SDK Document

Time	Version	Author	Modified contents
2019-02-28	v1.0 beta	Jinming Chen	Initial publish
2019-03-05	v1.0 beta2	Jinming Chen	Add section of shared memory
2019-03-06	v1.0 beta3	Jinming Chen	Add section of video data and range data
2019-03-07	v1.0 beta4	Jinming Chen	Add new project and open project

Overview

This is the official SDK that helps third party developers to write apps to manipulate 3D cameras of Shining3D's 2X series.

It runs as a standalone executable and communicates with 3rd apps via ZMQ. The interfaces try to keep consistent with the latest EXScan Pro®, the official controlling software of 2X series cameras.

Installation

The SDK consists of only two files:

- Sn3DPlatform.exe: the main executable of the SDK.
- platform.ini: the configuration file.

The executable cannot run alone. It has dependencies of the controlling software. Therefore, these two files have to be put into the root directory of EXScan Pro installation path, i.e. in the same directory of EXScanPro.exe.

Configuration & run

Currently there are only two parameters in platform.ini in version v1.0:

- `Pub`: the publish url of ZMQ. Default is `tcp://*:11398`.
- `Rep`: the request url of ZMQ. Default is `tcp://*:11399`.

Normally you only need to change the ports if you want.

Double click Sn3DPlatform.exe and then the SDK is ready to talk to you via ZMQ interfaces.

Interfaces

Each message between Sn3DPlatform.exe and your app consists of two frames:

- Envelop: An ASCII string with a maximum length of 255. It contains two parts:
  - Version: the version of this message. Currently only `"v1.0"` is supported.
  - Commands: the actual commands of this message. The commands may have several parts, who are joined by character `/`.
- Payload: A byte array with a maximum length of 1024. It could be an integer, a string, or a JSON object, depending on the certain command.

The heartbeat message is a special message publishing from the SDK. It has only the evelop. Third party app should pay attention if they receive a message with such envelop.

There are three types of interfaces:

- **Publish**: The publish messages are published automatically by SDK when the corresponding properties are changed.
- **Request Get**: Third apps use this interface to get certain value from the device with a REQ/REP pattern.
- **Request Set**: Third apps use this interface to set certain value on the device with a REQ/REP pattern.

To receive all the publish messages, you should connect to the `Pub` url. To do Req/Rep communication to the SDK, you should connect and send/receive data to the `Rep` url.

The payload has different types:

- **Int Bool**: integar with 4 bytes, 0 means `false`, 1 measn `true;
- **Int**: signed integar with 4 bytes;
- **Int LL**: signed longlong integar with 8 bytes;
- **String**: C-style string;
- **JSON**: JSON object.

Heartbeat

This is a publish message broadcasted by the SDK repeatedly to indicate the main SDK process is still alive.

Type	Envelop	Payload

Type	Envelop	Payload
Publish	v1.0/hb	None

The interval of heartbeat is 1 seconds currently. Third app who cares about the aliveness of SDK could moniter the heartbeat. If no heartbeat comes any more, the main SDK process should be dead and needs relaunching.

Asynchronous actions

Many actions on the device take time to perform. To inform the third party apps the current status of the actions, the SDK broadcasts three messages/signals:

- **The beginning messages**: indicating that the action has begun.
- **The progress messages**: telling how far the action has gone. The progress value is between 0 and 100. It is ideal for developers to bind the value to a visual progress bar. Note: some actions don't have this kind of messages.
- **The finishing messages**: signaling that the action has finished.

Type	Envelop	Payload
Publish	v1.0/beginAsyncAction	JSON
Publish	v1.0/progress	Int
Publish	v1.0/finishAsyncAction	JSON

All the asynchronous actions are listed below:

- `"AAT_CHECK_DEVICE"`: Check the device.
- `"AAT_ENTER_CALI"`: Enter the calibration state.
- `"AAT_CHANGE_CALI_TYPE"`: Change the calibration type.
- `"AAT_EXIT_CALI"`: Exit the calibration state.
- `"AAT_COMPUTE_CALI"`: Compute the calibration.
- `"AAT_NEW_PROJECT"`: Create a new project.
- `"AAT_OPEN_PROJECT"`: Open the specified project.
- `"AAT_ENTER_SCAN"`: Enter the scanning state.
- `"AAT_EXIT_SCAN"`: Exit the scanning state.
- `"AAT_CANCEL_SCAN"`: Cancel the current scanning.
- `"AAT_END_SCAN"`: Finish the current scanning.
- `"AAT_SCAN"`: Start scanning.
- `"AAT_WHITE_BAL"`: Compute the white balancing.
- `"AAT_MANUAL_ALIGN"`: Start manual alignment for multiple point clouds.
- `"AAT_MESH"`: Start warping/meshing the point cloud.
- `"AAT_SIMPLIFY"`: Simplify the current scanned model.
- `"AAT_EDIT"`: Edit the current model.
- `"AAT_EDIT_LIST"`: Edit the data list for the current scanning project.
- `"AAT_DEL_POINT_CLOUD"`: Delete certain point cloud.
- `"AAT_SAVE"`: Save the current scanned model to the project.
- `"AAT_FIX_SINGLE_EDIT"`: Edit a single patch of point cloud in fix scan mode.
- `"AAT_AXIS_VERIFY"`: Compute & adjust the turnable axis.

