

**astrOcast**



## ASTRONODE PILOT ASSET API USER GUIDE

Astrocast SA  
This is a confidential document  
Last updated: 8 December 2020

## TABLE OF CONTENT

Table of Content	1
<b>1 Definitions</b>	<b>2</b>
1.1 Applicable Documents	2
1.2 Acronyms and Abbreviations	2
<b>2 Icons</b>	<b>3</b>
<b>3 Introduction</b>	<b>4</b>
3.1 Satellite vs. Wi-Fi Development Kits	4
<b>4 Conventions, Data Format</b>	<b>5</b>
4.1 Unsigned integer values format convention	5
4.2 Bit/octet numbering convention	5
4.3 Terms and descriptions	6
<b>5 General protocol description</b>	<b>7</b>
5.1 UART configuration	7
5.2 Timing and Timeouts	7
5.3 Message structure	7
5.4 ID list	8
5.5 Configuration	9
5.6 Telemetry	10
5.7 Acknowledgment events	10
5.8 Error management	12
<b>6 Detailed protocol description</b>	<b>14</b>
6.1 Astronode Satellite Dev-Kit registers	14
6.1.1 Read registers	14
6.1.2 Write configuration	16
6.2 Wi-Fi Dev Kit registers (Wi-Fi Dev Kit only)	18
6.3 Enqueue telemetry payload	20
6.4 Dequeue telemetry payload	22
6.5 Geolocation write command	23
6.6 Downlink event management	25
6.6.1 Receiving notification	25
6.7 Reading satellite acknowledgement	27
<b>7 Annex A: CRC</b>	<b>29</b>
7.1 Software implementation	29
7.2 Verification of compliance	31
<b>8 Disclaimer</b>	<b>32</b>

## 1 DEFINITIONS

### 1.1 APPLICABLE DOCUMENTS

- [AD1] Pilot Wi-Fi Development Kit User Guide
- [AD2] Astronode Pilot Satellite Development Kit User Guide
- [AD3] Data Management Platform Introduction
- [AD4] Pilot Program Welcome

### 1.2 ACRONYMS AND ABBREVIATIONS

**AES.** Advanced Encryption Standard

**API.** Application Programming Interface

**DL.** Down Link: Satellite to the Development Kit, RF link

**DM.** Data Management

**HW.** Hardware

**IoT.** Internet of Things

**M2M.** Machine to Machine

**RF.** Radio Frequency

**SW.** Software

**MCU.** Microcontroller Unit

**TC.** Telecommand: incoming command from satellite Down Link to Development Kit

**TM.** Telemetry: outgoing payload from the asset, Dev. Kit to the satellite via Up Link

**UL.** Up Link: Development Kit to Satellite. RF link

## 2 ICONS

Within this document, the following icons may help the reader on some aspects:



Important to read and remember



Insight into the NUCLEO64 Example\_Asset driver



Additional details for a given function



A system performance constraint

### 3 INTRODUCTION

This document describes how to use the Astronode Pilot Asset API on both Precursor Wi-Fi and Satellite Development Kits.

#### 3.1 SATELLITE VS. WI-FI DEVELOPMENT KITS

As described in each of the Satellite and Wi-Fi Development Kit User Guides ([AD1], [AD2]), the kits are designed to be interchangeable to ease development. The protocol described in this document is almost identical for each kit. There are two differences to highlight:

1. The read configuration message will return a different product ID for the two kits, allowing them to be distinguished.
2. The Satellite Development Kit will answer with an error if the Wi-Fi configuration command is sent to it.

When integrating the protocol into a system, be aware of the significant performance differences in the latency of the Wi-Fi and Satellite development kits. On Wi-Fi, expect to receive uplink acknowledgments in seconds. With the satellites, expect to wait much longer. Refer to [AD4] for more information on the service levels.

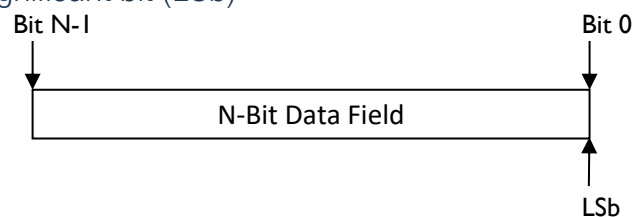
## 4 CONVENTIONS, DATA FORMAT

### 4.1 UNSIGNED INTEGER VALUES FORMAT CONVENTION

Hexadecimal values are always prefixed by the two characters "0x". Example: 0x8000 is equal to the decimal value 32768.

### 4.2 BIT/OCTET NUMBERING CONVENTION

The first bit in the field to be transmitted (i.e. the most right-justified bit when drawing a figure) is defined to be "Bit 0"; the following bit is called "Bit 1" and so on up to "Bit N-1". Bit 0 is the Least Significant bit (LSb)



When the field is used to express a binary value (such as an integer of more than one byte), the Least Significant Byte (LSB) shall be the first transmitted byte, i.e. in **little-endian** format.

Example for 0x12345678:

Byte 0 (LSB)	Byte1	Byte 2	Byte 3 (MSB)
0x78	0x56	0x34	0x12

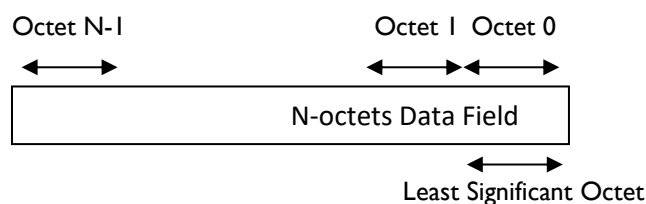
An octet (i.e. a byte) is 8-bits length.

