Reference Guide

Ayla Module and MCU Interface Specification

Version: 12.0

Date Released: March 8, 2018

Document Number: AY006MSP3-12

Ayla Networks

Copyright Statement

Trademarks Statement

Referenced Documents

Ayla Networks does not supply all documents that are referenced in this document with the equipment. Ayla Networks reserves the right to decide which documents are supplied with products and services.

Contact Information

Ayla Networks TECHNICAL SUPPORT and SALES
Contact Technical Support: https://support.aylanetworks.com
or via email at support@aylanetworks.com
Contact Sales: https://www.aylanetworks.com/company/contact-us
Ayla Networks REGIONAL OFFICES

GREATER CHINA
Shenzhen
Room 310-311
City University of Hong Kong
Research Institute Building
No. 8 Yuexing 1st Road
High-Tech Industrial Park
Nanshan District
Shenzhen, China
Phone: 0755-86581520

HEADQUARTERS
Silicon Valley
4250 Burton Drive, Suite 100
Santa Clara, CA 95054
United States
Phone: +1 408 830 9844
Fax: +1 408 716 2621

JAPAN
Wise Next Shin
Yokohama, 2-5-14
Shnyokohama, Kohokuku
Yokohama-shi, Kanagawa-ken
Yokohoma, 222-0033 Japan

EUROPE
London
30 Great Guildford St
London SE1 0HS
United Kingdom

TAIWAN
Taipei
5F No. 250 Sec. 1
Neihu Road, Neihu District
Taipei 11493, Taiwan

For a Complete Contact List of Our Offices in the US, China, Europe, Taiwan, and Japan:
https://www.aylanetworks.com/company/contact-us

Table of Contents

Revision History

Revision	Date	Change Description
2	2015-09-15	Updated with switch region feature
4	2016-02-01	Updated with user registration notification feature
5	2016-02-02	Updated with Wi-Fi power feature
6	2016-02-16	Updated with Datapoint Metadata feature
7	2016-03-27	Updated with Datapoint Acknowledgement feature
8	2016-5-23	Updated byte numbering in sending property with metadata section
9	2016-6-6	Updated Ack_ID TLV, length to 15
10	2017-04-19	Removed description of unimplemented floating-point and BCD TLVs. Reformatted to new Ayla styles.
11	2017-06-10	Added table to Section 8.and updated Table 1.
12	2018-01-31	Added tokens to send setup token, fixed WiFi Status-Final

1 Introduction

This document describes the over-the-wire protocol used to connect a Microcontroller Unit (MCU) to the Ayla Module using SPI (serial peripheral interface) or UART (universal asynchronous receiver/transmitter).

1.1 Audience

This document is written for engineers and includes:

- Introduction to the concepts of properties and datapoints
- High-level description of packet protocols
- Description of SPI protocol
- Description of UART protocol
- Details of TLV (type / length / value items)
- Data and control protocols

1.2 Properties and Datapoints

The Ayla Module sends and receives name/value pairs between the MCU and the Ayla Device Service (ADS). Names are called properties. Values are called datapoints.

A property and its datapoints can represent anything that the MCU defines. The property may represent the state of a light or switch, a temperature sensor reading, the desired thermostat temperature, a moisture sensor, a schedule for a sprinkler system, an input to a calculation, or just about anything.

Property names are ASCII strings (up to 27 characters):

- Upper and lower-case letters
- Numbers
- Hyphens (-)
- Underscores (_)
- Spaces or special characters are not allowed
- First character must be alphabetic

Properties have a value data type, such as integer, Boolean, string (UTF-8), or binary. Property datapoints are values associated with the property (usually the most recent datapoint).

Most datapoints have values are less than 255 bytes (maximum that can be passed in a single message). A datapoint can be up to 4 GB long.

1.3 Protocol Overview

A host MCU connects to an Ayla Module with SPI or UART. The interface is designed to work with variety of MCU sizes. Some modules may support only one of these modes.

Most of these documented capabilities are for module configuration and are optional. The MCU may also perform module configuration during manufacturing.

Whether SPI or UART is used for communications, three levels of protocols are:

- Single-byte simple commands to get status or start a transfer.
- Packet transfer command sequences.
- Data or control protocol packets with a header and optional TLVs.

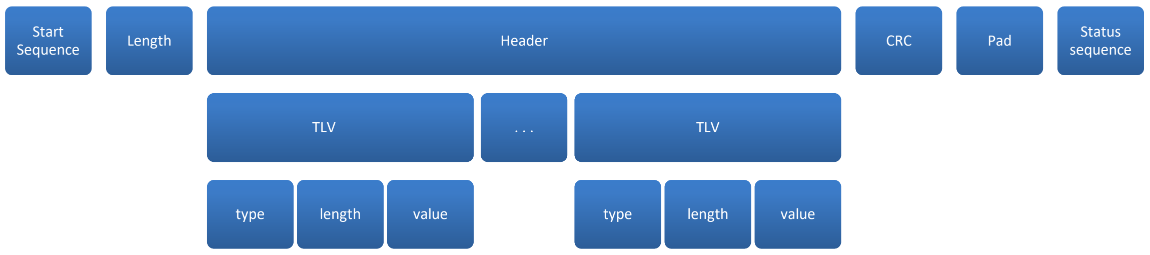
Figure 1 shows phases of a packet transfer sequence. The first row of boxes shows the sequence. The Start sequence, pad, and status sequence are different for SPI and serial, but Length, Packet and CRC (cyclic redundancy check) are similar.

Every packet transfer includes:

- Start sequence (single-byte commands and status exchanges)
- Length field
- Packet being sent
- Cyclic redundancy check (CRC)

Padding bytes may be required before the status sequence is performed.

Figure 1. Protocol Hierarchy for SPI



The first row shows the packet and its encapsulation. The second row shows the header and TLVs forming the packet. Third row shows TLV contents (type, length, value).

2 SPI Protocol

The SPI protocol has two roles, called master and slave.

NOTE	The MCU is the SPI master and the module is the SPI slave.

With SPI, the MCU initiates all communications. When the module is initialized, the READY_N signal is asserted. It is recommended to wait for the READY_N signal before attempting to use SPI.

After initialization, every time the MCU sends a single-byte command to the module, the module returns a one-byte status in this format:

Bit 7 (MSB)	6	5	4	3	2	1	Bit 0 (LSB)
Invalid	-	-	-	ADS Busy	Error	Attention	Busy

- **Bit 7: Invalid (must be zero)** - status values from message start byte. MCU checks this bit. If set, ignores the rest of the status.

  Helps detect bugs (for example, module sends all ones or MISO line is floating).
- **Bits 6 - 4: Reserved** - MCU ignores these bits.
- **Bit 3: ADS Busy** - if set, module cannot accept a data service operation.

  An operation may be in progress, or the link may be down.
- **Bit 2: Error** – an error on last message (for example, overrun, underrun, or other error).
- **Bit 1: Attention** - set when module has data the MCU should fetch.
- **Bit 0: Busy** - module sets this after a command is recognized, and clears it on completion.

  A new command must not be sent unless a non-Busy status is seen first. As soon as the new command starts, Busy status should be set.

2.1 Single-Byte Commands

Single-byte commands have the following format:

Bit 7 (MSB)	6	Bits 5 thru 0
Transfer	Direction	Length / 8 for transfer to slave, opcode for other commands.

- **Bit 7: Transfer** - set when command is a data transfer.
- **Bit 6: Direction** - transfer direction (0 = MCU to module, 1 = module to MCU).
- **Bits 5 – 0: Length** - payload length (in 8 byte units)

With this format, these single-byte commands are defined:

- **NUL (0)** - used to poll status
- **Start Master Out: (0x80 thru 0xbf)** - begin transfer from MCU to module

- **Start Master In: (0xf1)** - begin transfer from module to MCU
- **Start Retry Master In: (0xf2)** – (for error recovery) repeat of previous transfer from module to MCU

All other commands are reserved.

NOTE To form the Start Master Out command, round the packet length up to the next higher multiple of 8, then divide by 8 and add the constant 0x80.

Length in the command allows module to start DMA before additional bytes are received.

The packet length that forms the command does not include the command byte itself or the following two bytes of length, but does include the CRC.

2.2 Packet Transfers

The MCU initiates all transfers. The module notifies the MCU of a transfer – by setting the attention bit and asserting the INTR_N signal.

MCU controls the clock and, when data transfer completes, stops the clock.

2.2.1 Packet from MCU to Module

Transfers from the MCU to module start with the Master Out command byte:

1. **Wait not busy** - MCU sends 0 bytes to the module and receives the Status.

   This is repeated until it sees a valid Status (bit 7 clear) and Not Busy (bit 0 clear). For some commands, ADS Busy (bit 3) must also be clear.
2. **Start** - MCU sends Start Master Out command.

   This contains the transfer length (8-byte multiples). Estimated length includes packet and CRC, rounded up to next multiple of 8. Opcode is 0x80 + (len + 7) / 8.
3. **Wait busy** – Each time MCU sends Start command, it receives the module Status byte.

   This repeats until it sees the Busy status (indicates module sees the command and is ready for the next part of the transfer).
4. **Length** - MCU sends the length (in bytes, MSB first) of the packet and CRC. (Does not include command bytes or length.)