- `"AAT_FIX_SCAN"`: Broadcast current state in fix scan mode.
- `"AAT_FIX_REMOVE_DATA"`: Remove data in fix scan mode.
- `"AAT_FIX_UPDATE_DATA_RT"`: Update the RT matrix in fix scan mode.
- `"AAT_EXPORT_SHARE_DATA"`: Export shared data.

The beginning JSON format is below:

```json
{
    "type": "AAT_CHECK_DEVICE",
    "props": {}
}
```

The `props` differs between different actions. It will be explained in detail under the corresponding action interface later.

The finishing JSON format is below:

```json
{
    "type": "AAT_CHECK_DEVICE",
    "result":{
        "success": false,
        "error": 2,
        "detail": "Cannot connect to the device."
    },
    "props": {}
}
```

The `props` differs between different actions. It will be explained in detail under the corresponding action interface later.

Shared memory

Messages through ZMQ are commands, signals or small data. It is not feasible to transfer large data through ZMQ since it involves redundant copying. We use shared memories to transfer large data.

There are 6 different types of large data transferring via shared memories currently:

- `"MT_POINT_CLOUD"`: Point cloud captured by the device and sent to apps.
- `"MT_DELETE_POINTS"`: Indices of points that need deleting. It is sent from apps to the device.
- `"MT_MARKERS"`: The positions information of markers captured by the device and sent to apps.
- `"MT_TRI_MESH"`: Triangle mesh information sent from the device to apps.
- `"MT_VIDEO_DATA"`: Video streaming sent from the device to apps.
- `"MT_RANGE_DATA"`: Single frame data with depth information sent from device to apps in fix mode only.

To receive shared memories, you need to do the following steps:

1. Setup a new `ZMQ_REQ` type ZMQ socket, and bind to a url.

2. Register to the SDK by calling `v1.0/scan/register` and send the url. It should be done before the scanning, otherwise data might be lost.
3. Wait for requesting from the SDK with the key and type of the shared memory.
4. Open the shared memory using language-specific manner and analyzing the data.
5. Reply the SDK with Int Bool `true` if you sucessfully handle the data, `false` otherwise.
6. Finish analyzing. Return to Step 3, waiting for the next large data.

Related interfaces are:

Type	Envelop	Payload
Request	v1.0/scan/register	String
Request	v1.0/scan/unregister	String

The message sent from SDK to the app contains JSON with the information (key and type) of the shared memory. It has no envelop. The JSON definition is below:

```
{
    "type": "MT_POINT_CLOUD",   // The data type
    "key": "qtipc_XXXXXXXXX",   // The native shared memory key
    "name": "currentPointCloud",// The name of current data
    "offset": 100,              // The offset from the beginning of the
data (bytes)
    "props":{}                  // type-specific parameters
}
```

Open from other languages

The key is a native mapped file descriptor on Windows. You can use corresponding functions in your favorite language to open and manipulate it. For example, in Python, the shared memory can be opend like this:

```python
import mmap
shm = mmap.mmap(0, 512, 'qtipc_XXXXXXXXX')
```

Then you can use member functions of mmap to read/write the shared memory.

In C#, the same thing can be done like this:

```csharp
using System.IO.MemoryMappedFiles;
MemoryMappedFile mm = MemoryMappedFile.OpenExisting(@"qtipc_XXXXXXXXX",

MemoryMappedFileRights.ReadWrite);
using(var accessor = mm.CreateViewAccessor(0, 512)){
    // read/write using accessor
}
```

Properties and structures on the shared memory

Different data types have different props and structures.