A short word is 16-bits length (i.e. 2 octets).

A word is 32-bits length (i.e. 4 octets).

A long word is 64-bits length (i.e. 8 octets).

Numbering of octets in a field is done from LSB as Octet 0 to MSB as Octet in an N-octet field.



### 4.3 TERMS AND DESCRIPTIONS

Term	Description
Request	The term used in the Asset - Development Kit protocol for write or read operation from the Asset to the Development Kit.
Answer	When the Asset sends a Request (above), the Development Kit will reply with an Answer, providing the status of the request operation. The direction is from the Development Kit to the Asset.
Notification	The term used in the Asset - Development Kit protocol for the Development Kit spontaneously notifying the Asset via the DLN pin.
Message	Command or response identifier
Uint8	8-bit unsigned integer (1 byte)
Uint16	16-bit unsigned integer (2 bytes)
Byte[N]	An array of N bytes
Timeout	The Asset API has 2 timeouts: inter-byte timeout within a message; and another timeout between commands and responses. See section 5.2.

## 5 GENERAL PROTOCOL DESCRIPTION

### 5.1 UART CONFIGURATION

The following settings shall be used for the UART:

- 8 data bits
- 1 start bit
- 1 stop bit
- No parity
- No hardware flow control
- 9600 Baud rate


### 5.2 TIMING AND TIMEOUTS

All answers to requests can be expected to be received within 100ms after reception of the last byte of the request. Past that delay, the asset can consider that a transmission error has occurred, and the asset should retry.

There should be minimum of 10ms delay between an answer (development kit to asset) and the next request (asset to development kit). i.e. After receiving an answer, the asset should delay 10ms before sending the next request.

A 100ms inter-byte timeout is implemented to recover from incomplete requests. Within a single command, bytes should not be spaced by more than 100ms. With more than 100ms between bytes, the development kit will discard the in-progress receive operation and wait for the next start byte.

### 5.3 MESSAGE STRUCTURE

 All messages between the Satellite Development Kit and Asset will have the following fields:

Start Byte	Message ID	Length	Message Parameters	CRC-16-CCITT*
0x7F	1 byte	2 bytes	variable	2 bytes

Data used for CRC computation

Variable size determined by Length

Each command is acknowledged by an answer message.

Requests and answers shall comply with the above message format.

See the following table for details of the message IDs.

The CRC is explained in Annex A.



## 5.4 ID LIST

 List of Request IDs, corresponding Answer IDs, and their descriptions:

Asset à Terminal			Terminal à Asset		
Request			Answer		
ID	Name	Description	ID	Name	Description
0x05	CFG_WR	Writes configuration to Development Kit volatile memory	0x85	CFG_WA	Answers last configuration write operation with status
0x06	WIF_WR	Writes Wi-Fi settings in Development Kit volatile memory (Wi-Fi only)	0x86	WIF_WA	Answers last Wi-Fi settings write operation with status (Wi-Fi only)
0x15	CFG_RR	Reads configuration from Development Kit volatile memory	0x95	CFG_RA	Answers last configuration read operation with value
0x25	PLD_ER	Enqueues asset payload in Development Kit volatile memory	0xA5	PLD_EA	Answers last payload enqueue operation with status
0x26	PLD_DR	Dequeues asset payload from Development Kit volatile memory	0xA6	PLD_DA	Answers last payload dequeue operation with status
0x35	GEO_WR	Writes geolocation longitude and latitude in Development Kit volatile memory	0xB5	GEO_WA	Answers last geolocation write operation with status
0x45	SAK_RR	Reads Satellite Acknowledgment	0xC5	SAK_RA	Answers with Satellite Acknowledgment information
0x46	SAK_CR	Confirms to the Development Kit that Satellite Acknowledgment was properly decoded and can be deleted.	0xC6	SAK_CA	Answers last SAK_CR confirmation
0x65	DLN_RR	Reads DL event register	0xE5	DLN_RA	Answer indicates which Downlink Events are currently pending
			0xFF	ERROR	Answers a request reporting an error.

Naming conventions for the table above:

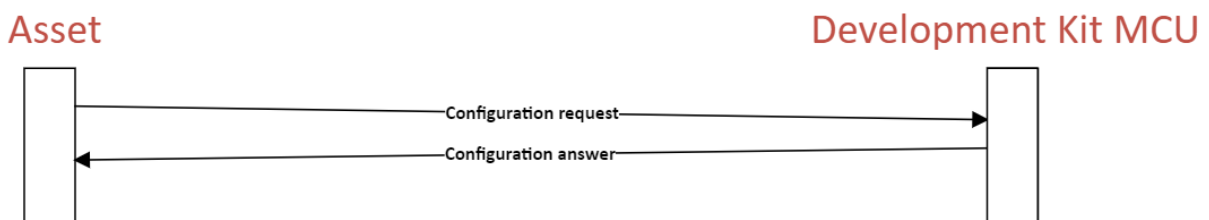
- R suffix (\*\*\*\_R) refers to **Request**.
- A suffix (\*\*\*\_A) refers to **Answer**.
- The letter R/A is usually the action related to the command:
  - \*\*\*\_W\* - Write,
  - \*\*\*\_R\* - Read,
  - \*\*\*\_E\* - Enqueue,
  - \*\*\*\_D\* - Dequeue,
  - \*\*\*\_C\* - Confirm.
- The 3-letter prefix describes the operation further:
  - CFG\_\*\* - Configuration,
  - WIF\_\*\* - Wi-Fi,
  - PLD\_\*\* - Payload,
  - GEO\_\*\* - Geolocation,
  - SAK\_\*\* - Satellite Acknowledgement,
  - DLN\_\*\* - Downlink Notification.

