

EVK   
Messaging Protocol  
  
Reference Guide

Revision 1

Vayyar Imaging Ltd.

Web: <https://vayyar.com>

Copyright 2020 Vayyar Imaging Ltd. All rights reserved.

This document is the property of Vayyar and all materials and information contained herein are the confidential and proprietary information of Vayyar and are protected by international copyright laws. The copyrights are owned by Vayyar or the original creator of the material. The materials and information contained herein are provided to you for internal information purposes only, and no portion of the materials and information herein may be copied, reproduced, distributed, transmitted, displayed, published, broadcasted or used in any way whatsoever directly or indirectly without the prior written consent of Vayyar or in any case of third party materials, without that third party's consent. This document shall be returned to Vayyar promptly upon request.

The materials and information contained herein are provided "AS IS" and Vayyar does not provide any warranties of any kind, whether express or implied, including but not limited to implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Description |
| 1 | August 2020 | Noam Hoze |  |
| 2 | September 2020 | Noam Hoze |  |

Table of Contents

[1 Introduction 4](#_Toc49192781)

[1.1 Audience 4](#_Toc49192782)

[1.2 Related Materials 4](#_Toc49192783)

[1.3 Glossary 4](#_Toc49192784)

[2 EVK Messaging Protocol 5](#_Toc49192785)

[2.1 Websocket Connection 5](#_Toc49192786)

[2.2 JSON Format 5](#_Toc49192787)

[2.2.1 Message Type 5](#_Toc49192788)

[2.2.2 Message ID 6](#_Toc49192789)

[2.2.3 Payload 6](#_Toc49192790)

[2.3 Binary Format 7](#_Toc49192791)

[2.3.1 Message Size 7](#_Toc49192792)

[2.3.2 Header 7](#_Toc49192793)

[2.3.3 Payload 8](#_Toc49192794)

[3 Message Flow and System States 9](#_Toc49192795)

[4 Messages 10](#_Toc49192796)

[4.1 Command Messages 10](#_Toc49192797)

[4.1.1 SET\_PARAMS 10](#_Toc49192798)

[4.1.2 GET\_PARAMS 14](#_Toc49192799)

[4.1.3 RESET\_PARAMS 16](#_Toc49192800)

[4.1.4 SET\_OUTPUTS 18](#_Toc49192801)

[4.1.5 START 19](#_Toc49192802)

[4.1.6 STOP 19](#_Toc49192803)

[4.2 Query Messages 20](#_Toc49192804)

[4.2.1 GET\_STATUS 20](#_Toc49192805)

[4.2.2 JSON\_DATA / BINARY\_DATA 21](#_Toc49192806)

[4.3 Notification Messages 29](#_Toc49192807)

[4.3.1 EVK\_PROLOG 29](#_Toc49192808)

# Introduction

This reference guide describes how to generate and parse the EVK Messaging Protocol messages.

## Audience

This document is targeted at the following audiences:

* Product and development personnel who are interested in using the EVK Messaging Protocol to manage Vayyar RF sensors.

## Related Materials

Refer to the following documents for additional information:

* Vayyar Application Note 001 - API Reference Guide

## Glossary

|  |  |
| --- | --- |
| Term | Definition |
| API | Application Programming Interface |
| EVK | Evaluation Kit |
| GUI | Graphical User Interface |
| ID | Identifier |
| IP | Internet Protocol. An IP address is a numerical identifier assigned to a computing device or node in a TCP/IP network. The address is used to locate and identify the node in communications with other nodes on the network. |
| JSON | JavaScript Object Notation |
| MTI | Moving Target Indicator |
| PC | Personal Computer |

# EVK Messaging Protocol

The EVK Messaging Protocol serves as a messaging platform between an API client and the Vayyar EVK Engine server. The protocol can be invoked following the establishment of a [websocket connection](#_Websocket_Connection) between the client and the server.