   Value may be 1 thru 384, inclusive.
5. **Packet** - MCU sends packet followed by CRC-8.
6. **Padding bytes** - MCU sends zero bytes to pad transfer to the rounded-up length that was used in the command byte.
7. **Status** - MCU sends zero bytes and waits for non-Busy status - and checks for Error status.

2.2.2 Packet from Module to MCU

Transfers from module to MCU start at the MCU after it sees an interrupt from the module and/or an Attention status from module. Transfer sequence is:

1. **Wait not busy** - MCU sends 0 bytes to module and receives status.

Repeats until MCU sees a valid Status (bit 7 clear) and Not Busy (bit 0 clear) and Attention (bit 1).

2. **Start** - MCU sends **Start Master In** command, 0xf1. Received byte from module is examined.

   MCU repeats this until 0xf1 byte is received.
3. **Module replies with Start** - module sends Start command byte to MCU.

   Indicate length bytes to follow.
4. **Length** - module sends transfer length (in bytes, MSB first), of packet and CRC-8.

   Value may be 1 thru 384, inclusive.
5. **Payload** - MCU sends zero bytes to clock in the rest of the transfer.

   Module sends the packet and CRC.

2.2.3 Retry Packet from Module to MCU

If MCU detects CRC error, the last transfer from the module can be retried (uses same procedure as initial receive - except start byte is 0xf2).

MCU sends 0xf2 start byte until it sees 0xf1 byte from module. Procedure is the same as steps 3 through 5 above.

2.3 Error recovery

Possible error cases are:

- **Slave overrun in data** – MCU sends data faster than the module can get it from data registers.

  The module indicates transfer complete with the status ERROR flag – and immediately ready for another transfer. When this occurs frequently – if using programmed I/O, slow the MCU transfer rate. If using DMA, reduce SPI clock rate.
- **Slave overrun in length** - same as overrun in data.
- **Slave unresponsive** - If module does not set Busy flag soon after MCU sends the Start byte (indicates a failure in module firmware).

  Recovery action may involve a module reset.
- **Master underrun** – MCU does not send expected length to module.

  Can occur if MCU firmware error or module overrun in length transmission. If this, MCU can send zeros until module Busy flag clears.
- **Slave underrun** – On transfers (module to MCU), MCU paces the transfer - and module could not supply data fast enough.

  Unlikely after length bytes are sent because module uses DMA. Problem is because data is still being transferred and MCU unable to determine a valid Status byte or just data. The MCU would detect this as a CRC error. When this occurs frequently – if using programmed I/O, slow the MCU transfer rate. If using DMA, reduce SPI clock rate.

- **CRC error** - should not occur with functional hardware and software. If occurs in either direction, retry message several times - and count in error statistics.

3 Asynchronous Serial Protocol (UART)

UART is an alternative to SPI. The module can be configured to use UART to communicate. For some MCUs, UART can be more convenient.

By default, UART operates at 115,200 bits/sec (module can be configured with different speeds). It uses 8-bit characters, odd parity, and 1 stop bit.

The interface requires hardware flow control. Each side uses flow-control signals to send messages.

The serial protocol has framed and encapsulated packets similar to SPI protocol packets.

3.1 Framing

Each packet is framed using a scheme similar to that used by PPP, as described in RFC-1622 section 4. Ayla does not use the address and control bytes that PPP defines, however. It isn't necessary to be familiar with that RFC or PPP to understand the framing.

Each frame, in each direction, begins and ends with a flag byte (0x7e).

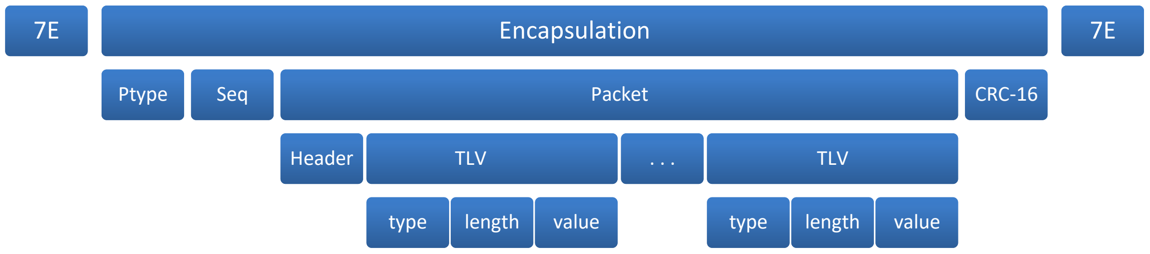
Any flag byte occurrence in a packet is replaced by the two-byte sequence 0x7d 0x5e. 0x7d is the escape byte. Any 0x7d in packet is replaced by 0x7d 0x5d. The second byte of this escape sequence is the original byte XORed with 0x20.

Only one flag byte is necessary between frames (sometimes a flag byte is sent at the frame beginning, if not sent in a while). Two flag bytes in a row imply an empty frame (to be discarded).

3.2 Encapsulation

Each packet is encapsulated with a packet type with a sequence number before the packet and a CRC-16 code after the packet. Then it is framed.

Figure 2. Protocol Hierarchy for UART



The encapsulation between flag bytes is shown here:

Bytes	Name	Meaning
0	Ptype	Packet type code
1	Seq	Sequence number
2 to N+1	Payload	Data or Control packet
N+2 to N+3	CRC-16	Redundancy check over payload, MSB first

The packet (header and TLVs) are same as in SPI protocol.

CRC-16 is computed over all bytes after the flag byte, before byte-stuffing for transmit. This is done with a CRC-generator using the CCITT polynomial. It is initialized to 0xffff and computed with the payload's MSB. CRC is placed in network byte order after the payload. A CRC payload check and the transmitted CRC results in a return value of 0.

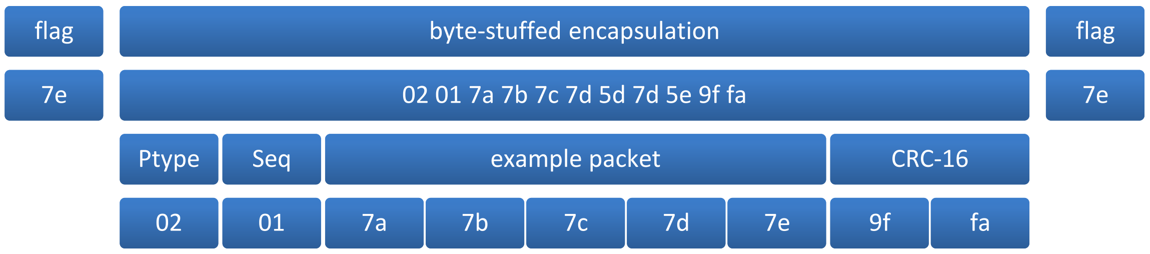
To prepare a packet for transmission, the following process takes place:

1. Start with the payload to be sent (for example, 7a 7b 7c 7d 7e).
2. Prepend the ptype (e.g., 0x02) and sequence number (e.g., 0x01).
3. Compute CRC-16 over the ptype, sequence number and payload and append it. (CRC in this case would be 0x9ffa.)

   At this point, the frame is: 02 01 7a 7b 7c 7d 7e 9f fa. (5 bytes plus 4 bytes of overhead.)
4. Perform octet stuffing and add flag bytes.

   At this point, the frame is: 7e 02 01 7a 7b 7c 7d 5d 7d 5e 12 34 7e.
5. These 13 bytes are sent. 5 (data) + 4 (length and CRC) + 2 (byte stuffing) + 2 (flag bytes) = 13.

Figure 3. Example packet



3.3 Packet Types

These are the currently-defined packet types. All other packet types are reserved and should be ignored if received by the MCU.

Ptype	Name	Meaning
0x01	Data	The packet is a data, control or ping packet.
0x02	ACK	Acknowledges packet receipt indicated by the sequence ID.

3.4 Error Handling

Errors detectable on the serial protocol include CRC mismatch, invalid frames, invalid Ptype, overruns, and parity errors. These should be counted, but otherwise ignored.

When receiver gets a data packet, an ACK (acknowledge) packet is sent with the same sequence number.

A missing ACK (timeout) causes retransmission (at least the first two times it happens) on a given sequence ID. The module drops packets after two retries. The MCU may reset the module after too many retries, or take other appropriate action. Sender should ensure the packet really was transmitted and not just unable to be sent due to flow-control.

A packet received with the same sequence number as the previous packet (but not zero) should generate an ACK, but otherwise be dropped. This is how lost ACKs are handled.

The special sequence number zero indicates sender has restarted. Both MCU and module start with a zero-sequence number (after starting, zero is never reused) - called a "lollipop" sequence number scheme.

4 TLV format

Most messages contain one or more TLVs (type / length / value items). The TLVs have the following format:

- **Byte 0** - type code
- **Byte 1** - length (N) of the value, not including type or length
- **Bytes 2 thru (N+1)** - value

Table 1. TLV Type codes

NOTE: Highlighted rows are TBD (to be determined).

