INDUCTOSENSE UAS UWM HTTPS API SPECIFICATION

Project Code: INDU0275
Revision: 08
Author: L O'Donnell
Date: 28/04/2025

Contents

History

Revision	Date	Author	Comment
01	15/05/2024	L O'Donnell	Initial issue
02	28/05/2024	L O'Donnell	Added Sleep and Wi-Fi control commands Added Get Information command Changed responseCode so 0 is the successful state and > 0 is an error

			condition
03	21/06/2024	L O'Donnell	Added LR enable to Wi-Fi configuration Added 403 Forbidden response and security levels Added authorization requests
04	06/09/2024	L O'Donnell	Updated /live request to include optional query parameters for a truncated response Removed getting and setting server certificates
05	19/09/2024	L O'Donnell	Correction to table of contents
06	02/10/2024	L O'Donnell	Updated Get Live Reading to correct schema version. Added Get Live XY Trace. Updated Get Measurement to include an additional query parameter to restrict to just header data. Added Get Scan Error
07	08/01/2025	L O'Donnell	Updated Start Scan to have a responseCode of 2 where a scan would fail to start
08	28/01/2025	L O'Donnell	Update 2.3.5 to include Invalid sensor configuration Added deprecation of 2.3.9 Added 2.3.20

References

Reference	Document	Author, link, attachment
REF 1.	Authentication methods proposal for Wand v3 – rev 4	IND
REF 2.		
REF 3.		

Glossary of terms

WAND	Wireless And Non-Destructive, Inductosense branding for the core inductive technology
UAV	Unmanned Aerial Vehicle
UAS	Unmanned Aerial System
UWM	UAV WAND Module
AP	Access Point

1 Introduction

In order to control and view/review scans from the UAS UWM module an interface over the long-range Wi-Fi link an interface is required.

Inductosense are responsible for developing the software to communicate with the UAS UWM module, this document is intended to describe the interface for that communication method such that Inductosense are able to develop the aforementioned software.

It is expected that the interface will be in the form of a restful HTTP interface over HTTPS. However, the transport layer of this interface may be subject to change if it is found that this is not suitable for meeting the requirements.

This document forms the basis of that interface.

This is a live document and is subject to change.

2 HTTPS API Interface

2.1 Brief

The HTTPS API interface shall be limited in scope to starting and stopping scans and acquiring readings from the UWM.

A transparent serial request shall also be provided for convenience, this shall transparently send serial data to the UWM and respond with the serial response, or a timeout.

It is expected that the body data and response of a request shall be in JavaScript Object Notation format (JSON).

The contents for each request may differ and shall be explained within each request.

2.2 Authorization

It is expected that HTTP Basic Authorization would be used to authorize all API requests, it is expected this should be sufficient as the data exchange is already encrypted over the HTTPS transport layer.

It is expected that the login credentials shall come from the *Users* list stored on the UWM, i.e.; the username shall be the username of the user in the users list and the password shall be the string representation of the PIN code in the users list.

The Users list shall be retrieved once the Wi-Fi component starts. It is not expected that the Users list shall be updated in-flight, however the users list may be updated via issuing a *Restart Wi-Fi* command.

Additional authorization is required to effect certain commands, the principles behind this are described in the document *Authentication methods proposal for Wand v3 – rev 4*.

However, the methodology for authentication differs slightly in that the serial traffic between the ESP32 and the HDC main board shall not be encrypted. Encrypting the serial traffic in this fashion adds a significant time penalty, compounded with the time spent processing HTTPS requests means that extracting measurement data would not be too slow to meet the requirements.

Instead, the existing principles used to switch between level 1 and level 2 described in *Wand v3 – rev4* shall be used to raise the security level of the UWM from level 0 to level 1. There are no level 2 HTTPS API requests at the time of writing.

Serial commands to issue device keys to the ESP32 for the purposes of handling the challenge / response mechanism are described in *INDU0276 UAS Serial API Specification Rev 2*.

2.3 Requests

2.3.1 Start Scan

The command requests that a continuous scan is started on the UWM. A scan shall continue until the Stop Scan command has been issued.
A re-issue of a Start Scan command whilst a scan is in progress shall have no effect, and the response shall indicate this.