### Point cloud related data

For `MT_POINT_CLOUD`, `MT_MARKERS`, `MT_RANGE_DATA` and `MT_TRI_MESH`, the props definition is:

```
{
    "pointCount": 1000,     // The number of points
    "hasTexture": false,    // Whether there is texture attribute for each
point
    "hasNormal": true,      // Whether there is normal attribute for each
point
    "incremental": true,    // Whether this data is incremental to the
last data
    "hasMarkers": true,     // Whether this data contains markers
    "haveUsed": false,      // [Todo]
    "hasTexturePicture": false,// [Todo]
    "faceCount": 500,       // The number of triangles
    "textureImgWidth": 512, // The width of texture image
    "textureImgHeight": 512,// The height of texture image
    "textureUVCount": 1000, // [Todo]
    "hasFaceNormal": true,  // Whether there is normal attribute for each
triangle
    "packID": 0,            // Current package index, used for data
segmentation
    "totalPacks": 10        // Total package count, used for data
segmentation
}
```

There are 8 different named shared memories currently:

- `"currentPointCloud"`: Current frame of point cloud.
- `"currentMarker"`: Current frame of markers.
- `"wholePointCloud"`: The whole point cloud.
- `"failedPointCloud"`: Current frame of point cloud that cannot be aligned to existing points.
- `"frameMarkerPoint"`: Global marker file information.
- `"wholeMarkerPoint"`: The whole markers information.
- `"meshData"`: The triangle mesh data.
- `"rangeData"`: One frame of point cloud data in fix mode.

For `currentPointCloud`, `failedPointCloud` and `wholePointCloud`, the structures on the shared memory starting at offset `offset` are:
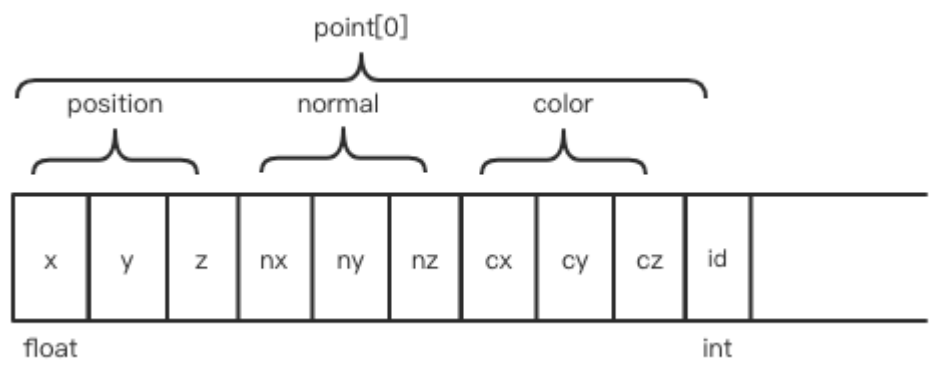
If `hasNormal` is `false`, then the normal part will be missing. If `hasTexture` is false, then the color part will be missing. If `incremental` is false, the id part will be missing.

The id is integer with 4 bytes, and the others are all float with 8 bytes.

For non-incremental `wholePointCloud`(the `incremental` property is `false` of its props), the complete data may be divided into`totalPacks` packages, and `packID` denotes the current package index. Developers should gather all the packages before processing the whole data.

For `currentMarker`, `frameMarkerPoint` and `wholeMarkerPoint`, the structures on the shared memory starting at offset `offset` are:

For `meshData`, since the data is too large, it is sent via 4 sequential units, each of which may consist of several packages (`totalPacks` is larger than 1):

Unit 1: Vertices unit.

If `pointCount` is non-zero, this unit delivers the vertices. The structures on the shared memory starting at offset `offset` are almost the same as point cloud except the absence of `id`:
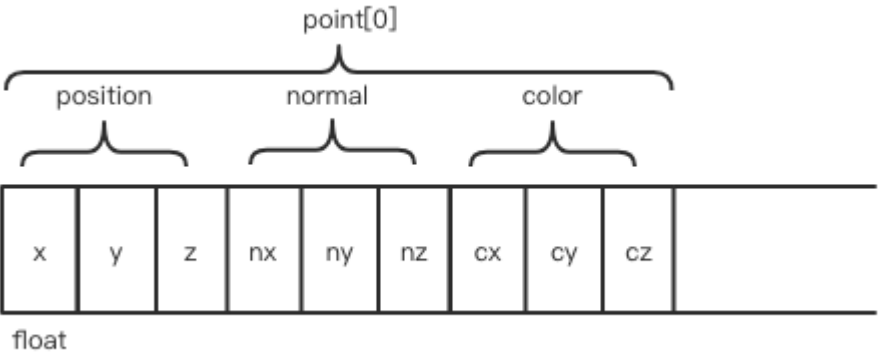
If `hasNormal` is `false`, then the normal part will be missing. If `hasTexture` is false, then the color part will be missing. If `incremental` is false, the id part will be missing.

Unit 2: Texture image unit.

If `hasTexturePicture` is `true`, this unit delivers the texture image. The image has `textureImgWidth *` `textureImgHeight` pixels, each of which is 3 bytes of RGB.

Unit 3: Triangles unit.