TLV Type	Name	Description
0x01	Name	Property name, without NUL termination.
0x02	Integer	1-, 2-, 4-, or 8-byte signed integer value, in network byte order.
0x03	Unsigned Integer	1-, 2-, 4-, or 8-byte unsigned integer value in network byte order. Note that this type is never used in the protocol, only internally on the MCU. Unsigned integers greater than 2^{31}-1 are sent as 8-byte signed integers.
0x04	Binary	Unstructured data bytes.
0x05	Text	UTF-8 encoded text value, without NUL termination.
0x06	Configuration tokens	UTF-8 encoded configuration tokens.
0x07	Error	Error number, used in a NAK packet.
0x08	Format	(optional) Set of flags how the current named value should be formatted.
0x0b (TBD)	Floating point	Floating point value
0x0f	Boolean	One-byte value of 1 or 0.
0x10	Continuation token	4-byte value used in stepping through tables or lists.
0x11	Offset	Byte offset of the start of the value TLV that follows. (For large datapoints.)
0x12	Length	Total length of a datapoint value
0x13	Location	Location string identifying datapoint.
0x14	EOF	End-of-file for datapoint, e.g., length is always 0.
0x15 (TBD)	BDC	Decimal number. NOTE: BCD TLVs are not currently implemented. The BCD TLV is coded as a one-byte number of digits before the decimal point, followed by 4-bit coded digits. Negative numbers start with the digit code 0xf. For example, TLV for the value -123.45 is shown in Figure 2.

TLV Type	Name	Description
0x16	Cents	Fixed-point integer 100 times actual value. For example, 98.72 is 9872 decimal. Usually 4-bytes.
0x17	Nodes	One-byte bitmap of destinations or sources for property updates. Bit 0 is device service, other bits assigned for local destinations.
0x18	Echo	Indicates prop update is an echo. Length is always 0.
0x19	Feature Mask	One-byte bitmap of the supported features in MCU.
0x1a	Factory Reset	Configuration name indices.
0x1b	Delete Configuration	UTF-8 encoded configuration tokens.
0x1c	Registration	Contains single 1-byte value set to 1 or 0. 1 = device registered with a user 0 = device deregistered.
0x1d	Event Mask	Module sends this to indicate an event. Supported event: - standby power mode: Module goes into standby mode. (To wake up, reset MCU.)
0x1e	Ack ID	• When from module to MCU: contains acknowledgement (ack) ID received for an ack-enabled property. • When from MCU to module: contains ack ID, ack status and ack message.
0x20	Schedule	Schedule property composed of Schedule-related TLVs.
0x21 to 0x33	Schedule TLV	Schedule-related TLVs. See Table 1.1 below.
0x34	Datapoint Metadata	Key of the key-value pair metadata for a property datapoint update.
0x35	Wi-Fi Status	A Wi-Fi status event - success or failure to join a Wi-Fi network. This TLV contains: • one-byte value of number of characters in the SSID • <n> bytes containing the SSID as a UTF8 value • Status code See Table1.2 below.
0x36	Wifi Status Final	Sends final Wi-Fi status to MCU that module is done trying a wifi join. (Used for firmware version 2.8+.)

Table 1.1. Schedule-related TLV Type codes

TLV Type	Name	Description
0x21	UTC	Indicates date/time in schedules are UTC
0x22	AND	ANDs the top two conditions in schedule
0x23	Disable	Disables the schedule

TLV Type	Name	Description
0x24	In_Range	Stack is true if current time is in range
0x25	At_Start	Stack is true if current time is at start
0x26	At_End	Stack is true if current time is at end
0x27	Start Date	Date must be after value
0x28	End Date	Date must be before value
0x29	Days of Month	32-bit mask indicating which day of month. Bit 0 is the first day, bit 31 represents the last day of the month, which may be day 28, 29, 30, or 31, depending on the month and year.
0x2a	Days of Week	7-bit mask indication which day of the week. Bit 0 is Sunday. Bit 7 is reserved.
0x2b	Day Occur In Month	Day Occurrence in Month. Bit 0 is the first occurrence.
0x2c	Months of Year	Months of Year, 1 through 12.
0x2d	Start Time Each Day	Start Time for each valid day of schedule
0x2e	End Time Each Day	End Time for each valid day of schedule
0x2f	Duration	Event duration
0x30 (TBD)	Time Before End	Time must be <value> secs before end.
0x31	Interval	Start every <value> secs from start
0x32	Set Prop	Value is a TLV pair of the action to take
0x33	Version	Version of Schedule

Refer to the <ayla_proto_mcu.h> header file in the API for definitions of these type codes.

NOTE BCD TLVs are not currently implemented. The BCD TLV is coded as a one-byte number of digits before the decimal point, followed by 4-bit coded digits. Negative numbers start with the digit code 0xf. For example, TLV for the value -123.45 is shown in Figure 4.

Figure 4. Example TLV representation

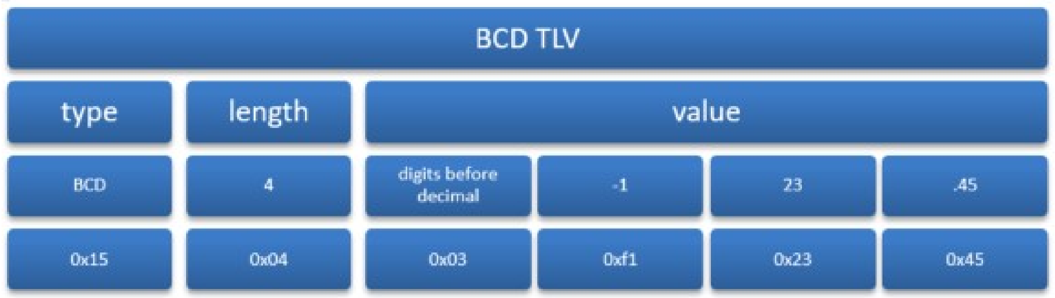


Table 1.2. Status values in Wi-Fi status (0x35) event

Status	Name	Description
0x00	Success	Module joined Wi-Fi network identified by SSID.
0x01	Resource Unavailable	Resource unavailable to complete join operation (may be temporary).
0x02	Connection Timed Out	Connection to AP timed out. May indicate a few underlying causes – for example, inconsistent visibility of the AP due to range or noise; too many other APs in the vicinity; incorrect Wi-Fi password.
0x03	Invalid Key	Wi-Fi password supplied to module was not valid for configured security method on AP.
0x04	Network Not Found	Module could not find network with specified SSID. Can occur if many APs in vicinity with strong signals - and a scan failed to find requested SSID.
0x05	Not Authenticated	Attempt to authenticate with the AP failed. May indicate connection to the AP was lost, or provided password was incorrect.
0x06	Wrong key	Wi-Fi password was incorrect. Whether an AP indicates this error or a Not Authenticated error is specific to AP.
0x07	No IP Address	Failed to obtain IP address from AP with DHCP.
0x08	No Route	Failed to obtain default route from AP with DHCP.
0x09	No DNS Server	Failed to obtain DNS Server address from AP with DHCP.
0x0a	AP Disconnected	AP disconnected from module. May indicate AP cannot reliably receive data from module, or AP was restarted or powered off.
0x0b	Loss of Signal	Module failed to receive consecutive beacons from AP. Module cannot reliably receive data from AP, or AP was restarted or powered off.
0x0c	DNS lookup failed	OEM-based hostname used to reach the ADS was not resolved to an IP address.
0x0d	ADS Connection Redirect	Connection to ADS failed due to an HTTP redirect.
0x0e	ADS Connection Timeout	Connection to ADS timed out. May be a failure in network beyond Wi-Fi AP, or failure to communicate with AP.
0x0f	No Profile Available	Connection to the Wi-Fi network succeeded, but configuration was not saved, because too many configured Wi-Fi profiles.
0x10	Security Method Not Supported	Module or Access Point does not support security method specified in Wi-Fi join request
0x11	Network Type Unsupported	Wi-Fi network type (e.g., Ad-Hoc, Enterprise) not supported by module.

Status	Name	Description
0x12	Wi-Fi protocol error	Access Point protocol not supported. Error can occur on attempt to connect to a personal hotspot, or device in tethering mode.
0x13	ADS authentication error	Module failed to authenticate with ADS. Report error immediately to Ayla Customer Service (module code may need to be updated).
0x14	Operation In Progress	Wi-Fi join operation still in progress. Subsequent status event indicate success or failure when join operation completes.
0x15	Setup Unsupported	Module is in setup mode. Join Wi-Fi network only permitted in user mode.

5 Packets

This section describes the common packet format for all protocols.

Messages in either direction start with one-byte protocol number.

For most protocols, this is followed by a per-protocol opcode and other payload data.

- Byte 0: protocol
  - 0x00 - reserved for Ayla Configuration and Control Operations
  - 0x01 - used for Ayla Data Operations
  - 0x02 - used for SPI Ping test

6 SPI Ping Protocol

The ping protocol is a simple way to test the SPI interface from MCU. Any packet sent to module with protocol 2 is returned to MCU. The procedure is to send the packet, then poll for ATTN in the status, and receive the identical reply packet.

7 Data Operations

MCU or module can issue data operations. From MCU, request is sent to the ADS or other server, as configured. From module, requests can come from ADS or from the module's web server.

The packet format is:

- **Byte 0: Protocol (1)** - protocol for data operations is 1.
- **Byte 1: Opcode** - Opcodes for these operations are defined in the header file <ayla_spi_mcu.h> as an exported part of the API.
- **Bytes 2 - 3: Request ID** - Request ID associates responses and NAKs with the request. Only one request from MCU should be outstanding at a time.