The protocol supports messaging in two formats:

* [JSON format](#_JSON_Format)
* [Binary format](#_Binary_Format_1)

## Websocket Connection

The websocket connection between the API client and the Vayyar EVK Engine server serves a platform for the messaging protocol.

NOTE

The API Client software can run on the same PC as the EVK Engine, or on its own dedicated PC given that the EVK engine PC IP is accessible.

The websocket server runs on http://<EVK-PC-IP>:1234/, (where <EVK-PC-IP> = the IP address of the PC on which the EVK Engine is running). The API client, running either on the same PC or a different PC, should connect to ws://<EVK-PC-IP>:1234/

NOTE

The client-server connection is not secure, therefore the ws:// prefix should be used. The prefix wss:// is **not** supported.

## JSON Format

The EVK JSON message contains 3 keys: [Message Type](#_Message_Type), [Message ID](#_Message_ID), and [Payload](#_Payload_1), as illustrated in the figure below.



### Message Type

The message type field describes the message’s use. The following message types are supported:

* **COMMAND.** Command messages are sent by the client to read or write configuration settings, or to start or end an imaging recording or playback session. Responses by the server are also of this type.
* **QUERY**. Query messages are sent by the client to request data from the imaging session in JSON or binary format. Responses by the server are also of this type.
* **NOTIFICATION**. Notification messages are sent by the server to the client in JSON format.

### Message ID

The message ID field contains a string with the name of the message, and determines the message’s function and the content structure of the payload. The following message IDs are supported:

|  |  |  |
| --- | --- | --- |
| Command Message IDs | Query Message IDs | Notification Message IDs |
| * “SET\_PARAMS” * “GET\_PARAMS” * “RESET\_PARAMS” * “SET\_OUTPUTS” * “START” * “STOP” | * “GET\_STATUS” * “BINARY\_DATA” * “JSON\_DATA | * “EVK\_PROLOG” |

For [in-depth details](#_Messages) on each message type, see Chapter ‎4.

### Payload

The payload contains the actual data to be transmitted, and appears in JSON <key>:<value> format. The structure of the payload is determined by the message ID.

## Binary Format

The Binary Format is a special-purpose format that supports the transmission of binary data from an EVK Engine to an API client. The protocol is optimized for sending large matrices of imaging data over a Websocket connection, while supporting a high frame rate.

A Websocket Messaging Protocol Binary Format message contains a [message size field](#_Message_Size), [header](#_Header), and [payload](#_Payload), as illustrated in the figure below.



NOTE

The server may transmit data either in big endian (most-significant byte first) or little endian (least-significant byte first) format, depending on the host that is used to run the EVK engine.

### Message Size

The message size field is described in the table below.

|  |  |  |
| --- | --- | --- |
| Field | Field Length (Bytes) | Description |
| Message Size | 4 | The size of the entire message, in bytes. |

### Header

Header Structure

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Offset (Bytes) | 0 | 4 | 4+k | 4+k+2 | |
| Field | Header Length | ID String | Record  Separator | Field Name and  Unit Separator Block (block occurs ‘n’ times) | |
| Field Name ‘n’ | Unit Separator ‘n’ |
| Size (Bytes) | 4 | k =  variable length | 2 | m =  variable length | 2 |

Field Description

|  |  |  |
| --- | --- | --- |
| Field | | Description |
| Header Length | | The header length, in bytes |
| ID string | | Unicode string (string of 16-bit characters) |
| Record Separator | | 0x001e |
|  | Field Name ‘n’ | The field name (key) associated with the field appearing in the payload. |
|  | Unit Separator ‘n’ | 0x001f |

### Payload

The payload consists of one or more concatenated data fields. Each data field can represent a matrix of values or a single scalar value.