If `faceCount` is non-zero, this unit delivers the indexed triangles. The structures on the shared memory starting at offset `offset` are:
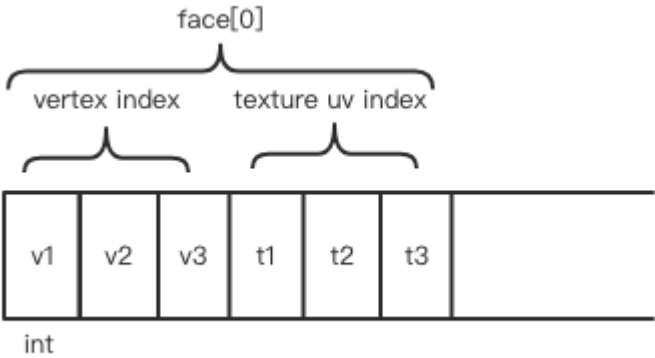
If `hasTexturePicture` is false, then the part of texture uv index will be absent.

Unit 4: Texture UV unit.

If `textureUVCount` is non-zero, this unit delivers the texture UV coordinates. The structures on the shared memory starting at offset `offset` are:
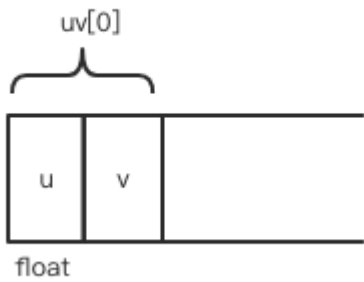
After receving the 4 units, developers can put them togather and analyze/render the triangle mesh.

For `rangeData`, the structures on the shared memory starting at offset `offset` are:
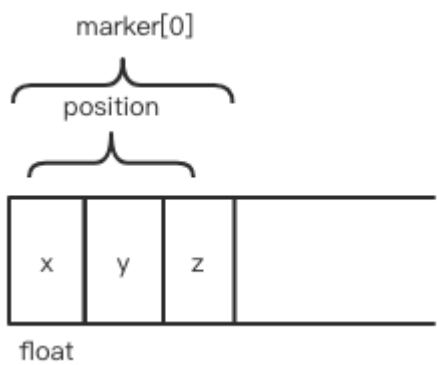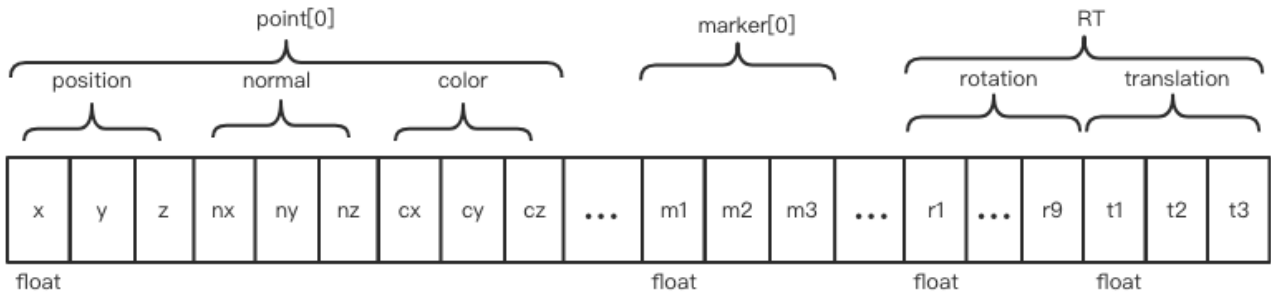

If `hasNormal` is `false`, then the normal part will be missing. If `hasTexture` is false, then the color part will be missing. If `incremental` is false, the id part will be missing.

`rotation` is a 3x3 column-major matrix, and `translation` is a 3-element vector.

Video data

For `MT_VIDEO_DATA`, the props definition is:

```
{
    "width": 512,  // Image width
    "height": 512, // Image height
    "rotation": 90,// The rotation angle (in degree)
    "channel": 1   // The channel number for each pixel
}
```

If `channel` is 1, it means that it is a grey image; if `channel` is 3, it means that it is a colorful image coming from the texture camera, and each pixel contains RGB 3 bytes.

There are 4 different named shared memories currently:

- `"cam0"`: The 1st camera.
- `"cam1"`: The 2nd camera.
- `"cam2"`: The 3rd camera.
- `"cam3"`: The 4th camera.

To render the images correctly, developers should rotate the images counterclockwise with the degree of `rotation`.

Device type

The type of your connected camera. For 2X Series, there three types:

- 2X.
- 2X Plus.
- EP.

Type	Envelop	Payload
Publish	v1.0/device/type	String
Request Get	v1.0/device/type	REQ: None REP: String

Device check

Check the hardware environment to determine whether the GPU and USB are good for the camera.

Type	Envelop	Payload
Request Get	v1.0/device/check	REQ: None REP: Int Bool

The reply of request set denotes whether the action is successful.

Asynchronous signals will be emitted.