7.1 Data Operations Opcodes

Table 2. Implemented Data Operations opcode

Data Opcode (hex)	Description	Sent by
01	(Obsolete) Send TLVs version 1	MCU
02	Property value request	MCU
03	Receive TLVs	Module
05	Negative Acknowledge (NAK)	Module
06	Send Property	Module
07	Response to Send Property Request	MCU
08	Send Next Property	Module
09	Send TLVs	MCU
0a	File Datapoint Request	MCU
0b	File Datapoint Response	Module
0c	File Datapoint Create	MCU
0d	File Datapoint Fetched	MCU
0e	File Datapoint Stop	MCU
0f	File Datapoint Send	MCU
11	Connectivity status	Module
12	Echo Failure	Module
13	Enable Service Listener	MCU
14	Error	Module
15	Confirm	Module

Data Opcode (hex)	Description	Sent by
16	Property Notification	Module
17	Event Notification	Module

(obsolete) Opcode 0x01: Send TLV V1: Send Data Service Properties

This operation is superseded by opcode 9 (Send TLV).

Opcode 0x02: Property value request

(MCU to module) This packet asks module to GET a property value from ADS.

Message can contain a Name TLV to get a single property value. Module responds with opcode 3.

If message has no Name TLV, values of all to-device properties are fetched from ADS. After all of the to-device properties are sent to MCU, an EOF packet ends the message.

Opcode 0x03: Receive TLV: Receive TLV values from Data Service

(module to MCU) Delivers new property values.

(optional) If Ack_ID (Acknowledgement) TLV included, explicit ack is needed for datapoint update. MCU uses ack ID to acknowledge datapoint after execution.

(optional) Nodes TLV indicates which node originated the property value. If not present, property is assumed to originate from ADS.

Opcode 0x05: Negative Acknowledge (NAK)

(module to MCU) NAKs inform MCU of an unsuccessful data operation.

- NAK contains an Error TLV to define problem (for example, timeout or incorrectly formed request).
- If NAK is for a failed property Post or Get, it includes a Name TLV with property.
- Node TLVs have a bitmask of failed destinations / sources.

Opcode 0x06: Request Property

(module to MCU) Module requests a specific property.

Request has a Name TLV to specify the property. MCU must respond with opcode 7.

If module does not provide a ATLV_NAME property, MCU responds with a NAK and Feature Mask TLV (8-bit representation of MCU supported features).

For the feature mask, use OR to put together the following values based on Host MCU-supported features:

Mask Value	Feature
0x01	LAN mode
0x02	Host MCU OTA
0x04	Time Subscription

Mask Value	Feature
0x08	Datapoint confirmation

Opcode 0x07: Response to Send Property Request

This contains the property name and value requested by Send Property or Send Next Property.

Opcode 0x08: Request Next Property

(module to MCU) This contains a Continuation Token TLV.

If value is 0, the first property is requested. Response uses opcode 7 (Response to Send Property Request).

If more properties, response contains a continuation token. The value gets the next property in MCU-maintained sequence.

Opcode 0x09: Send TLV: Send Data Service Property

(This operation replaces opcode 1.)

(MCU to ADS) This provides the service property's current value.

The property name TLV and value TLV are included. (If needed, Value TLV may be preceded by a Format TLV.) The value is sent to ADS as a new datapoint.

(optional) MCU can send a Nodes TLV (0x17) to provide datapoint destinations. If no Nodes TLV is sent, the datapoint is sent to all currently reachable destinations.

If this is a to-device property, Echo TLV (0x18) is included because this is in response to an earlier source property update.

While ADS_BUSY status is set, do not use this operation. ADS_BUSY is set by this and remains set until ADS receives the new datapoint.

This opcode can also send acknowledgement (ack) status of an ack-enabled property back to the MCU. This returns the ack ID received from the module for the:

- datapoint update
- ack status (success or failure)
- ack message (a unique number that indicates device execution results).
- property name and value
- source of the datapoint (ATLV_NODES)
- Echo TLVs (appended in case of datapoint acks)

  Echoes are delayed till the datapoint has been executed. (MCU is required to initiate echoes.)

  Module sends echoes to all destinations except datapoint source. An acknowledgement is sent to the source.

Opcode 0x0a: File Datapoint Request

MCU sends this packet to request data for a file datapoint.

- Location TLV specifies a file datapoint.

- Offset TLV can be specified.
- Length TLV can be specified to limit response length.

Opcode 0x0b: File Datapoint Response

When module receives a file datapoint request, it sends this packet.

Opcode 0x0c: File Datapoint Create

MCU sends this packet to create a new file datapoint on ADS.

A Name TLV is included with the property name. The module replies with the Name TLV and a Location TLV (used in subsequent File Datapoint Send operations).

A datapoint-create operation must be followed by a datapoint-send operation (0x0f). If not, the unexpected operation error (0x14) is returned.

Opcode 0x0d: File Datapoint Fetched

MCU sends this packet to confirm the complete fetch of a file datapoint.

Request contains the Location TLV for the file datapoint.

Opcode 0x0e: File Datapoint Stop

MCU sends this packet to abort an ongoing file operation.

Opcode 0x0f: File Datapoint Send

MCU sends data for a file datapoint.

Included TLVs are:

- Location TLV
- Offset TLV
- Length TLV (on the first packet, if known)
- EOF TLV

This operation must be called immediately after datapoint-create is complete. If not, the unexpected operation error (0x14) is returned.

Opcode 0x11: Connectivity Status

Module sends this packet to update MCU on reachability of the service and local applications receiving property updates.

Local applications typically run on mobile devices with access to the module over the local area network. The message contains ATLV_NODES TLV with a bitmask of all available destinations.

Opcode 0x12: Echo Failure

(module to MCU) Module sends this packet to let MCU know of a failed auto-echo to ADS for a property.

Message contains Name TLV of the failed property echo.

Opcode 0x13: Enable Service Listener

MCU sends this when ready to listen to property and commands from ADS.

Incoming connectivity status messages (0x11 above) update MCU when module has established service connectivity. MCU sends out pending property echoes before this packet is sent.

Opcode 0x14: Error

(module to MCU) Informs MCU an error occurred.

Opcode request ID is always 0. One or more Error TLVs are included.

For example, AERR_OVERFLOW occurs when MCU is too slow in processing property updates from ADS. For this error code, the recommended action is that the MCU requests all to-device properties from ADS to ensure latest values are known.

Opcode 0x15: Confirm

This is used only in UART mode. It confirms a successful property post to ADS and/or mobile app.

Opcode 0x16: Property Notification

When a property update is pending in ADS, module sends this to MCU.

Opcode 0x17: Event Notification

Module send this to MCU to notify of a device-related event.

Notification is not ack'ed by MCU. Contains one or more TLVs with event information. The events are considered "in order".

7.2 Example Data Messages

This section provides message examples of messages of various operations. The start and length bytes from the framing protocol are also shown. (Zero padding bytes at the end are not shown.)

7.2.1 Send a property

(SPI – MCU to module). Value of 1 is sent for the name "led0":

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
82	00	0e	01	09	12	34	01	04	6c	65	64	30	0f	01	01	CRC
Start	Length (14)		Protocol	Opcode	Request ID		Name TLV, Length 4, Value "led0"						Boolean TLV, Length 1, Value 1			

Start byte 0x82 indicates packet up to 16 bytes long. May be repeated until Busy status is seen. Then the length bytes, 0x000e, for 14 bytes of overall payload length (includes CRC, but not start command or length bytes).

Payload begins with protocol 1 (for Ayla Data protocol), followed by opcode 9, "Send TLV", and arbitrary request ID 0x1234.

Name and value TLVs follow.

- Name TLV are type 1 (name), length 4, followed by the 4 bytes of the name "led0".
- Value TLV (in this example) holds the one-byte Boolean value of 1, so the type is 15 and the length is 1.

Each TLV has only 1 byte for the "length". Maximum TLV length is 255 bytes (more than enough for most property types). If application requires longer string values (ATLV_UTF8) see "Send Long Strings".

7.2.2 Send property with metadata

(SPI – MCU to module) A value of 1 is sent for the name "led0" with datapoint metadata key-value pair (key "key1" and value "value1").

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	17	18	19	20	21	22	23	24
82	00	1f	01	09	12	34	34	04	6b	65	79	31	05	06	76	61	6c	75	65	31
Start	Length (31)		Protocol	Opcode	Request ID		Datapoint Metadata TLV, Length 4, Key "key1"						UTF8 TLV, Length 6, Value "value1"							

25	26	27	28	28	29	30	31	32	33
01	04	6c	65	64	30	0f	01	01	CRC
Name TLV, Length 4, Value "led0"						Boolean TLV, Length 1, Value 1			

Start byte 0x84 indicates packet is up to 32 bytes long. May be repeated, until Busy status is seen.

Length bytes, 0x001f, for 31 bytes of overall payload length (includes CRC, but not start command or length bytes.

Payload begins with Protocol 1 (Ayla Data protocol), followed by opcode 9, "Send TLV", and arbitrary request ID 0x1234.

