibility to extend the available storage size to 32 GB, using low-grade memory. Accumulated data is transferred to the

S-band transmitter during the configured communication session. No additional Payload data polling is performed in that case.

The Payload has to send CSP packets with data to the dedicated CSP port of the PC. The maximum data size in one CSP packet is 256 bytes and new data is always appended at the end of the Payload's dedicated file. After the end of the transfer session, the file is flushed, even if not all data was transferred to the ground station. If the file is filled, further data is discarded.

Files can also be accessed through a regular file system interface. Access to the file system interface for the Payload is denied. For the Payload, it is recommended to have an option to change buffer filling rate.

If the payload does not support CSP protocol and only emits data packets that need to be transferred via S-band, then the payload can be connected directly to the PC via UART interface. This and other similar additional features can be implemented under separate agreement with the Customer.

3.1.6 End of Session

A session can be aborted in the following cases:

- ABORT command is executed;
- Acknowledgement for synchronization message is not received;
- S-band failure.

Normally, a session stops when the configured time duration passes or the maximum number of data packets is reached. If the maximum number value is equal to zero, the corresponding condition is ignored. At the end of the session, the final result is recorded.

Last session-result values can be seen in Table 3.

Table 3. Last session-result values

Session result	Value	Description
DONE	0	Finished successfully or never executed
ABORTED	1	Aborted using ABORT command
NO ACK	2	Payload does not acknowledge synchronization message
S-BAND FAILURE	3	Aborted due to S-band transmitter failure

3.1.7 Current Session Status

The STATUS command is used to get the current state and configuration of the PC session and last session result.

3.1.8 Commands (Data Transfer)

3.1.8.1 For PC

Table 4. Commands for the Payload controller

Command	ID	Description	Request data	Size	Reply data	Size
SETUP	1	Setup new session	uint8_t sessionId uint32_t timestamp uint16_t sessionDuration uint16_t maxNumOfPackets uint8_t pollFrameSize uint8_t syncMessage[N]	10 + N	-	0

STATUS	2		uint8_t sessionId	1	uint8_t lastSessionResult uint8_t currState uint32_t timestamp uint16_t sessionDuration uint16_t maxNumOfPackets uint8_t pollFrameSize uint8_t syncMessage[N]	11 + N
ABORT	3	Abort ACTIVE or PENDING session	uint8_t sessionId	1	-	0

3.1.8.2 For Payload

Table 5. Commands for the Payload

Command	ID	Description	Request data	Size	Reply data	Size
SYNC	1	Synchronization message, configured using SETUP command	uint8_t array[N]	0-128	Equal to request data	Equal to request data size
POLL	2	Data frame poll	uint8_t frameSize	1	uint8_t array[frameSize]	1-requested data frame size

4 Direct Communication with Flight Computer (FC)

4.1 CSP Ports and Available Commands

There are various FC ports that can be used for specific types of actions. Here are a list of ports and commands and a data tree:

- Telemetry port *
  - Get all beacon telemetry (TMALL)
  - Get general telemetry (TMGEN): *
    - Date/time;
    - Uptime;
    - Current mode;
    - Current state;
    - MCU temperature;
    - MCU reset cause
  - Get attitude and position data (TMATP): *
    - Date/time;
    - Orientation quaternion;
    - Position in orbit;
  - Get attitude control data (TMACD): *
    - Date/time;
    - State of algorithm;
    - State of RW;

        - State of magnetorquers;
        - Target mode;
        - Target attitude;
        - Target point;
    - Get raw sensor data (TMRAWSENS):
        - Date/time;
        - Raw sun sensors;
        - Raw rotation speed;
        - Raw acceleration;
        - Raw magnetic field strength;
        - Raw sensors temperatures;
    - Get pre-processed sensor data (TMSENS): *
        - Date/time;
        - Sun vector;
        - Rotation speed vector;
        - Acceleration vector;
        - Magnetic field strength vector;
        - Sensors temperatures in Celsius;
- Real-time clock (RTC) port *
    - Get RTC timestamp (RTCSTMGET) *
- Attitude target control port *
    - Get target attitude (ATCATGET);
    - Set target attitude (ATCATSET);
    - Get target point (ATCPTGET);
    - Set target point (ATCPTSET);
    - Get target mode (ATCMDGET);
    - Set target mode (ATCMDSET);
- CSP service handler port *
    - Send ping (CSPPING) *
    - Get system uptime (CSPUPTIME) *

* Ports and commands available for Payload.

Note: All available ports, commands and names can change depending on mission and satellite configuration. In addition, we present only preliminary data structures available using specified ports and commands.

4.2 Commands (Flight Computer)

4.2.1 FC Telemetry Port Commands

Table 6. FC telemetry port commands

Command	ID	Description	Request data	Size	Reply data	Size
TMGEN	1	Get general telemetry	-	-	uint32_t rtc_timestamp uint32_t uptime uint32_t global_on_time uint8_t current_mode uint8_t current_state int8_t MCU_temp uint8_t reset_cause	16
TMATP	2	Get attitude and position data	-	-	uint32_t rtc_timestamp float64_t quaternion[4] float64_t position[3]	60
TMACD	3	Get attitude control data	-	-	uint32_t rtc_timestamp uint8_t state uint8_t rwstate[4] uint8_t rwrpm[4] uint8_t mtqstate[3] uint8_t trgt_mode uint8_t trgt_attitude uint8_t trgt_point	19
TMSENS	4	Get pre-processed sensor data	-	-	uint32_t rtc_timestamp float64_t sun_vec[3] float32_t rotation[3] float32_t acc_vec[3] float32_t mag_vec[3] int8_t sensor_temps[6]	70

4.2.2 FC Real-time Clock Port Commands

Table 7. FC Real-time clock port commands

Command	ID	Description	Request data	Size	Reply data	Size
RTCSTMGET	2	Get RTC timestamp	-	-	uint32_t rtc_timestamp	4

4.2.3 FC Attitude Control Port Commands

Table 8. FC attitude control port commands

Command	ID	Description	Request data	Size	Reply data	Size
ATCATGET	1	Get target attitude	-	-	float64_t trgt_att_quat[4]	16
ATCATSET	2	Set target attitude	float64_t trgt_att_quat[4]	16	-	-
ATCPTGET	3	Get target point	-	-	float64_t trgt_pt_quat[3]	16
ATCPTSET	4	Set target point	float64_t trgt_pt_quat[3]	16	-	-
ATCMDGET	5	Get target mode	-	-	uint8_t trgt_mode	1
ATCMDSET	6	Set target mode	uint8_t trgt_mode	1	-	-

4.2.4 FC CSP Service Handler Commands

Table 9. FC CSP service handler commands

Command	ID	Description	Request data	Size	Reply data	Size
CSPPING	1	Send ping	(optional)	N	(same as req.)	N
CSPUPTIME	6	Get system uptime in ms	-	-	uint32_t uptime	4