Put this all together for something like DLN\_RR as Downlink Notification Read Request, or DLN\_RA as Downlink Notification Read Answer.

The following sections discuss the usage of these commands. Section 6 provides a detailed description of the protocol commands and responses.

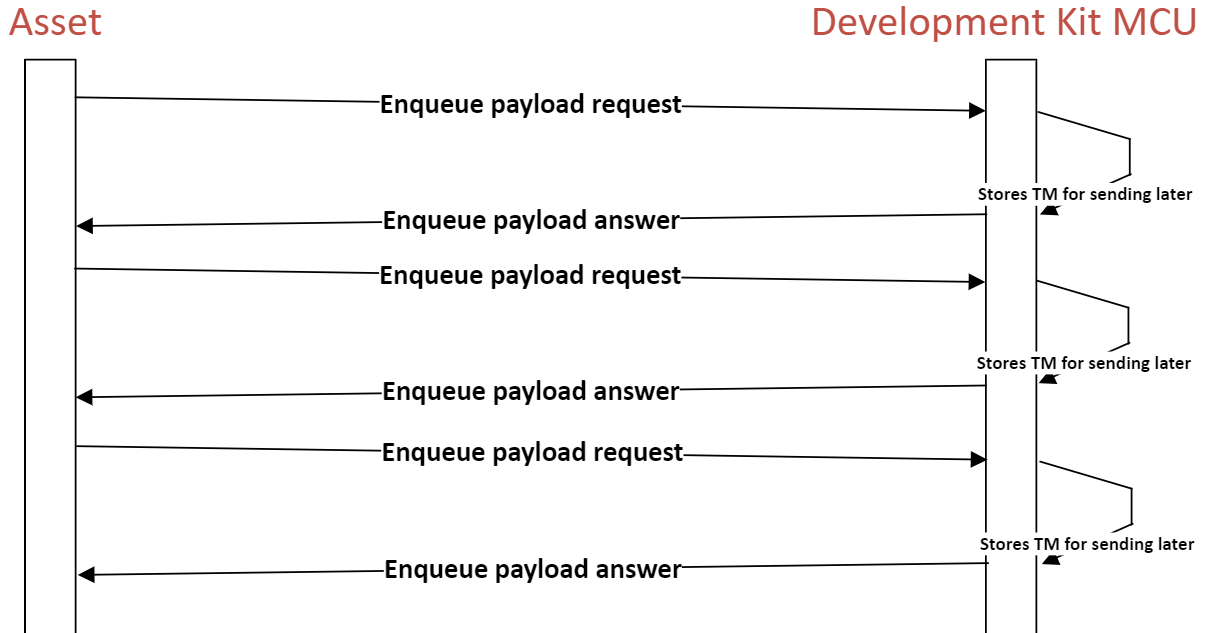
## 5.5 CONFIGURATION


Sending TM data and receiving acknowledgments can be done directly using default factory-settings transmission parameters. To use non-standard parameters, custom configuring is done the following way, using CFG\_WR / CFG\_WA, CFG\_RR / CFG\_RA, GEO\_WR / GEO\_WA:




## 5.6 TELEMETRY

 Sending TM payload is done the following way, using PLD\_EQ and PLD\_EA IDs:



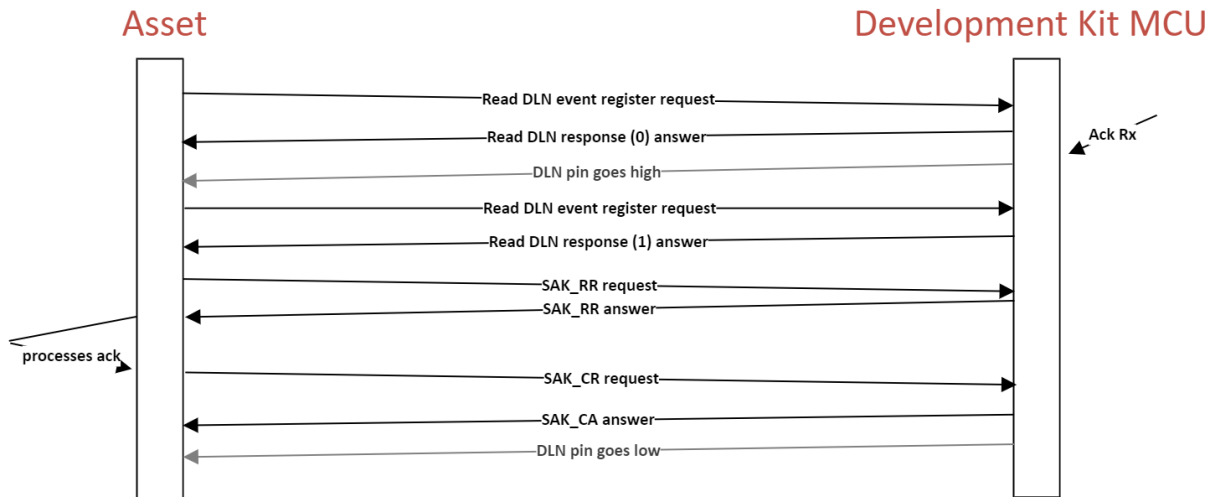
 Up to 44B per message can be queued, a maximum 8 messages at a given time. Each message can be tagged with a TM counter, to recognize the corresponding acknowledgment later. It is currently not possible to wipe the buffer in a single command. However, this will be a feature that will be available in our future terminal. The current workaround is to use the payload dequeue multiple times in a row.