Metadata TLV is next (must always precede name and value TLV). Datapoint metadata are key-value pairs. Metadata TLV contains the key and followed by UTF8 TLV with the value. Metadata TLV has type 52 (ATLV_DPMETA), length 4, followed by bytes of "key1". The UTF8 TLV of type 5, length 6, holds the bytes of "value1". The MCU can send up to four metadata key-value pairs. Maximum allowed length for Key is 16 bytes, and for Value is 32 bytes.

Name and value TLVs follow.

- Name TLV are type 1 (name), length 4, followed by the 4 bytes of the name "led0".
- Value TLV (in this example) holds the one-byte Boolean value of 1, so the type is 15 and the length is 1.

7.2.3 Send long strings

Maximum string length is module-dependent. Current modules (version bc-1.8 +) limit strings to 1024 bytes.

Some byte values count as multiple bytes towards the limit.

The following are sent as single bytes, but count as 2-bytes:

- 0x08 (backspace)
- 0x0a (newline)
- 0x09 (tab)
- 0x0c (form-feed)
- 0x0d (carriage-return)
- 0x22 (") (double quote)
- 0x5c (\) (backslash).

Bytes of value 0 thru 0x1f, other than mentioned, count as 6 bytes towards the limit. For example, a property of only backslashes is limited to 512 bytes.

To send a string >255 bytes, multiple sequential packets are sent. First packet includes ATLV_LEN TLV with value of string total size. Subsequent packets contain an ATLV_OFF TLV with value offset. Final packet contains ATLV_EOF TLV (end of sequence).

Request ID is the same throughout the entire sequence. EOF TLV can also truncate a value earlier than specified in the ATLV_LEN TLV (in first packet).

Here is an illustration of the packet sequence that sends a string of length 515.

1)

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A2	01	10	01	09	12	34	01	04	73	74	72	67	12	02	02	03
Start	Length (272)		Protocol	Opcode	Request ID		Name TLV, Length 4, Value "strg"						Length TLV, Length 2, Value 515			

17	18	19	20	21	22	23	24	25	26	27	28	29	..	..	..	274
05	ff	61	62	63	64	65	66	67	68	69	70	71	..	..	..	CRC
Text TLV, Length 255 Value "abcd……."																

2)

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A2	01	10	01	09	12	34	01	04	73	74	72	67	11	02	00	ff
Start	Length (272)		Protocol	Opcode	Request ID		Name TLV, Length 4, Value "strg"						Offset TLV, Length 2, Value 255			

17	18	19	20	21	22	23	24	25	26	27	28	29	..	..	..	274
05	ff	61	62	63	64	65	66	67	68	69	70	71	..	..	..	CRC
Text TLV, Length 255 Value "......."																

3)

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
83	00	18	01	09	12	34	01	04	73	74	72	67	11	02	01	fe
Start	Length (24)		Protocol	Opcode	Request ID		Name TLV, Length 4, Value "strg"						Offset TLV, Length 2, Value 510			

17	18	19	20	21	22	23	24	25	26
05	05	61	62	63	64	65	14	0	CRC
Text TLV, Length 5, Value "abcde"							EOF TLV, Length 0		

7.2.4 Receive a property

(SPI – module to MCU) Message has a new value 0x123 for variable "go".

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
f1	00	0f	01	03	34	56	01	02	67	6f	02	04	00	00	01	23	CRC

Start	Length (15)	Protocol	Opcode	Request ID	Name TLV, Length 2, Value "go"	Integer TLV, Length 4, Value 0x00000123	

For time

- Start byte is 0xf1 (packet comes from module to MCU).
- Payload length is 15 (0x0f) (does not include start byte and length).
- Protocol is 1 (Ayla Data protocol)
- Opcode 3, "Receive TLV" (request ID 0x3456)
- Name and value TLVs (Name "go" has no zero padding. Value is a signed integer (type 2) of 4 bytes)

7.2.5 Receive a property with metadata

The mobile app sends a value of 1 (in LAN/WAN mode) to the MCU for the name “led0” with the data point metadata key-value pair (key = ”key1”, value = ”value1”.

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	17	18	19	20	21	22	23	24
82	00	1f	01	03	12	34	34	04	6b	65	79	31	05	06	76	61	6c	75	65	31
Start	Length (31)		Protocol	Opcode	Request ID		Datapoint Metadata TLV, length 4, key “key1”						UTF8 TLV, length 6, value “value1”							

25	26	27	28	28	29	30	31	32	33
01	04	6c	65	64	30	0f	01	01	CRC
Name TLV, length 4, value “led0”						Boolean TLV, length 1, value 1			

Payload begins with Protocol 1 (Ayla Data protocol), followed by opcode 3 “receive TLV” and arbitrary request ID 0x1234.

Next is the metadata TLV (always precedes name and value TLV). Datapoint metadata is sets of key-value pairs. Metadata TLV contains the key, followed by a UTF8 TLV (contains the value).

Metadata TLV has:

- type 52 (ATLV_DPMETA)
- length 4
- “key1” bytes

UTF8 TLV is:

- Type 5

- Length 6
- “value1” bytes”

MCU can receive up to four metadata key-value pairs. Maximum allowed length for key is 16 bytes, and for value is 32 bytes.

Name and Value TLVs follow. Name TLV is:

- Type 1 (name)
- Length 4
- Name “led0” (4 bytes)

Value TLV (in this case) holds the one-byte Boolean value of 1:

- Type 15
- Length 1

7.2.6 Explicit Acknowledgement of Datapoint (example)

If property is marked ack_enabled (datapoint originates from any source - ADS or LAN app), module passes ack_id to MCU.

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
f1	00	0f	01	03	34	56	01	02	67	6f	02	04	00	00	01	23
Start	Length (15)		Protocol	Opcode	Request ID		Name TLV, Length 2, Value "go"				Integer TLV, Length 4, Value 0x00000123					

17	18	19	20	21	22	23
1e	04	69	64	30	31	CRC
Ack_ID TLV, Length 4, Value – "id01" (from ADS/LAN app)						

After datapoint is executed, MCU sends ack (nested TLV format) to module of success or failure.

ATLV_ACK_ID contains:

- ack_id - associated with the datapoint
- ack_status in ATLV_ERR (0 = success, 1 = failure)
- ack_message - integer format

Node TLV value identifies datapoint source. With this, the module marks the datapoint acknowledged on the source.

Property name/value and Echo TLV initiates echoes to all destinations except datapoint source. Because datapoint must be executed before echoed, MCU initiates echoes.

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
f1	00	0f	01	09	34	56	01	02	67	6f	02	04	00	00	01	23
Start	Length (15)		Protocol	Opcode	Request ID		Name TLV, Length 2, Value "go"				Integer TLV, Length 4, Value 0x00000123					

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
1e	0f	1e	04	69	64	30	31	07	01	00	02	04	00	00	00	90
Ack_ID TLV, Length 15		UTF8 TLV, Length 4, Value – "id01" (used for ack_id)						Error TLV, Length 1, Value – 0 (used for ack_status)			Integer TLV, Length 4, Value – 0x90 (used for ack_message)					

34	35	36	37	38	39
18	00	17	01	01	CRC
Echo TLV, Length 0		Nodes TLV, Length 1, Value – 1 (used to indicate source of ack)			

7.2.7 Example NAK

This shows a NAK packet delivered to MCU. In addition to the request ID, there is an Error TLV, type 7, length of 1, and error code 1 (timeout).

Byte 0	1	2	3	4	5	6	7	8	9	10
f1	00	08	01	05	ab	cd	07	01	01	CRC
Start	Length (8)		Protocol	Opcode	Request ID		Error TLV, Length 1, Value 1 (timeout)			

Table 3. NAK Error Codes

Error Code	Name	Description
0x01	Timeout	Timeout in communications with server.
0x02	Length Error	TLV in request message indicates a length that extends past end of received packet.

Error Code	Name	Description
0x03	Unknown Type	TLV has an unsupported data type.
0x04	Unknown Variable	Configuration TLV requests unknown config/status variable.
0x05	Invalid TLV	TLV sequence is invalid (not a configuration value).
0x06	Invalid Opcode	Opcode is invalid.
0x07	Invalid Datapoint	Datapoint location is invalid.
0x08	Invalid Datapoint Offset	Datapoint offset too large.
0x09	Invalid Request	Request is invalid.
0x0a	Invalid Name	Property name contains illegal characters or too long.
0x0b	Connection Error	Connection to ADS (or some destinations) failed. If Nodes TLV is present, indicates destinations not reached.
0x0c	ADS Busy	Issued request cannot be handled while ADS_Busy status is set.
0x0d	Internal	Internal Error (Host Error)
0x0e	Checksum Error	Checksum Error
0x0f	Already Done	MCU OTA already done for that version
0x10	Boot Error	MCU did not boot to new image
0x11	Overflow Error	MCU took too long to receive data response from the request to module. Request must be resent.
0x12	Bad Value	Properly value not valid UTF-8.
0x13	Property length	Property value length exceeds limit.
0x14	Unexpected operation	Operation sequence not allowed
0x15	Datapoint Metadata Error	Datapoint metadata format invalid.
0x16	Acknowledgement ID Error	Acknowledgement information invalid.

7.3 File Datapoints

A file datapoint is longer than 255 bytes – and require multiple packets. On a network connection loss, packets can be re-sent or re-fetched. The data type is binary, and up to 4GB.