Payload Structure

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Offset (Bytes) | 0 | 4 | 8 | 12 | 12+4\*n |
| Field | Field Length | Data Type | Number of Dimensions | Vector of Dimensions | Matrix |
| Size (Bytes) | 4 | 4 | 4 | 4\*n | see description |

Field Description

|  |  |
| --- | --- |
| Field | Description |
| Field Length | The payload length, in bytes |
| Data Type | The data type, according to the following enumeration:   * 0: int8 * 1: uint8 * 2: int16 * 3: uint16 * 4: int32 * 5: uint32 * 6: float32 * 7: float64 |
| Number of Dimensions ‘n’ | The number of dimensions (or “sub-fields”) in the field.   * 0: scalar * 1..n: matrix |
| Vector of Dimensions | This field does not appear if n=0 (scalar).  If n>0 (matrix), this field appears n times. |
| Matrix | The matrix data.  Matrix size:  If number of dimensions = 0,  size of scalar = size of (Data Type)  If number of dimensions > 0, size of matrix = size\_of (Data Type) \*  (Dimension1 length) \*  (Dimension2 length) \* … \*  (Dimensionn length) |

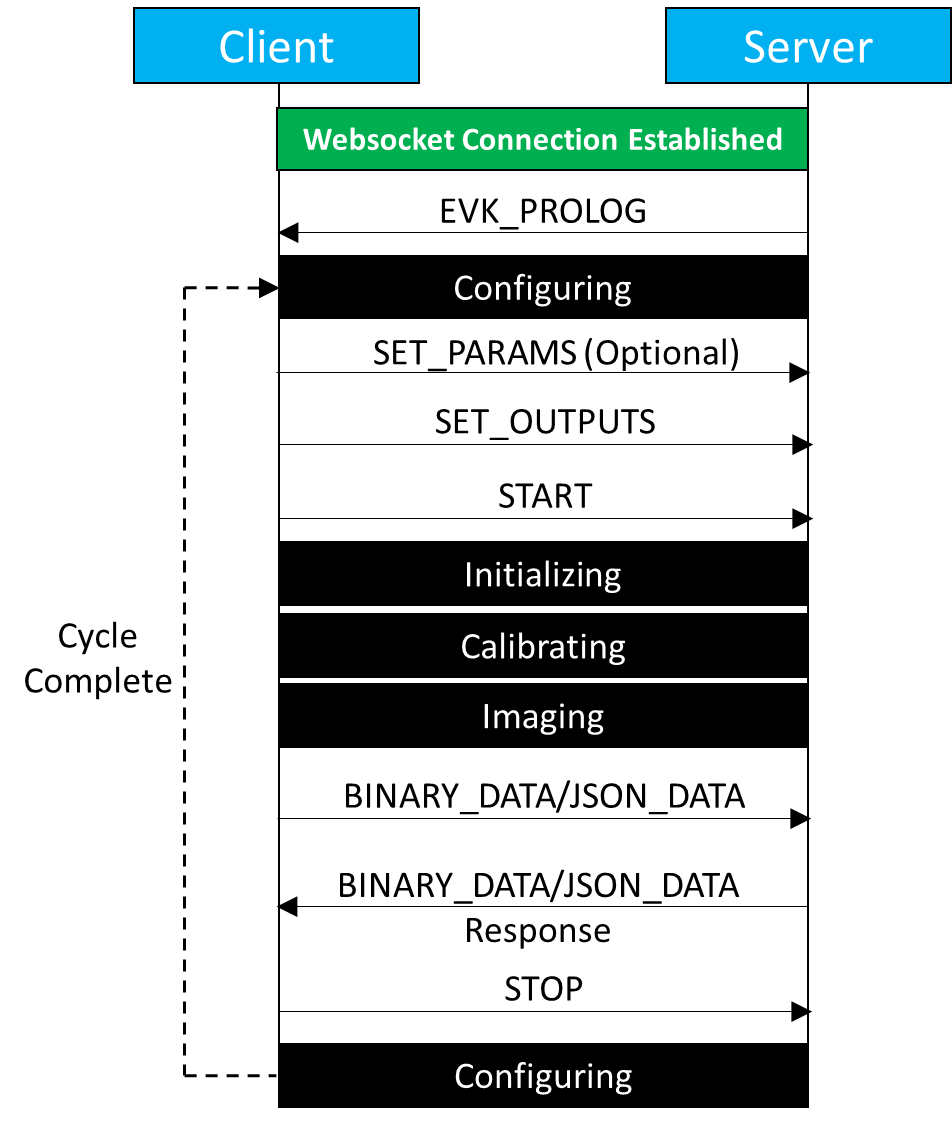
# Message Flow and System States

The imaging system’s state machine and the messaging flow between the API Client and Server (EVK Engine) are depicted in the figure below.

* The Websocket Connection Established event appears on a green background. The connection enables the transmission of messages between the client and the server.
* System messages are arrowed.
* System states appear on a black background.

NOTE

System state changes trigger notification messages with the ID, “GET\_STATUS”.



Notes:

* Messages sent from the client to the server (EVK Engine) before establishing Websocket connectivity are not received by the server.
* The SET\_PARAMS message is optional. If the message is not sent, then default parameters are used.

# Messages

The EVK Messaging Protocol supports the following message types:

* [Command messages](#_Command_Messages) – used to read or write configuration settings
* [Query messages](#_Query_Messages) – used to request imaging data
* [Notification messages](#_Notification_Messages) – used to indicate the occurrence of system events

## Command Messages

This section contains in-depth descriptions of the following commands:

* [SET\_PARAMS](#_SET_PARAMS)
* [GET\_PARAMS](#_GET_PARAMS)
* [RESET\_PARAMS](#_RESET_PARAMS)
* [SET\_OUTPUTS](#_SET_OUTPUT)
* [START](#_START)
* [STOP](#_STOP)

### GET\_PARAMS

Requests an object containing the currently-defined configuration parameters.

Request

|  |  |
| --- | --- |
| Type | “COMMAND” |
| ID | “GET\_PARAMS” |
| Parameter | Description |
| For a complete list of parameters and descriptions, see [SET\_PARAMS](#_SET_PARAMS). | |

Request Example

### RESET\_PARAMS

This command triggers two actions:

* Resets the EVK engine parameters to their default values.
* Requests an object containing the default configuration parameters (similarly to the [GET\_PARAMS](#_GET_PARAMS) command).

Request

|  |  |
| --- | --- |
| Type | “COMMAND” |
| ID | “RESET\_PARAMS” |
| Parameter | Description |
| For a complete list of parameters and descriptions, see [SET\_PARAMS](#_SET_PARAMS). | |

Request Example

### SET\_OUTPUTS

Sends an object specifying the data to be recorded during each frame. The data is stored in a queue, and transmitted to the API client in response to a JSON\_DATA or BINARY\_DATA query. The data can be transmitted in two formats:

* “binary\_outputs” – data output in binary format. This format is useful for large transfers of raw data (I, Q) and 2D/3D mat. Binary output supports numerical data only.
* “json\_outputs” – data output in JSON format. This format is used to transfer all types of data, but is less efficient for large data transfers.

Request

|  |  |
| --- | --- |
| Type | “COMMAND” |
| ID | “SET\_OUTPUTS” |
| Payload Parameters | Description |
| clientId | The unique ID of the API client application |
| json\_outputs | The JSON data to be output following a query. |
| Binary\_outputs | The binary data to be output following a query. |

Request Example

|  |
| --- |
| {  Type: "COMMAND",  ID: "SET\_OUTPUTS",  Payload: {clientId: "client\_3hja23xfj",  json\_outputs: ["ValidPoints", "TrailPoints", "RawPoints"],  binary\_outputs: [“LocationMatrix”,”NumOfPeople”]   }  } |

### START

The START command starts the imaging process. In additions, it is used to specify the data to be recorded during each frame. For further information see [SET\_OUTPUTS](#_SET_OUTPUTS).