The beginning `props` is empty.

The finishing `props` has the following definition:

```
{
    "GPU": true,// true: OK false: error
    "USB": true // true: OK false: error
}
```

There is no `progress` signal.

Infrared radiation

Whether the device has infrared radiation camera.

Type	Envelop	Payload
Publish	v1.0/device/haslr	Int Bool
Request Get	v1.0/device/haslr	REQ: None REP: Int Bool

Texture/Color camera

Whether the device has a texture/color camera.

Type	Envelop	Payload
Publish	v1.0/device/hasColor	Int Bool
Request Get	v1.0/device/hasColor	REQ: None REP: Int Bool

Discovery

Whether the device has a discovery module.

Type	Envelop	Payload
Publish	v1.0/device/hasDiscovery	Int Bool
Request Get	v1.0/device/hasDiscovery	REQ:None REP: Int Bool

Firmware upgradable

Whether there is a new version of firmware for your device.

Type	Envelop	Payload
Publish	v1.0/devce/firmwareUpgradable	Int Bool
Request Get	v1.0/device/firmwareUpgradable	REQ: None REP: Int Bool

PLE

Current PLE of your device.

Type	Envelop	Payload
Publish	v1.0/device/ple	String
Request Get	v1.0/device/ple	REQ: None REP: String

Device status

The current status of the device. There are two status currently:

- `"DS_OFFLINE"`: The device is not connected to your computer or its power is off.
- `"DS_ONLINE"`: The device is well connected.

Type	Envelop	Payload
Publish	v1.0/device/status	String
Request Get	v1.0/device/status	REQ: None REP: String

Device events

The 2X series camera carries several buttons. This interface broadcasts the button events. There are four events currently:

- `"DE_DOUBLECLICK"`: The start button is double clicked.
- `"DE_CLICK"`: The start button is clicked once.
- `"DE_PLUS"`: The plus button is clicked.
- `"DE_SUB"`: The sub button is clicked.

Type	Envelop	Payload
Publish	v1.0/devce/event	String

Calibration time

Get the last (or current used) calibration time, using the epoch of seconds starting from Jan 1st, 1970.

Type	Envelop	Payload
Publish	v1.0/cali/time	Int LL
Request Get	v1.0/cali/time	REQ: None REP: Int LL

Snap enabled

Whether to start the calibration capturing.

Type	Envelop	Payload
Publish	v1.0/cali/snapEnabled	Int Bool
Request Get	v1.0/cali/snapEnabled	REQ: None REP: Int Bool
Request Set	v1.0/cali/snapEnabled/set	REQ: Int Bool REP: Int Bool

The reply of request set denotes whether the setting action is successful.

Calibration type

Get or set the current calibration type. There are 4 different calibration type at most currently:

- `"CT_STEREO"`: The stereo calibration, which is the most important and necessary calibration for normal usage.
- `"CT_HD"`: HD calibration.
- `"CT_WHITE_BALANCE"`: White balance calibration.
- `"CT_DEFINITION"`: The accuracy definition calibration.

Type	Envelop	Payload
Publish	v1.0/cali/type	String
Request Get	v1.0/cali/type	REQ: None REP: String
Request Set	v1.0/cali/type/set	REQ: String REP: Int Bool

The reply of request set denotes whether the action is successful.

Asynchronous signals will be emitted.

The beginning `props` has the following definition:

```
{
    "type": "CT_STEREO" // the current calibration type
}
```

The finishing `props` has the following definition:

```
{
    "type": "CT_STEREO" // the current calibration type
}
```

There is no `progress` signal.

Calibration group

Get or set the current calibration group for the current calibration type. For example, there 5 groups for stereo calibration, all of which need snapping.

Type	Envelop	Payload
Publish	v1.0/cali/currentCaliGroup	Int
Request Get	v1.0/cali/currentCaliGroup	REQ: None REP: Int
Request Set	v1.0/cali/currentCaliGroup/set	REQ: Int REP: Int Bool

The reply of request set denotes whether the action is successful.

Calibration distance

Get the current calibration distance. Note: it is not the real distance between the camera and the calibration board, but it's the index of the distance gauges (it's better to refer to the calibration UI of EXScan Pro®.)

Type	Envelop	Payload
Publish	v1.0/cali/currentCaliDist	Int
Request Get	v1.0/cali/currentCaliDist	REQ:None REP: Int

Calibration distance states

Get the current calibration distance states for the current calibration group. It's better to refer to the calibration UI of EXScan Pro®.

Type	Envelop	Payload
Publish	v1.0/cali/caliDistStates	JSON
Request Get	v1.0/cali/caliDistStates	REQ:None REP: JSON

The JSON definition is as below:

```
{
    "states": [true, true, false, false, true]
}
```

Note: The number of elements of `states` depends on the corresponding calibration type.