Url:	/scan
Request Type:	POST
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	application/json
Body Example:	{“scan”: 1}
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0, “message”: “OK”}
Response Example 2:	{“responseCode”: 1, “message”: “Scan already in progress”}
Response Example 3:	{“responseCode”: 2, “message”: “Failed to start scan”}

The body data of the request shall consist of a *scan* type, using the following enumerated values.

Enumeration	**Function**
0	Stop Scan
1	Start Scan
2	Perform One-shot Scan

A successful response, shall consist of a *responseCode* and a *message* where applicable.
A *responseCode* of 0 shall indicate that the request was performed successfully, a *responseCode* greater than 0 shall indicate that the request was not performed successfully, and *message* shall be populated with a reason to indicate why the request could not be performed.

It is expected that this command shall return a *responseCode* of 0 to indicate that the Scan has started and is in progress, which could then be verified by the *Get Scan Status* request.

It is expected that this command shall return a *responseCode* of 1 to indicate that the Scan could not be started due to a Scan already being in progress, as such *message* would be set to Scan already in progress.

The command shall return a responseCode of 2 to indicate the scan could not be started due to either an invalid subscription or another fault. The **Get Scan Error** command should be used to clarify the reason for the failure in the event that the subscription is valid.

If no body data is sent with this request, or the body data is malformed and could not be parsed, the request will return a 400 Bad Request.

This command is asynchronous, the Get Scan Status and Get Scan Error commands should be used to verify that a scan has started.

2.3.2 Stop Scan

The command requests that an in progress continuous scan is stopped on the UWM.
If a scan is not in progress, this command has no effect and the response shall indicate this.

Url:	/scan
Request Type:	POST
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	application/json
Body Example:	{“scan”: 0}
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0, “message”: “OK”}
Response Example 2:	{“responseCode”: 1, “message”: “No scan in progress”}

This command shall use the same body data parameters as detailed in the *Start Scan* request.

A successful response to this command shall use the same body data as detailed in the *Start Scan* request.

It is expected that this command shall return a *responseCode* of 0 to indicate that the currently running Scan has been stopped, it should then be verified by the *Get Scan Status* request.

It is expected that this command shall return a *responseCode* of 1 to indicate that the Scan could not be stopped. The expected error condition for this is that a Scan was not already in progress, as such *message* would be set to No scan in progress.

If no body data is sent with this request, or the body data is malformed and could not be parsed, the request will return a 400 Bad Request.

2.3.3 One-shot Scan

The command issues a "One-shot" scan command to the UWM. The functionality of this command shall be equivalent to running a single scan on the UWM.

Url:	/scan
Request Type:	POST
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	application/json
Body Example:	{"scan": 2}
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/json
Response Example 1:	{"responseCode": 0, "message": "OK"}
Response Example 2:	{"responseCode": 1, "message": "Scan already in progress"}

This command shall use the same body data parameters as detailed in the *Start Scan* request.

The response to this request shall use the same body data as detailed in the *Start Scan* request.

It is expected that this request shall return a *responseCode* of 1 to indicate that the One-shot scan has started.

It is expected that this request shall return a *responseCode* of 0 to indicate that a One-shot scan could not be started. The expected error condition for this is that a live scan is already in progress, as such, *message* shall be set to Scan already in progress.

If no body data is sent with this request, or the body data is malformed and could not be parsed, the request will return a 400 Bad Request.

2.3.4 Get Scan Status

The command is used to retrieve the current scanning status on the UWM.

Url:	/scan
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	None
Body Example:	
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/json
Response Example 1:	{"scan": 0}

Response Example 2:	{“scan”: 1}

A successful response shall indicate whether a scan is currently in progress.

Enumeration	Meaning
0	Scan not in progress
1	Scan in progress

No body data shall be sent with this request.

2.3.5 Get Scan Error

This command shall be used to return the error condition of the last requested scan.

Url:	/scanerror
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	None
Body Example:	
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0, “message”: “No error”}
Response Example 2:	{“responseCode”: 1, “message”: “Subscription lapsed”}
Response Example 3:	{“responseCode”: 2, “message”: “Too many readings”}
Response Example 4:	{“responseCode”: 3, “message”: “Cartridge not detected”}
Response Example 5:	{“responseCode”: 4, “message”: “Invalid/no cartridge detected”}
Response Example 6:	{“responseCode”: 5, “message”: “Invalid sensor configuration”}
Response Example 7:	{“responseCode”: 6, “message”: “Failed to read error codes”}

2.3.6 Get Live Reading

It is expected that this command is to be used whilst a scan is in progress to provide live measurement data, equivalent to the live mode available on HDC.