The protocol handles transfer of large property values (datapoints) differently than other ones. By design, MCU knows which properties support file datapoints.

7.3.1 Receive File Datapoint

These are the steps to receive a file datapoint:

1. A file property value sent to MCU from module contains a Location TLV (not the value TLV).

The Location TLV is a byte string, saved by MCU and used to request the file datapoint value.

2. MCU sends **File Datapoint Request** packet.

   Location TLV is included. Optionally, an Offset TLV can be included.
3. Module responds with one or more **File Datapoint Response** packets.
   - Contents include Location TLV followed by Binary Data TLV. Offset TLV is included.
   - If datapoint overall length is known, first packet contains a Length TLV.
   - Final data packet contains EOF TLV.
4. MCU receives data values in sequence.

   Do not issue new requests while transfer is in progress.

   Sequence gaps indicate an error.

   Other property values may be received during transfer.
5. On transfer complete, MCU should send **File Datapoint Fetched** command.

   Module informs ADS that file datapoint was fetched.
6. The MCU can abort a File download operation at any time through a **File Datapoint Stop** packet.

NOTE No other property updates are sent to MCU during an in-progress File upload. Module notifies MCU of pending property updates with AD_PROP_NOTIFY opcode (n case MCU wants to abort file transfer to receive the update).

7.3.2 Send File Datapoints

To send a file datapoint to the module and ADS, the MCU uses this sequence:

1. MCU sends **File Datapoint Create** command.

   Includes property name TLV. If required, datapoint metadata is included.
2. Module responds with **File Datapoint Response** packet.

   Contains new datapoint Location TLV, or (if unsuccessful) Error TLV.
3. MCU sends **File Datapoint Send** packet for value's initial bytes.

   Contains:
   - Location TLV
   - Offset TLV (for offset 0)
   - Length TLV (total length of value to be sent)
   - Binary data TLV.
4. MCU sends additional **File Datapoint Send** packets.

   Length TLV is not included.

   Final command must be EOF TLV.
5. At any time, MCU can send **File Datapoint Stop** packet to abort File upload operation.

NOTE	No other property updates are sent to MCU with in-progress File upload. Module notifies MCU of pending property updates with AD_PROP_NOTIFY opcode (if MCU wants to abort file transfer to receive the update).

8 Control Operations

Control operations are messages between the MCU and module that perform actions on the module's configuration and status.

The message format is similar to data operation message format:

- Byte 0: Protocol (0)
- Byte 1: Opcode
- Bytes 2 - 3: Request ID.
- Bytes 3 - n: TLVs

8.1 Control Opcodes

Table 4. Implemented Control Operations Opcodes

NOTE: TBD items are not implemented or not fully tested.

Opcode (hex)	Description	Sent by
1	Response to a previous Get Configuration Items opcode	Module
2	Get configuration items	MCU
3	Set configuration items	MCU
4	Save configuration	MCU
5	Reserved	
6	NAK – error occurred on previous request	MCU
7	Load startup configuration	MCU
8	Load factory configuration	MCU
9	OTA status	MCU
0x0a	OTA command	MCU
0x0c (TBD)	Log operation	MCU
0x0d	MCU OTA report or start	Module
0x0e	MCU OTA Load	Module
0x0f	MCU OTA status	MCU
0x10	MCU OTA Boot	Module
0x11	Configuration update	Module
0x12	Wi-Fi Join network.	MCU
0x13	Wi-Fi Leave and forget network	MCU

The control operations use protocol number 0, and the following opcodes:

Opcode 1: Response

Response to a previous Get Configuration Items opcode (module to MCU).

Alternating TLVs - Conf TLVs (configuration tokens) and value TLVs.

Opcode 2: Get Configuration Items

Request to module for configuration items.

Reply with Conf TLVs of configuration tokens. See section 0.

Opcode 3: Set configuration items

Request to module to set one or more configuration items.

TLVs are alternating Conf TLVs (configuration tokens) and value TLVs.

See the section 0 for the available configuration items.

Opcode 4: Save configuration

Save running configuration to startup configuration.

Opcode 5: Reserved

Reserved for future use.

Opcode 6: NAK

Error occurred on a previous request.

Opcode 7: Load Startup configuration

Instructs module to restart with startup configuration.

Opcode 8: Load Factory configuration

Instruction to module - copy factory configuration to startup configuration and restart.

Opcode 9: OTA Status

Informs MCU that module has available OTA update.

- If TLVs, more OTA info is provided.
- If no TLVs, simple notification that module OTA update is available.

Opcode 0x0a, OTA Command

With no TLVs, informs module to install new module OTA update.

Module de-asserts READY (and be unavailable for some time). After a time (at least 30 seconds), the module is reset.

For battery-operated devices, voltage must be appropriate for flash operations.

(TBD) Opcode 0x0c: Log operation

Instructs module to perform a logging operation. Operations are: append MCU message, snapshot, clear snapshots, send log or snapshot to service.

Opcode 0x0d: MCU OTA Report or Start

This message (module to MCU) indicates an available firmware update.

Message from MCU to module can request firmware download start.

Message from module to MCU contains two TLVs:

- type ATLV_UTF8 contains firmware version string
- type ATLV_LEN contains firmware size (bytes).

Opcode 0x0e: MCU OTA Load

This message (module to MCU) contains a chunk of the firmware.

The module keeps sending these messages until firmware download is complete.

The message contains two TLVs. First TLV is type ATLV_OFF and contains data chuck offset. Second TLV is type ATLV_BIN and contains the actual data.

Opcode 0x0f: MCU OTA Status

This message (MCU to module) reports firmware download status.

If any problem, MCU must send non-zero status code in TLV of type ATLV_ERR. These are the status codes:

- 0x02 - AERR_LEN_ERR (TLV extends past end of received buffer)
- 0x0d - AERR_INTERNAL (Internal error)
- 0x0e - AERR_CHECKSUM (Checksum mismatch)
- 0x0f - AERR_ALREADY (Already running new version)
- 0x10 - AERR_BOOT (MCU did not boot to new image)

Opcode 0x10: MCU OTA Boot

This message (module to MCU) indicates which image to boot.

The version to boot is specified with an optional TLV of type ATLV_UTF8. If no version specified, MCU boots to the most recently downloaded image.

Opcode 0x11: Configuration Update

This message (module to MCU) notifies of configuration changes that the MCU might be interested in.

For example, if the MCU supports subscribes to time information (indicated to module with the feature mask), the module notifies MCU every change to configured time, timezone information, or daylight savings information.

TLVs are alternating Conf TLVs that contain configuration tokens and value TLVs.

Opcode 0x12: Wi-Fi Join

Configure and try to join a Wi-Fi network.

This message (MCU to module) contains three TLVs that describe the network. These TLVs must come in sequence.

First TLV is the network's SSID. SSID must have TLV type of ATLV_UTF8, but does not need to be UTF-8 encoded.

Second TLV must be type ATLV_INT and contain the security level. This is a configuration token, so values can be: none, WEP, WPA2 Personal.

Third TLV must be type ATLV_BIN and contains network password.

Opcode 0x13: Wi-Fi Delete

Forget and leave a Wi-Fi network.

This message (from MCU to module) identifies the network – SSID is included in a TLV.

8.2 Configuration

The module uses configuration variables with a hierarchical naming structure.

The write configuration variable name is a set of names separated by slashes (similar to Unix file names). For example, SSID for the first Wi-Fi prototype is `/wifi/profile/0/ssid`.

8.2.1 Configuration Tokens

To save program space and time, each config token is encoded with one number. The above example is actually coded as 0x4, 0x25, 0, 0x26. The third token is the index of profile, 0.

The conf_tokens.h file has token numerical assignments in lines to invoke a CONF_TOKEN macro. That macro can be user-defined so that the token number and name is used as the programmer prefers. For example, one line is:

CONF_TOKEN(4, wifi)

indicates the coding for the wifi token is 4.

File conf_token.h defines an *enum conf_token* with values for each token. For example, CT_wifi is set to 4.

Table 5. Token codes

Code (hex)	Token Name		Code (hex)	Token Name
1	enable		46	ca
2	ready		47	GIF
3	sys		48	OEM
4	wifi		49	log
5	server		4a	mod

Code (hex)	Token Name	Code (hex)	Token Name
6	client	4b	mask
7	ssl	4c	chan
8	status	4d	error
9	start	4e	link
a	complete	4f	reg
b	ip	50	default
c	n	51	min
d	time	52	standby
e	power	53	mode
f	user	54	dhcp
10	version	55	gw
11	file	56	snapshot
12	name	57	hist
13	type	58	source
14	model	59	WPS
15	serial	5a	listen
16	mfg_serial	5b	interval
17	hostname	5c	auto
18	timezone	5d	current
19	timezone_valid	5e	awake_time
1a	mfg_mode	5f	standby_powered
1b	dev_id	60	unconf_powered
1c	setup_mode	61	metric
1d	mfg_model	62	http
1e	sim	63	tcp
1f	reset	64	count
20	region	65	locale
21	acc	66	lan
22	char	67	setup_ios_app
25	profile	68	notify
26	ssid	69	gpio
27	security	6a	intr
28	none	6b	data

Code (hex)	Token Name	Code (hex)	Token Name
29	WEP	6c	clock
2a	WPA	6d	mfi
2b	WPA2_Personal	6e	hidden
2c	key	70	value
2d	pri	71	prop
2e	scan	72	port
2f	time_limit	73	sched
30	save_on_ap_connect	74	dst_active
31	save_on_server_connect	75	dst_change
32	max_perf	76	dst_valid
33	en_bind	77	dns
34	ap_mode	78	hw
35	rssi	79	rtc_src
36	bssid	7a	host
37	bars	7b	uart
38	poll_interval	7c	spi
39	addr	7d	speed
3a	mac_addr	7e	eth
3b	ant		
43	connected		
44	cert		
45	private_key		

8.3 Configuration Tree

The configuration variables are described here in one section per subtree in the hierarchy.