## 5.7 ACKNOWLEDGMENT EVENTS


 The terminal will notify the asset of events setting the DLN pin high. The nature of the event can then be checked using the DLN\_RR command. Currently only TM payload successfully sent to the satellite are notified. Notifications can be enabled or disabled using the configuration register (see section 6.1).

Alternatively of monitoring the status of the DLN pin, the events can be checked by polling the terminal with the DLN\_RR command.

In case of TM payload acknowledgement, the asset can read which TM payload was acknowledge using the SAK\_RR command, and then clear the acknowledgement using a SAK\_CR. As shown in the diagram below. Note that this is not an end-to-end acknowledgment but only satellite acknowledgment for Satellite Dev Kit and server acknowledgment for Wi-Fi Dev Kit.



Unless the end-user does not care about getting acknowledgments of *messages being received by the satellite*, the *TM payload acknowledged by satellite* bit should remain set to true (default value).

 Occasionally, this acknowledgement may only be reported during the second next satellite passage; this time delay will be up to 24 hours for the demo and 15 minutes with the full constellation. for more information about performance. Refer to downlink event management and read the DL information paragraphs for details.

## 5.8 ERROR MANAGEMENT

In case of error, Response can also be (0xFF) and will comply with the following description:

<i>Message structure</i>	<i>Start</i> 0x7F	<i>ID</i> 0xFF	<i>Length (Bytes)</i> 0x0002	<i>Parameters</i> See below	<i>Checksum</i> LSB MSB
<i>Parameters:</i>					
<i>Byte offset</i>	<i>Format</i>	<i>Name</i>	<i>Description</i>		
0					

Table 1 - Error code list

<i>Error code</i>	<i>Error type</i>	<i>Name</i>	<i>Description</i>
0x0001	COMM	CRC_ERROR	Discrepancy between provided CRC and expected CRC.
0x0011	COMM	WRONG_LENGTH	Field length of the message is invalid for the given command ID. Notes: <ul style="list-style-type: none"> <li>In case of frame too long, the terminal can send this error code before receiving the entire frame.</li> <li>This error code is also used in response of a PLD_ER if the payload sent is too long (44 bytes without geolocation, 36 with)</li> </ul>
0x0021	COMM	INVALID_ID	Invalid command ID used.
0x0601	WIF_ER	FORMAT_NOT_VALID	At least one of the fields (SSID, password, access token) is not composed of exclusively printable standard ASCII characters (0x20 to 0x7E) or is not null terminated.
0x2501	PLD_ER	BUFFER_FULL	Failed to queue the payload because the sending queue is already full.
0x2511	PLD_ER	DUPLICATE_COUNTER	Failed to queue the payload because a message with the same TM counter is already present in the buffer
0x2601	PLD_DR	BUFFER_EMPTY	Failed to dequeue a message from the buffer because the buffer is empty
0x3501	GEO_WR	INVALID_POS	Failed to update the geolocation information. Latitude and longitude

			fields must in the range [-90,90] degrees and [-180,180] degrees, respectively.
0x4501	SAK_RR	NO_ACK	No satellite acknowledgement available for any message.
0x4601	SAK_CR	NO_CLEAR	No message to clear or it was already cleared.

## 6 DETAILED PROTOCOL DESCRIPTION

### 6.1 ASTRONODE SATELLITE DEV-KIT REGISTERS

 The Astronode Pilot Satellite Development Kit comprises 6x 8bit registers:

- A Product ID
- A HW revision byte (read-only)
- 3 FW version bytes (read-only)
- A **Configuration Register** (R/W)

There is no memory context saving, meaning in case of reboot, the Development Kit will reset to its default values (see next).

 This configuration function is ensured by `astronode_devkit_initialize ()` in the driver.

 Development Kit **Configuration Register** allows you to specify the following:

- The Development Kit provides the possibility to tag all your messages with the GPS coordinates of the asset, simply by specify latitude and longitude once per test session with an appropriate command. Adding this geolocation data is only possible when *Add geolocation* corresponding bit is set to true. It is false i.e. disabled by default at each start up. If used the maximum payload size is reduced by eight bytes.

Refer to geolocation write command paragraphs for details.

- If the message acknowledgment from satellite is going to be reported to the asset or not.