The command retrieves the most recent live reading data from UWM memory if a scan is currently in progress.

If a scan is not in progress, the command shall return an error.

Url:	/live
Request Type:	GET
Parameters:	See Optional Query Parameters
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	none
Body Example:	
Response Codes:	200, 400, 401, 403, 500

Optional Query Parameters:	startIndex, numPoints
Response Type:	application/octet-stream
Response Example 1:	…
Response Example 2:	

It is expected that a successful response shall return binary data with a structure containing the necessary data for a live reading.

Optional query parameters may be used to alter the number of data points returned from the measurement; the query parameters act as a way to retrieve a subset of the measurement from a specific starting point. If the query parameters are not populated, the default number of data points (10000) will be used.
The query parameter startIndex is 0-based and allowed values range between 0 and 9999.
The query parameter numPoints is the number of points to be returned starting from startIndex.
Both startIndex and numPoints must be present and valid to be functional. If one is present and the other is missing, the command shall return an error.
If the query parameters are out of bounds the command shall return an error.
If numPoints is set to 0, only the header shall be returned.

It is expected that this would be a subset of the data used from schema 7 of a measurement.

2.3.6.1 Expected Live Response Data Structure

Name	**Type**
StructureVersion	Unsigned Short
HeaderLength	Unsigned Short
MeasurementYear	Unsigned Short
MeasurementMonth	Byte
MeasurementDay	Byte
MeasurementHours	Unsigned Short
MeasurementMins	Byte
MeasurementSecs	Byte
SensorId	Byte(12)
SampleInterval	Float32
MaterialIndex	Unsigned Short
CartridgeIndex	Unsigned Short
Velocity	Float32
SNR	Float32
SystemDelayTime	Float32
Temperature	Float32
Thickness	Float32
Data	Float32(10000)

2.3.7 Get Live XY Trace

This command shall be used during a scan to return the Live Trace XY co-ordinates used to draw the A-Scan polynomial curve.

It is expected this command can be used in bandwidth restricted environments where it may not be feasible to return measurement data (such as Long-Range mode).

Url:	/trace
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	none
Body Example:	
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/octet-stream
Response Example 1:	…
Response Example 2:	

It is expected that a successful response shall return binary data with a structure containing the header meta-data of a live reading in addition to the XY trace co-ordinates.

2.3.7.1 Get Live XY Trace Data

Name	**Type**
StructureVersion	Unsigned Short
HeaderLength	Unsigned Short
MeasurementYear	Unsigned Short
MeasurementMonth	Byte
MeasurementDay	Byte
MeasurementHours	Unsigned Short
MeasurementMins	Byte
MeasurementSecs	Byte
SensorId	Byte(12)
SampleInterval	Float32
MaterialIndex	Unsigned Short
CartridgeIndex	Unsigned Short
Velocity	Float32
SNR	Float32
SystemDelayTime	Float32
Temperature	Float32
Thickness	Float32
LiveTraceX	UInt16(100)
LiveTraceY	UInt16(100)

2.3.8 Get Measurement

It is expected that this command is to be used when a scan has finished to provide the measurement result, equivalent to the data shown on HDC after a user releases the scan button.

The command retrieves the last measurement data from UWM memory if a scan is not currently in progress.

If a scan is in progress, the command shall return an error.

Url:	/measurement
Request Type:	GET
Parameters:	See Optional Query Parameters
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	none
Body Example:	
Response Codes:	200, 400, 401, 403, 500
Optional Query Parameters:	header
Response Type:	application/octet-stream
Response Example 1:	…
Response Example 2:	

An optional query parameter of “header=1” may be used to return only the header data matching schema 7 of a standard measurement. If the query parameters is malformed, it shall be treated as if it is not present, and return full measurement data.

A successful response shall return binary data matching schema 7 of a standard measurement.