The type of imaging session initiated is dependent on the following [SET\_PARAMS](#_SET_PARAMS) flag settings:

* MPR.save\_to\_file – When set to 1, the system saves raw sensor data for a future playback session.
* MPR.save\_image\_to\_file – When set to 1, the system saves processed 3D images for a future playback session.
* Cfg.OutputData.save\_to\_file – When set to 1, the system creates a log file of tracked target locations.
* MPR.read\_from\_file – When set to 1, the system performs a playback of previously recorded files, without requiring a sensor.
* MPR.save\_pointCloud\_to\_file – When set to 1, the system saves files containing the point cloud data of the arena per frame.
* Request

|  |  |
| --- | --- |
| Type | “COMMAND” |
| ID | “START” |
| Payload Parameters | Description |
| clientId | The unique ID of the API client application |
| json\_outputs | The JSON data to be output following a query. |
| binary\_outputs | The binary data to be output following a query. |

Request Example

|  |
| --- |
| {  Type: "COMMAND",  ID: "START",  Payload: {clientId: "client\_3hja23xfj",  json\_outputs: ["ValidPoints", "TrailPoints", "RawPoints"],  binary\_outputs: [“LocationMatrix”,”NumOfPeople”]   } } |

### STOP

The STOP command stops the imaging process.

Request

|  |  |
| --- | --- |
| Type | “COMMAND” |
| ID | “STOP” |

Request Example

|  |
| --- |
| {  Type: "COMMAND",  ID: "STOP",  Payload: {} } |

## Query Messages

The Query message requests imaging data and EVK Engine status information from the server.

The following query messages IDs are supported:

* [GET\_STATUS](#_GET_STATUS)
* [JSON\_DATA/BINARY\_DATA](#_JSON_DATA_/_BINARY_DATA)

### GET\_STATUS

The GET\_STATUS query is used to request tracker status. The query response uses the same ID.

Request

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “GET\_STATUS” |

Response

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “GET\_STATUS” |
| Payload Parameters | Description |
| status | A string representing the current state of the EVK engine, one of the following:   * “INITIALIZING” * “CALIBRATING” * “CONFIGURING” * “IMAGING” * “ERROR” |
| errorMessage | When status = “ERROR”, this parameter contains a textual error message. |
| numConnectedClients | The number of API clients currently connected to the EVK engine. |

Request Example

|  |
| --- |
| {  Type: "QUERY",  ID: "GET\_STATUS" } |

Response Example

|  |
| --- |
| {  Type: "QUERY",  ID: "GET\_STATUS",  Payload: {status: "ERROR", errorMessage: "No device found!",  numConnectedClients: 1} } |

### JSON\_DATA / BINARY\_DATA

The JSON\_DATA and BINARY\_DATA query messages are used to request a frame of imaging data that was recorded and stored in a FIFO queue by the EVK engine. The data is removed from the queue following transmission.

These queries do not require a payload. The server transmits the data that was specified in the [SET\_OUTPUTS](#_SET_OUTPUT) or [START](#_START) commands.

* If “json\_outputs” was specified in the SET\_OUTPUTS or START commands, then the JSON\_DATA query should be submitted after every frame.
* If “binary\_outputs” was specified in the SET\_OUTPUTS or START commands, then the BINARY\_DATA query should be submitted after every frame.

#### Number of People

The Number of People query is used to determine the number of targets in the arena.

The parameters that should be specified in the [START](#_START) or [SET\_OUTPUTS](#_SET_OUTPUT) command are listed below.

SET\_OUPUTS Payload

* NumOfPeople - The number of targets in the defined arena.

The request and response parameters appear in the tables below.