#### 6.1.1 READ REGISTERS

HW revision, FW version bytes and the *Configuration Register* can be read for status using the CFG\_RR command below:

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	CFG_RR=0x15	0x0000	none	LSB MSB

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	CFG_RA=0x95	0x0006	See below	LSB MSB
<i>Parameters:</i>					
Byte offset	Format	Name	Description		
0	UInt8	Product ID (read only)	Wi-Fi (0) or satellite (1)		
1	UInt8	Hardware revision (read only)	Terminal electronics version		
2	UInt8	Firmware major version (read only)	Digit used when new features added or loss of compatibility		
3	UInt8	Firmware minor version (read only)	Digit used only for fixing bugs or minor feature improvement		
4	UInt8	Firmware revision version (read only)	Digit used only for fixing bugs or minor feature improvement		
5	UInt8	Configuration Register (R/W, write described in 6.1.2)	Read protocol settings (not persisted in flash memory once received in the terminal SRAM)		

CFG\_RA response is defined as:

*Configuration register value Bitfield:*

\*since messages are encrypted just after queuing: if geolocation set before queuing only

7	6	5	4	3	2	1	0
0	0	0	0	0	0	Add Geolocation (next payloads only*)	TM Payload Acknowledged by Satellite (immediate effect*)
						Default=0	Default=1

will be encapsulated. On the contrary acknowledgment bit applies immediately to all queued message(s).

**Example:**

Asset sends a CFG\_RR command.

Start	Message ID	Length	CRC
0x7F	0x15	0x0000	0xBAC8



The Development Kit will use CRC incoming bytes to verify data integrity.

Considering the example above, the Development Kit should reply with a CFG\_RA:

Start	Message ID	Length	Parameters	CRC
0x7F	0x95	0x0006	0x01 0x01 0x00 0x01 0x00 0x01	0xBBA2**

The asset may use the information to check that the precedent *configuration register* write command was done properly. It may also check CRC incoming bytes from the response to verify data integrity.

\*\* As defined earlier in 4.2 conventions and 5.2 message structure, CRC computation shall be applied to 0x7F 0x95 0x06 0x00 0x01 0x01 0x00 0x01 0x00 0x01 in that exact strict order.

### 6.1.2 WRITE CONFIGURATION

In case the default parameters are not suitable for the end-user, it is possible to change them by using the *Configuration Register* **CFG\_WR** command, using the following structure below. Note again that configuration is lost in case of power off/on cycle.

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	CFG_WR=0x05	0x0001	see below	LSB MSB
<i>Parameters:</i>					
Byte offset	Format	Name	Description		
0	UInt8	Development Kit Configuration Register	Configurable protocol parameters		

*Configuration register* value Bitfield (reminder):

7	6	5	4	3	2	1	0
0	0	0	0	0	0	Add geolocation	TM Payload Acknowledged by Satellite

The CFG\_WA response is defined as follows:

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	CFG_WA=0x85	0x0000	none	LSB MSB

#### Example:

The asset sends a CFG\_WR command.

Start	Message ID	Length	Parameters	CRC
0x7F	0x05	0x0001	0x03	0xD265

Development Kit will use CRC incoming bytes to verify data integrity. Considering the example above, the Development Kit should reply with a CFG\_WA:

Start	Message ID	Length	CRC
0x7F	0x85	0x0000	0xC2F1

The Development Kit will then use new custom transmission parameters.

The asset may use check CRC incoming bytes from the response to verify data integrity.

## 6.2 WI-FI DEV KIT REGISTERS (WI-FI DEV KIT ONLY)

A wireless connection of the Wi-Fi Development Kit with the Astrocast DM can be achieved by configuring the following parameters:

- WLAN\_SSID = your company network SSID name. See the note below on acceptable characters.
- WLAN\_KEY = corresponding key (password) of your SSID. See the note below on acceptable characters.
- AUTH\_TOKEN = 96 Byte access token generated on the Astrocast Portal.

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	WIF_WR=0x06	194	see below	LSB MSB
<i>Parameters:</i>					
<i>Byte offset</i>	<i>Format</i>	<i>Name</i>	<i>Description</i>		
0	Byte[33]	WLAN_SSID	LAN SSID (maximum 32 characters plus null termination; not persisted in flash memory)		
33	Byte[64]	WLAN_KEY	Corresponding key of the SSID (maximum 63 characters plus null termination; not persisted in flash memory)		
97	Byte[97]	AUTH_TOKEN	API access token for Wi-Fi module (fixed size 96 characters plus null termination; not persisted in flash memory)		

The CFG\_WA response is defined as follows:

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	WIF_WA=0x86	0x0000	none	LSB MSB

**Note on SSID and Key characters:**

The IEEE 802.11 Wi-Fi specification allows the SSID and key to include non-printable characters and null characters (0). The Wi-Fi development kit intentionally deviates from this, and only characters in the ASCII range 0x20 to 0x7E should be used, terminated with a null character. This includes 0-9, a-z, A-Z and many symbols. This development kit will not work with SSIDs or keys that use characters outside of this range. If this is a problem, please contact support.

**Example:**

The Asset sends a WIF\_WR command.

Start	Message ID	Length	Parameters	CRC
0x7F	0x06	194	0x02 0xFF 0xFF 0x02 0xEE 0xEE 0x...	custom

The Dev Kit will use CRC incoming bytes to verify data integrity.

Considering the example above, the Dev Kit should reply with a WIF\_WA:


Start	Message ID	Length	CRC
0x7F	0x86	0x0000	0x9BA1

The Dev Kit will then use new custom transmission parameters.

The Asset may use check CRC incoming bytes from the response to verify data integrity.

### 6.3 ENQUEUE TELEMETRY PAYLOAD

 This function is ensured by `astronode_send_message()` in the driver.