Scan type

Get or set the current scanning type. There are 3 different types currently:

- `"ST_FIXED"`: Fix mode scanning.
- `"ST_HD"`: HD scanning.
- `"ST_RAPID"`: Rapid mode scanning.

Type	Envelop	Payload
Publish	v1.0/scan/type	String
Request Get	v1.0/scan/type	REQ: None REP: String
Request Set	v1.0/scan/type/set	REQ: String REP: Int Bool

The reply of request set denotes whether the action is successful.

Scan sub-type

Get or set scan sub-type. Some scanning type has sub types. There are 2 different types currently:

- `"SST_FIXED_FREE"`: Free style for fix scanning.
- `"SST_FIXED_TURNABLE"`: Turnable style for fix scanning.

Type	Envelop	Payload
Publish	v1.0/scan/subType	String
Request Get	v1.0/scan/subType	REQ: None REP: String
Request Set	v1.0/scan/subType/set	REQ: String REP: Int Bool

The reply of request set denotes whether the action is successful.

Path of last saved project

Get the absolute path of last saved project.

Type	Envelop	Payload
Publish	v1.0/scan/lastProjectSavePath	String
Request Get	v1.0/scan/lastProjectSavePath	REQ: None REP: String

Path of current project

Get the absolute path of current project.

Type	Envelop	Payload
Publish	v1.0/scan/currentProjectPath	String
Request Get	v1.0/scan/currentProjectPath	REQ: None REP: String

Has turnable

Get or set whether this fix scanning is connected to a turnable.

Type	Envelop	Payload
Publish	v1.0/scan/hasTurnable	Int Bool
Request Get	v1.0/scan/hasTurnable	REQ: None REP: Int Bool
Request Set	v1.0/scan/hasTurnable/set	REQ: Int Bool REP: Int Bool

The reply of request set denotes whether the action is successful.

Scan with texture

Get or set whether this scanning is with texture. It can only be enabled when a texture camera module is equipped.

Type	Envelop	Payload
Publish	v1.0/scan/withTexture	Int
Request Get	v1.0/scan/withTexture	REQ: None REP: Int
Request Set	v1.0/scan/withTexture/set	REQ: Int Bool REP: Int Bool

The reply of request set denotes whether the action is successful.

Point distance range

Get the point distance (i.e. resolution) range.

Type	Envelop	Payload
Publish	v1.0/scan/pointDistRange	JSON
Request Get	v1.0/scan/pointDistRange	REQ:None REP: JSON

The JSON definition is below:

```
{
    "min": 0.2,
    "max": 1.0
}
```

Predefined resolution values

Get the predefined resolution values (i.e. point distances).

Type	Envelop	Payload
Publish	v1.0/scan/resolutionValues	JSON
Request Get	v1.0/scan/resolutionValues	REQ:None REP: JSON

The JSON definition is below:

```
{
    "high": 1.0,
    "mid": 0.5,
    "low": 1.0
}
```

Framerate

Get the framerate.

Type	Envelop	Payload
Publish	v1.0/scan/framerate	Int
Request Get	v1.0/scan/framerate	REQ:None REP: Int

Point count

Get the total point count while scanning.

Type	Envelop	Payload
Publish	v1.0/scan/pointCount	Int
Request Get	v1.0/scan/pointCount	REQ:None REP: Int

Marker count

Get the count of markers while scanning.

Type	Envelop	Payload
Publish	v1.0/scan/markerCount	Int
Request Get	v1.0/scan/markerCount	REQ:None REP: Int

Triangle count in fix mode

Get all the triangle count in fix mode scanning.

Type	Envelop	Payload
Publish	v1.0/scan/pointFaceCount	Int
Request Get	v1.0/scan/pointFaceCount	REQ:None REP: Int

Triangle count of wrapped mesh

Get the total triangle count of wrapped mesh.

Type	Envelop	Payload
Publish	v1.0/scan/triangleCount	Int
Request Get	v1.0/scan/triangleCount	REQ:None REP: Int

Frame memory

Get the speculated frame count and the memory size needed for each frame. It is emitted before the global optimization in handled rapid mode. User should make sure that current computer condition is capable to handle this.

Type	Envelop	Payload
Publish	v1.0/scan/frameMemory	JSON
Request Get	v1.0/scan/frameMemory	REQ:None REP: JSON

The JSON definition is below:

```
{
    "count": 10,  // the speculated frame count
    "memory": 100 // the speculated memory size for each frame (MB)
}
```

Camera position

Get the camera position while scanning. The position is represented using look at manner.