2.3.8.1 Schema 7 Rev 1

Name	**Type**
SchemaVersion	Unsigned Short
HeaderLength	Unsigned Short
MeasurementYear	Unsigned Short
MeasurementMonth	Byte
MeasurementDay	Byte
MeasurementHour	Unsigned Short
MeasurementMins	Byte
MeasurementSecs	Byte
SensorId	Byte(12)
SampleInterval	Float32
MaterialIndex	Unsigned Short
CartridgeIndex	Unsigned Short
Velocity	Float32
CartridgeSerial	Unsigned Int32
SystemDelayTime	Float32
Temperature	Float32
Thickness	Float32
UserGUID	Char(16)
SubscriptionGUID	Char(16)
Reserved	Byte(48)
SerialNumber	Unsigned Short

FirmwareVersion	Unsigned Short
MinimumThickness	Float32
AverageCount	Unsigned Short
TxCoilIndex	Byte
RxCoilIndex	Byte
Data	Float32(10000)

2.3.9 Transparent Serial

This command is deprecated due to serial traffic being encrypted and is expected to be removed.

This command shall provide a convenient method for Inductosense software to send serial commands from software to UWM via the Wi-Fi link.

Url:	/serial
Request Type:	PUT
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Dependent on body data
Body Data Type:	application/octet-stream
Body Example:	0x49, 0x00, 0x00, 0x02, 0xFF, 0xF0, 0x00, 0x00
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/octet-stream
Response Example 1:	0x49, 0x00, 0x00, 0x05, 0x06, 0x00, 0x00, 0x03, 0x00, 0x00
Response Example 2:	

This command expects body data populated with an array of bytes and length to be transparently sent over the serial link to the UWM.

A successful response to this command will return an array of data received via the serial link to the UWM.

If the UWM does not respond to the serial data sent within 100mS, the response shall return a 400 Bad Request and a *message* stating that the 100mS period has elapsed indicating a timeout.

The security level of this request is dependent on the serial data that is requested, note that no level 2 commands shall be sent via this method.

2.3.10 Sleep

This command should be used to put the ESP32 module into deep sleep mode. It shall not be able to recover from deep sleep mode without power cycling the device.

It is anticipated that this command is used in *Integrated* mode where the commands to control the UWM come from the drone rather than the ESP32. This command would be used to put the ESP32 into an ultra-low power sleep mode where it is not required.

Url:	/sleep
Request Type:	POST
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	None
Body Example:	
Response Codes:	200, 401, 403, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0, “message”: “OK”}

This command does not expect any body data to be sent alongside the command.

The *responseCode* and *message* response to this command shall always be **0** and **OK** respectively.

2.3.11 Get Wi-Fi Settings

This command shall be used to retrieve the Wi-Fi configuration of the ESP32 device.

Url:	/wifi
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	None
Body Example:	
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/json
Response Example 1:	{“ssid”: “test”, “channel”: 1, “ssid_hidden”: false, “mac”: “AB:CD:EF:00:01:02”, “timeout”: 300}

A successful response shall indicate the current Wi-Fi configuration. Passphrase is omitted from the response for security.

Name	Type	Explanation
ssid	String	AP SSID that the UWM broadcasts
channel	UInt8	Channel that the UWM is currently set to
ssid_hidden	Boolean	Indicates SSID visibility when other devices are performing a scan
mac	String	MAC Address of the UWM
timeout	UInt16	Value in seconds that the UWM will disconnect a client (STA) from the AP if the client does not send any data. Default value is 300 seconds
country	String	2-character country code which is used by the module to limit transmit power and channel usage for regulatory purposes
long_range	Boolean	Indicates whether ESP-LR mode is enabled.

2.3.11.1 Valid Country Codes

The default country code is “01” which is world safe mode.
Possible country codes are as follows.

“01”,"AT","AU","BE","BG","BR","CA","CH","CN","CY","CZ","DE","DK","EE","ES","FI","FR","GB","GR","HK","HR","HU","IE","IN","IS","IT","JP","KR","LI","LT","LU","LV","MT","MX","NL","NO","NZ","PL","PT","RO","SE","SI","SK","TW","US"

2.3.12 Set Wi-Fi Settings

This command shall be used to set the Wi-Fi configuration of the ESP32 device.
It is expected that a *Restart Wi-Fi* command shall be sent to the device for the changes to be made effective. Using the *action* parameter described in *Restart Wi-Fi* may be used to restart the Wi-Fi module without issuing the *Restart Wi-Fi* command.