 Directly after power on (default configuration), or once the Development Kit has been configured by the asset, a **PLD\_ER command** can be used to transfer one or multiple set(s) of TM payload and as many times as necessary.

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	PLD_ER=0x25	2+N (see below)	See below	LSB MSB
<i>Parameters:</i>					
Byte offset	Format	Name	Description		
0	Uint16	TM Counter	TM payload identifier chosen by the asset. This counter will be reused to tag the corresponding acknowledgments from the satellites. The TM counter needs to be unique and nonzero. See description below for details		
2	Byte []	TM Payload bytes	The N bytes of the payload to send. N must be <= 44 bytes if geolocation data is not activated, <= 36 bytes if geolocation is activated.		

#### Notes:

If the payload length is above the specified value (N>44 bytes if geolocation data is not activated and N>36 bytes if geolocation is activated), the terminal will return a `COMM_WRONG_LENGTH` error and the message will not be queued.

For the Beta demonstration, a maximum of 8 messages can be queued in the terminal. The messages queued are lost upon terminal power-cycle or reset. If 8 messages are already queued in the terminal, a `PLD_ER` will return a `PLD_ER_BUFFER_FULL` error and the message will not be queued. There exist two options to free space in the queue:

- If a message was successfully sent to the satellite, the asset can clear the message acknowledgment using a `SAK_CR` (see section 6.6 and 6.7)
- The asset can decide to dequeue the oldest message in the queue, even if it was not sent to the satellite, using the `PLD_DR` command (see section 6.4)

The TM counter is used to unambiguously refer to a specific TM payload in the `PLD_EA` response, `SAK_RR` and `PLD_DR`. To avoid any ambiguity, if the TM counter in the `PLD_ER` is identical to the one of a message already present in the buffer, the terminal will respond with a `PLD_ER_DUPLICATE_COUNTER` error and the new message will not be queued.

If PLD\_DR is successful, the PLD\_EA response is defined as:

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	PLD_EA=0xA5	0x0002	See below	LSB MSB
<i>Parameters:</i>					
<i>Byte offset</i>	<i>Format</i>	<i>Name</i>	<i>Description</i>		
0	Uint16	TM Counter	Refers to value provided in PLD WR		

**Example:**

The asset TM payload can be sent using the PLD\_EQ command:

Start	Message ID	Length	Parameters	CRC
0x7F	0x25	0x0004	0x01 0x00 0xBA 0xDC	0xC483

The Development Kit will answer with a PLD\_EA response:

Start	Message ID	Length	Parameters	CRC
0x7F	0xA5	0x0002	0x01 0x00	0x59E5

## 6.4 DEQUEUE TELEMETRY PAYLOAD

The PLD\_DR command is used to dequeue the oldest message present in the terminal's message buffer, even if it was not sent to the satellite. If the message buffer is empty, the terminal will respond to PLD\_DR with a PLD\_DR\_BUFFER\_EMPTY error.

Note: To clear the entirety of TM payload, the PLD\_DR request can be sent to the terminal until a PLD\_DR\_BUFFER\_EMPTY error is received.

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	PLD_DR=0x26	0	None	LSB MSB

If PLD\_DR is successful, the PLD\_DA response is defined as:

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	PLD_DA=0xA6	0x0002	See below	LSB MSB
<i>Parameters:</i>					
<i>Byte offset</i>	<i>Format</i>	<i>Name</i>	<i>Description</i>		
0	Uint16	TM Counter	TM counter of the message dequeued. Refers to value provided in PLD_ER when the message was queued		

### Example:

The oldest asset TM payload can be dequeued using the PLD\_EQ command:

Start	Message ID	Length	Parameters	CRC
0x7F	0x26	0x0000	none	0x263D

The Development Kit will answer with a PLD\_EA response:

Start	Message ID	Length	Parameters	CRC
0x7F	0xA6	0x0002	0x01 0x00	0xF9CD

## 6.5 GEOLOCATION WRITE COMMAND

⚠ This geolocation function is ensured by `astronode_set_gnss_coordinates ()` in the driver, using `asset_set_longitude ()` and `asset_set_latitude ()` double to decimal precision conversion functions.

Latitude and longitude values are encapsulated in the M2M message if the *Add Geolocation* bit has been set to 1. Therefore, these coordinates must be specified before enabling that bit, by using the command described here below. If not specified, then default coordinates (0,0) would be used.

*Note: the end user is free to encapsulate his geolocation data within the payload, without having to use this specific command.*

Longitude and latitude are set using the following command:

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	GEO_WR=0x35	0x0008	See below	LSB MSB
<i>Parameters:</i>					
Byte offset	Format	Name	Description		
0	Int32	Longitude	Longitude in 1/10 of $\mu^\circ$ (i.e. $^\circ$ multiplied by 1E7) e.g. 123456780 = 12.345678 $^\circ$ Must be in the range -180 $^\circ$ to 180 $^\circ$ (not persisted in flash memory once received in the terminal SRAM). Initial value at each start-up is 0.		
4	Int32	Latitude	Latitude in 1/10 of $\mu^\circ$ (i.e. $^\circ$ multiplied by 1E7) e.g. 123456780 = 12.345678 $^\circ$ Must be in the range -90 $^\circ$ to 90 $^\circ$ (not persisted in flash memory once received in the terminal SRAM). Initial value at each start-up is 0.		