Type	Envelop	Payload

Type	Envelop	Payload
Publish	v1.0/scan/cameraPosition	JSON
Request Get	v1.0/scan/cameraPosition	REQ:None REP: JSON

The JSON definition is as below:

```
{
    "center": [0, 0, 0],
    "position": [1, 0, 0],
    "up": [0, 0, 1],
    "boundBox": false // if true, the renderer needs to calculate the RT
according to the
                      // scene data while keeping the center unchanged
(effectively
                      // ignoring the position and up). Usually in fix
mode
}
```

Scan with global markers

Whether it is using global markers for scanning.

Type	Envelop	Payload
Publish	v1.0/scan/globalMarker	Int Bool
Request Get	v1.0/scan/globalMarker	REQ: None REP: Int Bool

Current brightness

Get or set the current scanning brightness. The value will be changed according to different scanning types.

Type	Envelop	Payload
Publish	v1.0/scan/currentBrightness	Int
Request Get	v1.0/scan/currentBrightness	REQ: None REP: Int
Request Set	v1.0/scan/currentBrightness/set	REQ: Int REP: Int Bool

The reply of request set denotes whether the action is successful.

Brightness range

Get the range for brightness that can be set on the camera.

Type	Envelop	Payload
Publish	v1.0/scan/brightnessRange	JSON

Type	Envelop	Payload
Request Get	v1.0/scan/brightnessRange	REQ:None REP: JSON

The JSON definition is below:

```
{
    "min": 0,
    "max": 10
}
```

Current point count

Get the current point count for this scanning.

Type	Envelop	Payload
Publish	v1.0/scan/pointCount	Int
Request Get	v1.0/scan/pointCount	REQ:None REP: Int

Point count in wrapped mesh

Get the point count in the wrapped mesh.

Type	Envelop	Payload
Publish	v1.0/scan/meshPointCount	Int
Request Get	v1.0/scan/meshPointcount	REQ:None REP: Int

HDR

Whether current scanning is HDR capable.

Type	Envelop	Payload
Publish	v1.0/scan/isHDR	Int Bool
Request Get	v1.0/scan/isHDR	REQ:None REP: Int Bool

Use discovery

Whether current scanning is using the discovery module.

Type	Envelop	Payload
Publish	v1.0/scan/useDiscovery	Int Bool
Request Get	v1.0/scan/useDiscovery	REQ:None REP: Int Bool

Has light box

Whether current scanning is using the light box.

[Todo] needs more explanation...

Type	Envelop	Payload
Publish	v1.0/scan/hasLightBox	Int Bool
Request Get	v1.0/scan/hasLightBox	REQ:None REP: Int Bool

Light box open

Whether the light box is opened.

Type	Envelop	Payload
Publish	v1.0/scan/lightBoxOpen	Int Bool
Request Get	v1.0/scan/lightBoxOpen	REQ:None REP: Int Bool

Has mesh data

Whether has mesh data.

[Todo] needs more explanation...

Type	Envelop	Payload
Publish	v1.0/scan/hasMeshData	Int Bool
Request Get	v1.0/scan/hasMeshData	REQ:None REP: Int Bool

Scan alignment type

Get the scan alignment type. There are 8 align types currently:

- `"AT_FEATURES"`: Use features to do alignment.
- `"AT_MARKERS"`: Use markers to do alignment.
- `"AT_HYBRID"`: For each frame, the device automatically choose features or markers to do alignment.
- `"AT_AUTO"`: The device automatically choose features or markers according to the very first frame, then it uses this type to do all the alignment.
- `"AT_TURTABLE"`: Use the turnable axis to do alignment in fix mode. Note: it's TUR**T**ABLE not TUR**N**ABLE. It's a historical typo, which should be hopefully corrected in the next version.
- `"AT_CODE_POINT"`: Use special code markers to do alignment.
- `"AT_GLOBAL_POINT"`: Use global markers to do alignment.

Type	Envelop	Payload
Publish	v1.0/scan/alignType	String
Request Get	v1.0/scan/alighType	REQ:None REP: String

Scan status

Get the current scanning status. There 5 different status currently:

- **"SS_PRE_SCAN"**: The initial status.
- **"SS_PRE_SCANNING"**: The prescanning status.
- **"SS_SCAN"**: The scanning status.
- **"SS_PAUSED"**: The scanning is paused.
- **"SS_SCAN_STOPED"**: The scanning is stopped. Note: it's STOP**ED** not STOP**PED**. It's a historical typo, which should be hopefully corrected in the next version.

Type	Envelop	Payload
Publish	v1.0/scan/status	String
Request Get	v1.0/scan/status	REQ: None REP: String

Scan distance

Get the current scanning distance. It's not the real distance but an evaluating integar with the following meaning:

- -1: Too near, the camera needs pulling away.
- 1~10: Acceptable distance.
- 100: Too far, the camera needs putting closer.