Request

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |

Response

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |
| Payload Parameters | Description |
| NumOfPeople | The number of targets in the defined arena. |

Request Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload: {}  } |

Response Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload:  {  NumOfPeople: 2  }  } |

#### Tracker Targets

The Tracker Targets query is used to request the location of up to 10 targets.

The parameters that should be specified in the [START](#_START) or [SET\_OUTPUTS](#_SET_OUTPUT) command are listed below.

SET\_OUPUTS Payload

* LocationMatrix - The x-, y-, and z-axis location of a target.
* targetsID. The target number.

The request and response parameters appear in the tables below.

Request

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |

Response

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |
| Payload Parameters | Description |
| targetsID | The target number. |
| LocationMatrix | The x-, y-, and z-axis location of a target. |

Request Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload: {}  } |

Response Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload:  {  targetsID: [1, 2, 0, 0, 0, 0, 0, 0, 0, 0],   * LocationMatrix: [[   [-0.5406589508056641, 0.3250932693481453, 1.5070443153381348],  [0.0991029813885688, 0.6120092272758484, 0.9153468608856201],  ["NaN", "NaN", "NaN"],  ["NaN", "NaN", "NaN"],  ["NaN", "NaN", "NaN"],  ["NaN", "NaN", "NaN"],  ["NaN", "NaN", "NaN"],  ["NaN", "NaN", "NaN"],  ["NaN", "NaN", "NaN"],  ["NaN", "NaN", "NaN"]]  }  } |

#### Cloud 3D Raw

The Cloud 3D Raw query is used to determine the intensity of the reflected signal from each point in the arena.

The parameters that should be specified in the [START](#_START) or [SET\_OUTPUTS](#_SET_OUTPUT) command are listed below.

SET\_OUPUTS Payload

* RawPoints. A 2-D matrix of all the voxels in the arena. Each voxel in the matrix is described by its Cartesian (X, Y, Z) location in meters, and the power level reflected to the sensor from each voxel in the arena.

The request and response parameters appear in the tables below.

Request

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |

Response

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |
| Payload Parameters | Description |
| RawPoints | A 2-D matrix of all the voxels in the arena. Each voxel in the matrix is described by its Cartesian (X, Y, Z) location in meters, and the power level reflected to the sensor from each voxel in the arena.  The matrix size is N \* 4, all values are floating point:   * N rows: 1 row for each voxel in the arena * 4 columns: X, Y, Z position of the voxel,  I intensity (normalized) |

Request Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload: {}  } |

Response Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload:  {  RawPoints: [[-0.028434, 0.201295, 1.889492, 1.590284],  [-0.019364, 0.280362, 1.859561, 2.179032],  ...]  }  } |

#### Heat Map

The Heat Map query is used to determine the intensity of the reflected signal from each point in the arena, in two and three dimensions.

The parameters that should be specified in the [START](#_START) or [SET\_OUTPUTS](#_SET_OUTPUT) command are listed below.

SET\_OUPUTS Payload

* RawPoints. A 2-D matrix of all the voxels in the arena. Each voxel in the matrix is described by its Cartesian (X, Y, Z) location in meters, and the power level reflected to the sensor from each voxel in the arena.
* rawImage\_XY. A 2-D image matrix, containing the power level reflected to the sensor from each voxel, averaged over the Z axis.

The request and response parameters appear in the tables below.

Request

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |

Response

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |
| Payload Parameters | Description |
| RawPoints | A 2-D matrix of all the voxels in the arena. Each voxel in the matrix is described by its Cartesian (X, Y, Z) location in meters, and the power level reflected to the sensor from each voxel in the arena.  The matrix size is N \* 4, all values are floating point:   * N rows: 1 row for each voxel in the arena * 4 columns: X, Y, Z position of the voxel,  I intensity (normalized) |
| rawImage\_XY | A 2-D image matrix, containing the power level reflected to the sensor from each voxel, averaged over the Z axis.  The matrix size is N \* M, all values are floating point:   * N: number of voxels in the X-axis * M: number of voxels in the Y-axis |

Request Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload: {}  } |

Response Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload:  {  RawPoints: [[-0.028434, 0.201295, 1.889492, 1.590284],  [-0.019364, 0.280362, 1.859561, 2.179032],  ...],  rawImage\_XY: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...],  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...],  ...]  }  } |

#### Point Cloud 2D, 3D Tracker

The Point Cloud 2D, 3D Tracker query is used to track target positions and movements in the arena.