If an invalid longitude or latitude are sent, both values are discarded and a `INVALID_POS` error is returned.

The `GEO_WA` response is defined as:

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	GEO_WA=0xB5	0x0000	none	LSB MSB



**Example:**

The assets present longitude and latitude values can be sent using GEO\_WR command:


Start	Message ID	Length	Parameters	CRC
0x7F	0x35	0x0008	0x98 0xBA 0xDC 0xFE 0x98 0xBA 0xDC 0xFE*	0x8B2D

\*see driver for float to compressed values examples.

The Development Kit will answer with an GEO\_WA response:


Start	Message ID	Length	CRC
0x7F	0xB5	0x0000	0x0754

## 6.6 DOWNLINK EVENT MANAGEMENT

 This downlink event management function is ensured for the acknowledgment by `astronode_get_acknowledgments ()` in the Example Asset driver. Acknowledgements are coming from the satellite in case of Satellite Dev Kit, whereas it is a server acknowledgement in case of Wi-Fi Dev Kit.

### 6.6.1 RECEIVING NOTIFICATION

#### DLN hardware pin notification

 Satellite TM payload acknowledgements as well as future incoming telecommands (with corresponding service level will be in service in 2021) are notified on the DLN pin of the Development Kit. For this precursor mission, DLN only indicates the presence of readable values in the form of a satellite acknowledgement. The pin will go low only once all information is read and cleared.

When DLN pin high occurs, the asset may use a `DLN_RR` command to read the *Event Register* (see `DLN_RR` and `DLN_RA` descriptions hereafter). A second method is to periodically poll that same register with again a new `DLN_RR` command.

#### DLN\_RR command

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	DLN_RR=0x65	0x0000	none	LSB MSB

#### DLN\_RA notification

Message structure	Start	ID	Length (Bytes)	Parameters	Checksum
	0x7F	DLN_RA=0xE5	0x0001	See below	LSB MSB
<i>Parameters:</i>					
Byte offset	Format	Name	Description		
0	UInt8	Event register	Represents the type of event(s) available for reading		

#### Event register value Bitfield:

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	SAK

- SAK: Satellite acknowledgement
- Other bits are reserved for future use.


#### Example:

An asset can read the type of Downlink notification using `DLN_RR` command:


Start	Message ID	Length	CRC
0x7F	0x65	0x0000	0x62C0

The Development Kit will answer using a DLN\_RA notification, e.g. here showing both SAK type is available:

Start	Message ID	Length	Parameters	CRC
0x7F	0xE5	0x0001	0x01	0x76CD

 Following this DLN\_RA notification, the actual data corresponding to the notification (i.e. TM data acknowledgement *TM Counter*) can be read, see next.

## 6.7 READING SATELLITE ACKNOWLEDGEMENT

 Upon DLN\_RA reception, with Event register bit 0=1, the asset can now read the **TM counter**, which the notification refers to; this is done using the following command:

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	SAK_RR=0x45	0x0000	none	LSB MSB


In response to this command, a SAK\_RA response is immediately transmitted by the Development Kit. The asset can then confirm the proper reception of the TM data Satellite Acknowledgement, using a SAK\_CR confirmation. If no satellite acknowledgement is available, the Development Kit will respond to the SAK\_RR with a NO\_ACK error.

SAK\_RA is defined as follows:

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	SAK_RA=0xC5	0x0002	See below	LSB MSB
<i>Parameters:</i>					
<i>Byte offset</i>	<i>Format</i>	<i>Name</i>	<i>Description</i>		
0	Uint16	<b>TM Counter</b>	Satellite Acknowledgment TM payload identifier which was originally provided by the asset		

The SAK\_CR confirmation as follows:

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	SAK_CR=0x46	0x0000	none	LSB MSB

 Note that if a SAK\_CR is sent multiple times in a row to the terminal or that a SAK\_RR was not first sent, the terminal will respond with a NO\_CLEAR error.

After clearing a satellite acknowledgement, sending the SAK\_RR command will return the next available satellite acknowledgement.

When clearing a Satellite Acknowledgement with this message, the Development Kit will free the corresponding memory and erase the oldest Satellite Acknowledgement information.

The SAK\_CA acknowledgment is as follows:

<i>Message structure</i>	<i>Start</i>	<i>ID</i>	<i>Length (Bytes)</i>	<i>Parameters</i>	<i>Checksum</i>
	0x7F	SAK_CA=0xC6	0x0000	none	LSB MSB

### Example:

Once notified or using polling mode, the asset can read the Satellite Acknowledgment notification using SAK\_RR command:

Start	Message ID	Length	CRC
0x7F	0x45	0x0000	0xE406

The Development Kit will answer using a SAK\_RA response, here showing that TM with identifier 0x0001 has been sent to the spacecraft:

Start	Message ID	Length	Parameters	CRC
0x7F	0xC5	0x0002	0x01 0x00	0x4039

The asset may check data consistency with the CRC information and then confirm with a SAK\_CR notification:

Start	Message ID	Length	CRC
0x7F	0x46	0x0000	0xBD56

To which the Terminal will answer (SAK\_CA):

Start	Message ID	Length	CRC
0x7F	0xC6	0x0000	0x860C

## 7 ANNEX A: CRC

The CRC-16-CCITT is defined by the following polynomial:  $x^{16} + x^{12} + x^5 + 1$  (0x1021)

The syndrome is initialized to all ones (0xFFFF) at the beginning.