Url:	/wifi
Request Type:	POST
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	application/json
Body Example 1:	{“ssid”: “test”, “passphrase”: “test”, “channel”: 1, “ssid_hidden”: false}
Body Example 2:	{“channel”: 2}
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0, “message”: “OK”}
Response Example 2:	{“responseCode”: 1, “message”: “An error occurred setting Wi-Fi settings”}

A successful response, shall consist of a *responseCode* and a *message* where applicable.
A *responseCode* of 0 shall indicate that the request was performed successfully, a *responseCode* of 1 shall indicate that the request was not performed successfully, and *message* shall be populated with a reason to indicate why the request could not be performed.

It is expected that this command shall return a *responseCode* of 0 to indicate that the requested settings have been set. However, a restart of the Wi-Fi is required for the settings to be made effective.

It is expected that this command shall return a *responseCode* of 1 to indicate that the requested settings could not be set.

If no body data is sent with this request, or the body data is malformed and could not be parsed, the request will return a 400 Bad Request with no further data.

Body data may contain all or a partial Wi-Fi configuration. In the example used above, only the channel parameter is set.

The following table indicates valid parameters for settings that may be changed..

Name	Type	Size
ssid	String	32 characters
passphrase	String	64 characters, minimum 8 characters
channel	UInt8	Value can be 1 - 13
ssid_hidden	Boolean	
timeout	UInt16	Minimum value of 10 and Maximum value of 65535. A value of less than 30 is not recommended.
country	String	2 characters, see 2.3.11.1 for a list of valid country codes
long_range	Boolean	

Strings should not be empty and should respect the size.

2.3.13 Restart Wi-Fi

This command shall be used to restart the Wi-Fi service on the ESP32 device.

Url:	/wifi
Request Type:	POST
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 1
Body Data Type:	application/json
Body Example:	{“action”: 1}
Response Codes:	200, 400, 401, 403, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0, “message”: “OK”}

The Wi-Fi service shall restart after the response has been sent to the caller.

Additionally, the action parameter may be used with the *Set Wi-Fi Settings* to automatically restart the Wi-Fi service after changing settings.
Omitting the *action* parameter from the *Set Wi-Fi Settings* command will only result in the settings being saved and not result in the Wi-Fi service being restarted.

Action Enumeration	Function
0	No Action
1	Restart

2.3.14 Get Information

This command shall be used to retrieve generic information about the Wi-Fi module and HDC hardware.

Url:	/info
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 0
Body Data Type:	None
Body Example:	
Response Codes:	200, 401, 403, 500
Response Type:	application/json
Response Example 1:	{“uwm_fw”: “1.0”, “hdc_fw”: “3.1”, “hdc_serial”: “00001” “temperature”: 38}

A successful response will return useful information about the system, including, but not limited to firmware versions, serial numbers and indicated temperature of the system.

Name	**Type**	**Format**
uwm_fw	String	Major.Minor
hdc_fw	String	Major.Minor
hdc_serial	String	5 Characters
temperature	Int32	Celcius

2.3.15 Request Challenge

This command requests a challenge from the UWM hardware to allow UWM to determine whether iDART is genuine. iDART should follow the response to this command with Send Challenge Response request.

Url:	/auth
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 0
Body Data Type:	None
Body Example:	
Response Codes:	200, 401, 500
Response Type:	application/json
Response Example 1:	{“challenge”: …}

The challenge shall be a hexadecimal string representation of 32 bytes that form the challenge.

2.3.16 Send Challenge Response

This command should follow Request Challenge.

Url:	/auth
Request Type:	POST
Parameters:	None
Authorization:	Basic Authentication

Security Level:	Level 0
Body Data Type:	application/json
Body Example:	{“response”: …}
Response Codes:	200, 400, 401, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0, “message”: “OK”}

The challenge response shall be a hexadecimal string representation of 32 bytes that form the response to the challenge.

2.3.17 Send Challenge

This command is used to send a challenge to the UWM hardware such that iDART can determine whether the UWM hardware is genuine. The response to this message shall contain the challenge response.

Url:	/auth
Request Type:	POST
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 0
Body Data Type:	application/json
Body Example:	{“challenge”: …}
Response Codes:	200, 400, 401, 500
Response Type:	application/json
Response Example 1:	{“response”: …}

The challenge and response shall be a hexadecimal string representation of 32 bytes that form the challenge and response.