Table 6. Top-level Configuration sub-trees

Sub-tree	Description
client	Connection to the Ayla Device Service (ADS)
eth	Ethernet device
gpio	module I/O configuration
hw	module hardware features

Sub-tree	Description
ip	IP networking for Wi-Fi
log	device logging
metric	module statistics
oem	OEM name and model
power	power management
server	module internal web server
sim	Internal simulation / testing (see testing documentation)
sys	overall system configuration
wifi	Wi-Fi settings

8.3.1 Client Configuration Settings

Table 7. Client Configuration Settings

Variable	Type	Access	Meaning
/client/enable	Boolean	r/w	Enable client subsystem
/client/poll_interval	Integer	r	time (seconds) between polls if ANS not used
/client/hostname	UTF-8	r	hostname of ADS server
/client/reg	UTF-8	r	registration token
/client/reg/ready	Boolean	r	User registered status
/client/reg/start	Boolean	w	request registration token
/client/reg/interval	Boolean	w	Open registration window
/client/reg/setup_mode	UTF-8	W	Set the setup-token passed by the mobile App for registration
/client/ssl/enable	Boolean	r	Use SSL
/client/server/region	UTF-8	r/w	Geographic region
/client/test	Boolean	r/w	Test connection to ADS

8.3.2 Client Module Registration

To register a module with ADS, use a mobile application or the ADS web interface. As an alternative for MCUs with a user interface, this can be done to register the module (if already connected to the Internet and to ADS):

1. On a button press or touch-screen or other user action, indicate to the ADS client that a new registration key should be generated (set `/client/reg/start` to 1).
2. On completion, a registration key is assigned to `/client/reg`. (Can be displayed to user or sent to ADS web server to register the device.

3. Set `/client/reg/interval` to 1 requests ADS to open a 2-minute device registration window. (Similar to WPS push-button registration.)
4. Get Boolean value of `/client/reg/ready`. (1 = user registered, 0 = no registered user).

   ADS must be accessible (see bit 0 in the available destinations mask as reported to the MCU via the Data Connectivity Status operation).
5. Confirm registration status and the regkey is not available unless ADS is accessible.

When /client/test is set to 1, ADS is informed that connections are for OEM testing (does not count as end-user first connection of device. (Setting does not persist - lasts only until the next reboot.)

8.3.3 Ethernet Configuration Settings

Table 8. IP Configuration Settings

Variable	Type	Access	Meaning
/eth/enable	Boolean	r/w	Enable Ethernet Interface (Not writable until the module firmware version 1.7.)
/eth/mac_addr	Binary	r/w	MAC address for Ethernet device (Not writable until the module firmware version 1.7.)

Not writable until the module firmware version 1.7.

8.3.4 IP Configuration Settings

Table 9. IP Configuration Settings

Variable	Type	Access	Meaning
/ip/n/<n>/addr	Integer	r/w	IP address
/ip/n/<n>/mask	Integer	r/w	Netmask
/ip/dhcp/enable	Boolean	r/w	Use DHCP
/ip/dns/n/<n>	Integer	r/w	DNS server addr
/ip/gw	Integer	r/w	default gateway

DHCP must be disabled to write IP address, netmask, DNS addresses, and default gateway IP address.

8.3.5 Log Configuration Settings

Table 10. Log Configuration Settings

Variable	Type	Access	Meaning
/log/mod/<n>/mask	integer	r/w	Mask of enabled log severities

Variable	Type	Access	Meaning
/log/snapshot/<n>/time	integer	r	Time (UTC) of snapshot <n> 1 – Default, 2 – Client, 3 – Conf, 4 – Dnss, 5 – Netsim, 6 – Notify, 7 – Server, 8 – Wifi, 9 – SSL, 10 - Log-client, 12 – Sched

8.3.6 OEM Configuration Settings

Table 11. OEM Configuration Settings

Variable	Type	Access	Meaning
/oem/oem	UTF-8	r/w	Device manufacturer name
/oem/model	UTF-8	r/w	Device model name
/oem/key	File/UTF-8	w	OEM validation string (see discussion)

Settings are accessed only in setup mode to allow product manufacturer to add OEM name and model. This is used by ADS.

The key verifies to ADS that the oem and model belong to the specified OEM. The key is used to encrypt or sign a string that includes the oem and model. The key is not stored by the module and if read returns undefined results.

Whenever the oem or model are set, the key must be re-written with a base-64 UTF-8 string. Ayla recommends that the key not be stored in the host MCU as shipped to customers. After setup, erase the key.

8.3.7 Power Configuration Settings and Status

Table 12. Power Configuration Settings

Variable	Type	Access	Meaning
/power/mode	Integer (token)	r/w	Power management mode - values are: CT_default CT_min CT_max_perf CT_standby
/power/current	Integer (token)	r/w	Active power management mode
/power/awake_time	Integer	r/w	Time in seconds that the module should stay awake after activity.
/power/standby_powered	Integer	r/w	Time in seconds that the module waits before standby after boot.
/power/unconf_powered	Integer	r/w	Time (seconds) module waits before standby after boot when no Wi-Fi profiles are configured.

See *Ayla OEM Installation Guide for Black Box* (AY006DSU0), Ayla Power Management section.

8.3.8 Module HTTP Server Configuration Settings

Table 13. OEM Configuration Settings

Variable	Type	Access	Meaning
/server/prop/time_limit	Integer	r/w	HTTP Server access to properties enable/disable
/server/security/enable	Boolean	r/w	HTTP clear-text access in AP mode

`prop/time_limit`

This variable enables a local HTTP client to access MCU properties (for a limited time) via the Wi-Fi module's HTTP server. To take effect, Wi-Fi module must be in AP mode and not registered with a user. (Generally, this is used for device testing during manufacturing.)

`time-limit`

This value is in seconds (<= 600). Access is automatically disabled after the time limit. If time_limit = 0, access is disabled.

`/server/security/enable`

This variable enhances AP mode server security on module. (Supported version 2.6+)

- 0 = disables feature - module allows wifi setup through clear-text server requests and wifi javascript page.
- 1 enables feature - clear-text access is blocked, wifi javacript page not accessible.

Wifi setup can be done only via an encrypted session through the mobile app.

8.3.9 System Configuration and Status Settings

Table 14. System Configuration and Status Settings

Variable	Type	Access	Meaning
/hw/rtc_src	Integer	r/w	RTC clock source selection; - 512 for LSI (low-speed internal oscillator) - 256 for LSE (external oscillator). Only exists for certain module types. Writable in setup mode only.
/sys/dev_id	UTF-8	r	Ayla DSN
/sys/mac_addr	Binary	r	MAC address
/sys/mfg_mode	Boolean	r	Manufacturing mode
/sys/mfg_mode/complete	Integer	r	Test completion time
/sys/mfg_model	UTF-8	r	Module mfg model name
/sys/mfg_serial	UTF-8	r	Model mfg serial number
/sys/model	UTF-8	r	Model name
/sys/serial	UTF-8	r	Serial number

Variable	Type	Access	Meaning
/sys/setup_mode	Boolean	r/w	Setup mode (write value must be 0).
/sys/time	Integer	r/w	UTC time (in seconds since Jan 1st, 1970). If written by MCU, set with 0x1250 source.
/sys/time/pri	Integer	r/w	Same as /sys/time but with 0x1280 source.
/sys/timezone	Integer	r/w	Minutes west of UTC
/sys/timezone_valid	Boolean	r/w	If the timezone settings are valid
/sys/dst_active	Boolean	r/w	If /sys/dst_change specifies the end of DST
/sys/dst_change	Integer	r/w	UTC time when daylight savings time ends or begins
/sys/dst_valid	Boolean	r/w	If dst settings are valid. See Table 14.
/sys/time/source	Integer	r	How time was set
/sys/version	UTF-8	R	Module software version

Table 15. Different values of sys/time/source translation

Value	Meaning
0x0000	Clock has never been set
0x1130	Clock set to an arbitrary default
0x1140	Clock set by internal web server
0x1250	Clock set by MCU (Lower Priority to ADS)
0x1260	Clock set using ADS
0x1270	Clock set using NTP
0x1280	Clock set by MCU (Higher Priority to ADS)

NOTES:

The MCU can set module time with `sys/time` or `sys/time/pri`. The only difference is the time source:

- `sys/time` setting is subordinate to Ayla Cloud and can be overwritten by the Cloud.
- `/sys/time/pri` setting cannot be overwritten by other sources.