It is the same CRC polynomial and size as used in various protocols such as Bluetooth, X.25 or CCSDS stack.

### 7.1 SOFTWARE IMPLEMENTATION

The following C-language code describes the software routine to implement the CRC encoder. To implement the CRC decoder, the same routines can be used: data and the syndrome are encoded, and the resulting syndrome should be equal to zero if no error is present.

Functions applicable to generate the CRC placed at the end of a packet:

- Crc function calculates the CRC for one byte in serial fashion and returns the value of the calculated CRC checksum.
- Crc\_opt function can be used instead of the Crc function given above. The Crc\_opt function generates the CRC for one byte and returns the value of the new syndrome. This function is approximately 10 times faster than the non-optimized Crc function.
- InitLtbl function initiates the look-up table used by Crc\_opt.

```

unsigned int Crc(Data, Syndrome)
unsigned char Data; /* Byte to be encoded */
unsigned Syndrome; /* Original CRC syndrome */
{
    int i;
    for (i = 0; i<8; i++) {
        if ((Data & 0x80) ^ ((Syndrome & 0x8000) >> 8)) {
            Syndrome = ((Syndrome << 1) ^ 0x1021) & 0xFFFF;
        }
        else {
            Syndrome = (Syndrome << 1) & 0xFFFF;
        }
        Data = Data << 1;
    }
    return (Syndrome);
}

unsigned int Crc_opt(D, Chk, table)
unsigned char D; /* Byte to be encoded */
unsigned int Chk; /* Syndrome */
unsigned int table[]; /* Look-up table */

{
    return (((Chk << 8) & 0xFF00) ^ table[(((Chk >> 8) ^ D) & 0x00FF)]);
}

void InitLtbl(table)

```

```

unsigned int table[];
{
    unsigned int i, tmp;
    for (i = 0; i<256; i++) {
        tmp = 0;
        if ((i & 1) != 0) tmp = tmp ^ 0x1021;
        if ((i & 2) != 0) tmp = tmp ^ 0x2042;
        if ((i & 4) != 0) tmp = tmp ^ 0x4084;
        if ((i & 8) != 0) tmp = tmp ^ 0x8108;
        if ((i & 16) != 0) tmp = tmp ^ 0x1231;
        if ((i & 32) != 0) tmp = tmp ^ 0x2462;
        if ((i & 64) != 0) tmp = tmp ^ 0x48C4;
        if ((i & 128) != 0) tmp = tmp ^ 0x9188;
        table[i] = tmp;
    }
}

/* Simple program to test both CRC generating functions */
void main()
{
    unsigned int Chk; /* CRC syndrome */
    unsigned int LTbl[256]; /* Look-up table */
    unsigned char indata[32]; /* Data to be encoded */
    int j;
    indata[0] = 0x31; indata[1] = 0x23; indata[2] = 0x48; indata[3] = 0x07;
    indata[4] = 0x00; indata[5] = 0xEC; indata[6] = 0xD0; indata[7] = 0x37;
    Chk = 0xFFFF; /* Reset syndrome to all ones */
    for (j = 0; j<8; j++) {
        Chk = Crc(indata[j], Chk); /* Unoptimized CRC */
    }
    printf(" CRC = %x(should be 0)\n", Chk);
    InitLtbl(LTbl); /* Initiate look-up table */
    Chk = 0xFFFF; /* Reset syndrome to all ones */
    for (j = 0; j<8; j++) {
        Chk = Crc_opt(indata[j], Chk, LTbl); /* Optimized CRC */
    }
    printf(" CRC = %x(should be 0)\n", Chk);
}

```

A different implementation could be the following:

```

// Computes the CRC-CCITT
// data          The buffer containing the data bytes.
// dataLength    The number of bytes to read from the buffer.
// init          Initial syndrome value.
uint16_t CRC_Compute(const uint8_t* data, uint16_t dataLength, uint16_t init)
{
    uint16_t x;
    uint16_t crc = init;

    while (dataLength--)
    {
        x = crc >> 8 ^ *data++;
        x ^= x >> 4;
        crc = (crc << 8) ^ (x << 12) ^ (x << 5) ^ (x);
    }
    return crc;
}

```

## 7.2 VERIFICATION OF COMPLIANCE

The binary sequences defined in this subclause (see table below) are provided to the end user as samples for early testing, so that they may verify the correctness of their CRC error-detection implementation. All data is given in hexadecimal notation. For a given field (data or CRC) the leftmost hexadecimal character contains the most significant bit.

Data	CRC
00 00	1D 0F
00 00 00	CC 9C
AB CD EF 01	04 A2
14 56 8 9A 00 01	7F D5



## 8 DISCLAIMER

The information contained within this document (the "Astronode Pilot Asset API User Manual") is furnished for informational purposes only. Even though Astrocast SA does its best to deliver this Astronode Pilot Asset API User Manual with correct and complete information, we cannot warrant that this document is free from any errors, inaccuracies, or omissions. We reserve the right to make additions, deletions, or modification to the content of the Astronode Pilot Asset API User Manual at any time.

Please make sure that you carefully read this information before using our products and ask us for support in case of any questions or doubts. Astrocast SA shall not be liable for any damages, losses, costs, or expenses, direct, indirect, or incidental, consequential, or special, arising out of or related to the use of this document or the incorrect use or operation of our products.