2.3.18 Keep Alive

This command shall be used to keep the current security level active by acting as a keep-alive signal.

Url:	/ping
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 0
Body Data Type:	None
Body Example:	None
Response Codes:	200, 400, 401, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0}

This command is always successful and always returns a response code of 0. There shall be no error messages.

2.3.19 Get Statistics

This command shall be used for debugging and shall indicate some basic statistics of the device, such as transmitted packets across both serial and Wi-Fi. The response content of this message shall not be strictly defined as it is for debugging purposes only.

Url:	/stats
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 0
Body Data Type:	None
Body Example:	
Response Codes:	200, 400, 401, 500
Response Type:	application/json
Response Example 1:	{…}

2.3.20 Get Error

Url:	/error
Request Type:	GET
Parameters:	None
Authorization:	Basic Authentication
Security Level:	Level 0
Body Data Type:	None
Body Example:	
Response Codes:	200, 400, 401, 500
Response Type:	application/json
Response Example 1:	{“responseCode”: 0, “errorCodes”: [0x00000000, 0x00000000, 0x00000000, 0x00000000]}

Error codes are split into 4 sections 32-bit bitfields, the following table describes the indexes used in the errorCode array. Bitfield data will be described below.

Array Index	**Error Section**
0	Wi-Fi errors
1	HTTPS server errors
2	Serial errors
3	Generic errors

2.3.20.1 Wi-Fi Error Bitfields

Value	**Description**
0x00000001	Failed to start
0x00000002	Invalid configuration
0x00000004	Invalid certificates

0x00000008	Invalid MAC address
0x00000010	Unable to respond
0x00000020	Failed to set mode (AP)
0x00000040	Failed to set protocol (LR / Normal)

2.3.20.2 HTTPS Server Error Bitfields

Value	Description
0x00000001	Error loading certificates
0x00000002	Error storing certificates
0x00000004	Error registering API uris
0x00000008	Error starting or stopping
0x00000010	Error loading keys
0x00000020	Failed to register API handlers

2.3.20.3 Serial Error Bitfields

Value	Description
0x00000001	Timeout
0x00000002	Parser failure
0x00000004	Handler failure
0x00000008	Response failure
0x00000010	Unable to read length
0x00000020	Unable to initialise serial port

2.3.20.4 General Error Bitfields

Value	Description
0x00000001	Partition validation error
0x00000002	Flash storage error
0x00000004	Network interface error
0x00000008	Internal event loop failure
0x00000010	Failed to read MAC address
0x00000020	Not provisioned

2.4 Response Codes

2.4.1 200 OK

A 200 OK will be sent in response to a request which meets all of the following requirements:

- Authorization is successful
- Body data (where applicable) has parsed successfully and meets the requirements of the request

- The request has processed successfully on UWM and a response is available

If any of the above requirements fail to be met a 200 OK response code will not be sent, and the appropriate error code will be returned.

2.4.2 400 Bad Request

A 400 Bad Request will be sent in response to a request which meets one or more of the following requirements:

- Body data (where applicable) was malformed, corrupted, or does not meet the requirements of the request
- UWM failed to process the request (e.g. UWM returns a NACK to the outgoing serial command, or UWM fails to respond in a timely manner; timeout)

2.4.3 401 Unauthorized

A 401 Unauthorized will be sent in response to a request which fails to pass basic HTTP authorization.

2.4.4 403 Forbidden

A 403 Forbidden will be sent in response to an issued command that does not have the correct authentication level.

2.4.5 404 Not Found

A 404 Not Found will be sent in response to a request sent to any URL not listed in section 2.3. (e.g. GET /url)

Additionally, a 404 Not Found will be sent in response to a request sent to a URL listed in section 2.3 with an incorrect request type (e.g. PUT /scan)

2.4.6 500 Internal Server Error

A 500 Internal Server Error will be sent in response to any request where an internal error has occurred.

It is expected that a 500 Internal Server Error should be interpreted as a fault condition within the UWM Wi-Fi module rather than a fault with the UWM.

2.5 Error / Response Codes

Inductosense would like error codes and messages to be provide more details about what caused the error conditions.

Error conditions are expected to be unique to the individual commands that requested them. In general, an error code or response code of 0 indicates a successful state and an error code or response code greater than 0 indicates an error condition.

Further details will be provided on error conditions as development progresses.