Type	Envelop	Payload
Publish	v1.0/scan/dist	Int
Request Get	v1.0/scan/dist	REQ: None REP: Int

Rapid mode of EP

Whether the scanning is in rapid mode. It is only meaningful for EP.

[Todo] what is rapid mode...

Type	Envelop	Payload
Publish	v1.0/scan/rapidMode	Int Bool
Request Get	v1.0/scan/rapidMode	REQ: None REP: Int Bool

Rapid save of EP

Whether it is capable of rapid saving. It is only meaningful for EP.

[Todo] what is rapid save...

Type	Envelop	Payload
Publish	v1.0/scan/rapidSave	Int Bool

Type	Envelop	Payload
Request Get	v1.0/scan/rapidSave	REQ: None REP: Int Bool

Create new project

Create new project with necessary parameters.

Type	Envelop	Payload
Request Get	v1.0/scan/newProject	REQ: JSON REP: Int Bool

The reply of request set denotes whether the action is successful.

The request JSON definition is below:

```json
{
    "path": "c:/a",              // Where to create the project
    "globalMarkerFile": "c:/b.p3",// The absolute global marker file path
if used
    "textureEnabled": false,     // Whether texture is enabled
    "pointDist": 0.5,            // The point distance/definition
    "alignType": "AT_FEATURES",  // The alignment type
    "rapidMode": false,          // Whether it's EP rapid mode
    "faseSave": false            // Whether it's EP fast save mode
}
```

Note: the project can only be created **after** entering a certain type of scanning. So developers should call entering scan before calling this interface.

Asynchronous signals will be emitted.

The beginning `props` and finishing `props` are both empty, and there is no `progress` signal.

Open project

Open an existing project.

Type	Envelop	Payload
Request Get	v1.0/scan/openProject	REQ: String REP: Int Bool

The request payload is the absolute path of the project. The reply of request set denotes whether the action is successful.

Asynchronous signals will be emitted.

The beginning `props` is empty.

The finish `props`'s definition is:

```
{
    "pointCount": 1000,// The point count of the model
    "hasTexture": true // Whether this project contains texture
}
```

There is no `progress` signal.

No markers detected

Emitted when there are no markers detected for current scanning.

Type	Envelop	Payload
Publish	v1.0/scan/noMarkerDetected	Int Bool
Request Get	v1.0/scan/noMarkerDetected	REQ: None REP: Int Bool

Too flat

Emitted when current model being scanned is too flat to do alignment.

Type	Envelop	Payload
Publish	v1.0/scan/tooFlat	Int Bool
Request Get	v1.0/scan/tooFlat	REQ: None REP: Int Bool

Track lost

Emitted when the tracking is lost for current scanning.

Type	Envelop	Payload
Publish	v1.0/scan/trackLost	Int Bool
Request Get	v1.0/scan/trackLost	REQ: None REP: Int Bool

Last mesh type

Get the mesh type of last wrapping operation. There are 2 different mesh types currently:

- `"MT_NON_WATERTIGHT"`: Non watertight mesh, it means the algorithm won't try to fill very large holes while wrapping and the model will remain open.
- `"MT_WATERTIGHT"`: Watertight mesh, means it algorithm will try to fill every hole while wrapping.

Type	Envelop	Payload
Publish	v1.0/scan/lastMeshType	String
Request Get	v1.0/scan/lastMeshType	REQ: None REP: String

Last degree of mesh detail

Get degree of mesh detail of last saved project.

Type	Envelop	Payload
Publish	v1.0/scan/lastMeshDetail	Int
Request Get	v1.0/scan/lastMeshDetail	REQ: None REP: Int

Last simplification params

Get parameters of last simplification operation.

Type	Envelop	Payload
Publish	v1.0/scan/lastSimplifyParams	JSON
Request Get	v1.0/scan/lastSimplifyParams	REQ: None REP: JSON

The JSON definition is below:

```
{
    "faceCount": 1000,        // triangle count
    "vertexCount": 2000,      // vertex count
    "needSmoothing": true,    // whether smoothing is needed
    "needSharping": false,    // whether sharping is needed
    "highQualityExtend": true,// whether texture extending is needed
    "fillMarkerHole": true,   // whether marker holes should be filled
    "fillPlainHole": true,    // whether normal/plain holes should be
filled
    "fillHolePerimeter": 3.0, // perimeter for plain holes to be filled
    "simplifyRatio": 10       // 0 - 100
}
```

Last resize params for saving

Get parameters of resize in last saving operation.

Type	Envelop	Payload
Publish	v1.0/scan/lastSaveResizeParams	JSON
Request Get	v1.0/scan/lastSaveResizeParams	REQ: None REP: JSON

The JSON definition is below:

```
{
    "dimensions": [1.0, 1.0, 1.0],// dimensions for x, y, z
    "vertexCount": 1000,          // vertex count
    ""
}
```