The parameters that should be specified in the [START](#_START) or [SET\_OUTPUTS](#_SET_OUTPUT) command are listed below.

SET\_OUPUTS Payload

* RawPoints. A 2-D matrix of all the voxels in the arena. Each voxel in the matrix is described by its Cartesian (X, Y, Z) location in meters, and the power level reflected to the sensor from each voxel in the arena.
* TrailPoints. A 2-D matrix of the 10 most recent locations of the center of mass of each target cluster. Each location in the matrix is described by its Cartesian (X, Y, Z) location in meters. The last location value is the current location of the target.
* ValidPoints. A 2-D matrix of all the relevant points in the arena after target clustering. Each voxel in the matrix is described by its Cartesian (X, Y, Z) location in meters, and the ID number of the target it is clustered with.

The request and response parameters appear in the tables below.

Request

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |

Response

|  |  |
| --- | --- |
| Type | “QUERY” |
| ID | “JSON\_DATA” |
| Payload Parameters | Description |
| RawPoints | A 2-D matrix of all the voxels in the arena. Each voxel in the matrix is described by its Cartesian (X, Y, Z) location in meters, and the power level reflected to the sensor from each voxel in the arena.  The matrix size is N \* 4, all values are floating point:   * N rows: 1 row for each voxel in the arena * 4 columns: X, Y, Z position of the voxel, I intensity (normalized) |
| TrailPoints | A 2-D matrix of the 10 most recent locations of the center of mass of each target cluster. Each location in the matrix is described by its Cartesian (X, Y, Z) location in meters. The last location value is the current location of the target.  The matrix size is N \* 4, all values are floating point:   * N rows: number of target \* 10 locations per target * 4 columns: X, Y, Z position of the center of mass of the target cluster,  ID number of the target |
| ValidPoints | A 2-D matrix of all the relevant points in the arena after target clustering. Each voxel in the matrix is described by its Cartesian (X, Y, Z) location in meters, and the ID number of the target it is clustered with.  The matrix size is N \* 4, all values are floating point:   * N rows: 1 row for each relevant point in the arena * 4 columns: X, Y, Z position of the relevant point, ID number of the target |

Request Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload: {}  } |

Response Example

|  |
| --- |
| {  Type: "QUERY",  ID: "JSON\_DATA",  Payload:  {  RawPoints: [[-0.028364, 0.204795, 1.689492, 1.690484],  [-0.021967, 0.300362, 1.759564, 2.079435],  ...],  TrailPoints: [[-0.342742, 1.515993, 1.782838, 1],  [-0.351295, 1.589492, 1.790284, 1],  ...],  ValidPoints: [[-0.218470, 0.188228, 1.740223, 1],  [-0.207864, 0.202208, 1.750829, 1],  ...]  }  } |

## Notification Messages

This section contains in-depth descriptions of the following notification messages:

* [EVK\_PROLOG](#_EVK_PROLOG)

### EVK\_PROLOG

The EVK engine sends a notification to the client following establishment of a websocket connection.

Notification

|  |  |
| --- | --- |
| Type | “NOTIFICATION” |
| ID | “EVK\_PROLOG” |
| Payload Parameters | Description |
| protocolVersion | The version number of the EVK Messaging Protocol that is supported by the EVK engine. |

Notification Example

|  |
| --- |
| {  Type: "NOTIFICATION",  ID: "EVK\_PROLOG",  Payload:  {protocolVersion: "0.0.1"}  } |