8.3.10 Wi-Fi Configuration Settings

Table 16. Wi-Fi Configuration Settings

Variable	Type	Access	Meaning
/wifi/ant	Integer	r/w	Antenna selection
/wifi/enable	Boolean	r/w	Enable Wi-Fi
/wifi/listen/interval	Integer	r/w	Listen interval in counts of beacons.

Variable	Type	Access	Meaning
/wifi/mac_addr	Binary	r/w	MAC address
/wifi/power	Integer	r/w	Maximum TX power (db)
/wifi/profile/start	Integer	w	Profile to activate
/wifi/profile/<n>/ssid	UTF-8 (see note)	r/w	SSID
/wifi/profile/<n>/security	Integer (token)	r/w	Type of security to use
/wifi/profile/<n>/key	Binary	r/w	Wi-Fi key
/wifi/profile/<n>/ enable	Boolean	r/w	Enable the profile
/wifi/region	Integer	r/w	Regulatory region where Wi-Fi module operates
/wifi/scan/ready	Boolean	r	Scan results are ready
/wifi/scan/ save_on_ap_connect	Boolean	r/w	Save profile when it connects to an AP
/wifi/scan/ save_on_server_connect	Boolean	r/w	Save profile when it connects to ADS
/wifi/scan/start	Boolean	w	Start Wi-Fi scan
/wifi/scan/n	Integer	r	Take a snapshot of current scan results
/wifi/scan/n/<n>/ssid	UTF-8 (see note)	r	Scan result SSID
/wifi/scan/n/<n>/bssid	Binary	r	Scan result BSSID
/wifi/scan/n/<n>/rssi	Integer	r	Signal (dbm)
/wifi/scan/n/<n>/bars	Integer	r	Signal strength in 0 to 5 "bars".
/wifi/scan/n/<n>/security	Integer	r	Security type token
/wifi/scan/n/<n>/chan	Integer	r	Scan result channel
/wifi/setup_ios_app	UTF-8	r/w	Custom URI for iOS setup app. Writable in setup mode only.
/wifi/setup_mode/enable	Integer	r/w	Start or stop setup mode. - set bit 0 (value 1) to start Airkiss setup - set bit 0 (value 0) to stop Wi-Fi must be in AP mode.
/wifi/setup_mode/key	Binary	w	Encryption key for setup mode. Airkiss uses an AES 128-bit key which must be exactly 16 bytes.
/wifi/status/profile	Integer	r	Profile of connected AP

Variable	Type	Access	Meaning
/wifi/status/rssi	Integer	r	Signal strength in dbm
/wifi/status/bssid	Binary	r	BSSID of connected AP
/wifi/WPS	Boolean	r/w	WPS virtual button press

Notes on Wi-Fi Configuration Settings table above:

`/wifi/enable` is the master Wi-Fi enable.

- If = 1, the Wi-Fi monitor tries to maintain a connection (or be in AP mode).
- If = 0, Wi-Fi chip is powered off.

Antenna selection varies by model. For 43362-based modules, antenna inputs can be 0 or 1 - or 3 to automatically use the best antenna. All other values are reserved.

If `/wifi/mac_addr` is non-zero, it can be an alternative Wi-Fi mac address (primarily for testing). Otherwise, if zero, `/sys/mac_addr` is used.

There are 11 profiles.

- Profiles 0-9 are for known networks.
- In AP mode, module uses profile 10.

When `/wifi/enable` is set , module tries to connect to configured profiles. It picks an appropriate profile. To override selection, set `/wifi/profile/start` to a value. Module drops the current connection and attempt to profile with that index.

To start module in AP mode, set `/wifi/profile/start` to AP profile's index. This setting persists until module is configured with a new profile or it does a Wi-Fi scan.

There are up to 20 scan results, numbered 0 thru 19. Some modules store fewer. When no scan is in progress, results can be retrieved.

Before scan results are fetched, MCU must read variable `/wifi/scan/n`. This does a 'snapshot' of current results. The module keeps serving scan information from this snapshot. A new scan removes old snapshot data. Another read `/wifi/scan/n` is required.

SSID in profiles and scan results are passed as a UTF-8 TLV. Any sequence up to 32 bytes is valid, even if some are not valid UTF-8 characters.

Module must be in AP mode to start Wi-Fi Protected Setup. WPS Push Button Configuration is supported.

- MCU can press virtual Push Button by writing '1' to /wifi/WPS.
- Button stays pressed until module finds a suitable AP advertising WPS (or after 120 seconds).
- When WPS is finished, button is released.

If module sees a suitable AP, it tries to join that network. The join progress is in Wi-Fi Connection History, if needed.

During manufacturing or setup modes, set `/wifi/region` to restrict Wi-Fi behavior (of the attached module) to region requirements. These are valid `/wifi/region` values.

0 - Default region for the module

1 - United States

2 - China

3 - EU

4 - Japan

5 - Canada

6 - Australia

8.2.11 Wi-Fi Connection History

Table 17. Wi-Fi Connection History

Variable	Type	Access	Meaning
/wifi/hist/n/<n>/ssid	Binary	r	First and last byte of SSID. Non-zero if history entry exists.
/wifi/hist/n/<n>/bssid	Binary	r	BSSID
/wifi/hist/n/<n>/dns/n/<n>	Integer	r	DNS server IP address
/wifi/hist/n/<n>/error	Integer	r	error code
/wifi/hist/n/<n>/time	Integer	r	UTC of attempt
/wifi/hist/n/<n>/addr	Integer	r	IP address
/wifi/hist/n/<n>/mask	Integer	r	netmask assigned
/wifi/hist/n/<n>/gw	Integer	r	gateway assigned

Connection history saves the last three connection attempts. History index 0 is the latest entry. Connection error codes are:

0 - No error.

1 - Resource problem, out of memory or buffers, perhaps temporary.

2 - Connection timed out.

3 - Invalid key.

4 - SSID not found.

5 - Not authenticated via 802.11 or failed to associate with the AP.

6 - Incorrect key.

7 - Failed to get IP address from DHCP.

8 - Failed to get default gateway from DHCP.

9 - Failed to get DNS server from DHCP.

10 - Disconnected by AP.

11 - Signal lost from AP (beacon miss).

12 - Device service host lookup failed.

13 - Device service GET was redirected.

14 - Device service connection timed out.

15 - No empty Wi-Fi profile slots.

16 - The security method used by the AP is not supported.

17 - The network type (e.g. ad-hoc) is not supported.

18 - The server responded in an incompatible way. The AP may be a Wi-Fi hotspot.

19 - Module failed to authenticate to device service.

20 - Connection attempt currently in progress. Check later for connection status.

8.3 Control Operations – Log

A log control operation (control opcode 0x0c) message includes an Integer TLV with the function number followed by optional argument TLVs.

Table 18. Log Control Functions

Function	Name	Meaning
1	Append	Append message to the log. MCU appends messages to module's log, and (f enabled) writes to the module's serial port. Log level is specified by the message's first character. Message is a UTF8 TLV and contains only printable ASCII characters 0x20 thru 0x7e. Message source is set to "mcu".
2	Save Snapshot	Save the current log. Module saves current log messages to the flash ROM for further diagnosis. Up to eight log snapshots may be saved. If save area is full, this command receives a NAK.
3	Clear Snapshots	Clear all saved snapshots. Module erases the saved snapshots.
4	Send	Send current log to ADS. Module sends current log to ADS.
5	Send Snapshot	Send specified snapshot to ADS. Module sends a snapshot to ADS. The snapshot number is specified using an Integer TLV. If no snapshot number is specified, the most recently saved one is sent.

9 Hardware Interface

9.1 General Signals

The module provides several signals used by MCUs with SPI or asynchronous serial protocol. MCUs with either interface can use these signals:

- RESET_N

  This is both a module input and output. Should be driven low by MCU to reset the module.

  The module pulls it up.
- READY_N

  This open-drain signal goes low after the module is initialized and ready for SPI communications.

  If module is reset, signal floats and should be pulled up by the MCU.
- LINK_N

  Output goes low when the module connects to the ADS.

  Goes high when the module cannot communicate with ADS.
- INTR_N

  This pin is an open-drain, active-low interrupt output from the module. It is active when module has SPI message pending for MCU.

  It is possible to poll status without this line. If used, pin should be pulled up to 3.3v (not 5v) on the MCU.
- WKUP

  When module's current power-management level = standby, this signal can be driven low (puts module into standby), and then brought high (wakes up module from low-power standby state). See the document for the particular module being used.

9.2 SPI Signals

For the SPI interface, signals are:

- SPI Clock
- SPI MOSI
- SPI MISO
- SPI SSN

Details of SPI signals are specific to a particular module type. Information on a typical configuration are provided below (variations may exist but are rare).

- MCU provides the SPI clock.

  Currently the interface is tested at 3.2 MHz. Higher rates up to 16 MHz should be possible.

- SPI interface is used in mode 1 (CPOL = 0, CPHA = 1).

  The clock polarity is set so that it is normally low when idle, and the phase is set such that the receiver latches the data on the falling edge of the clock.
- Transfers are sent with MSB first.
- The hardware slave select line (SSN) is honored and should be used.

  This helps synchronize the data. If not implemented by the host MCU, it should be pulled down.
- On some modules, MISO output from the module is configured to be an open-drain output.

  This permits a multi-slave configuration (MCU can access other SPI peripherals on the same bus). MCU should supply a pull-up resistor for MISO.