EventStream Client SDK 1.0

The EventStream Client SDK implements the EventStream command protocol that is needed to access the EventStream service of MOBOTIX cameras. This service provides access to live AV streams as well as to recordings stored on the camera. In addition, the EventStream service includes a notification mechanism that allows the camera to send event notifications to the client. The entire communication is handled through a single TCP connection.

1. EventStream Command Channel

1.1. Connection

The connection to the EventStream service is initiated through an HTTP-POST request. This establishes a bidirectional communication channel that is used for all further communication with the camera.

1.2. Requests and Responses

The communication with the EventStream service is based on the EventStream command protocol. The protocol uses JSON-RPC (http://json-rpc.org) as message format.

Requests can be seen as remote procedure calls. The basic structure of such a request consists of a JSON object with these properties:

- **method:** A String specifying the function name
- **params:** An Array that contains the parameters for the function call. The array can be empty if the function takes no parameters.
- **id:** An Integer that identifies the request.

The EventStream Client SDK provides a high-level API to these function calls. In your application, you only need to invoke C++ methods and retrieve the request ID once the remote procedure has been called. The camera will send a response message – including the ID - to such an RPC once it has executed the requested operation.

A response message consists of a JSON object with these properties:

- **result:** The result String of the function call on the camera. The actual content of the string depends on the function that is called. The result is null if an error occurred.
- **error:** An Array that contains an error number and an error text message. For example: „error“:[0, „invalid value“]. If no error occurred, the error property will be null.
- **type:** Optional. A String specifying the type of the response message. Possible types are:
  - **split:** The response has been split into several response messages.
    Please note: Split mode is currently not supported by the Client SDK!
  - **cont:** Continuous mode. The client should expect to get multiple responses associated with the same request ID. This mode is used for event subscriptions.
  - **info:** Status information from the camera.
- **id:** The ID that was included in the request that triggered the response.

The EventStream Client SDK currently only forwards these response messages together with the ID to the applications ”NotificationHandler” object.

1.3. Configmode

Some commands include an optional parameter named “configmode”. This parameter controls if the command should affect only the current session (**configmode = false**) or if the camera can be reconfigured to execute the command (**configmode = true**).

If the camera is reconfigured because of such an RPC, this will have an effect on the general operation of the camera. The configuration change will persist even after the session terminates.

Some operations will only work if **configmode** is activated (i.e., configmode was set to true).

For example, to change the active sensor on a dual camera, the live stream generation needs to be reconfigured. This will also affect all other clients that are displaying the live video stream. This sensor will remain active even after the session terminates.

2. Using the SDK

To use the SDK in your application, you need to implement four abstract C++ classes. Please see the header files in the “include” folder of the SDK:

- **MxPEG_SinkVideo.hpp**
  All decoded video data is passed to the doConsumeVideo() function of this object.
- **MxPEG_SinkAudio.hpp**
  All received audio data is passed to the doConsumeAudio() function of this object.
- **MxPEG_AuthorizationHandler.hpp**
  This object provides user credentials to the SDK.
- **MxPEG_EventNotificationListener.hpp**
  The notification listener object gets all response messages as raw JSON strings as they are received from the camera. Each of these strings will be associated with the ID of the request that triggered the response.

You can find example implementations for each of these abstract classes in the “samples” folder of the SDK.

3. Examples

The SDK contains two examples that demonstrate different uses cases. One shows how to receive the live AV stream from the camera. The other one additionally demonstrates basic access to recordings and playback.

Please see the ReadMe.txt file in the “doc” folder of the SDK for further information on how to build and run these examples.

3.1. Receive the live AV stream

You can find this example in the folder “samples/live-stream”.

The implementation of abstract classes are in the files:

- **AuthorizationHandler.cpp**
  Sample implementation of the MxPEG_AuthorizationHandler class. Provides the default admin user credentials admin/meinsm.
- **NotificationListener.cpp**
  Sample implementation of the MxPEG_EventNotificationListener class. Writes all received messages to stdout.
- **SinkAudio.cpp**
  Sample implementation of the MxPEG_SinkAudio class. Writes all audio data to a file named audio_stream.raw.
- **SinkVideo.cpp**
  Sample implementation of the MxPEG_SinkVideo class. Writes the decoded raw video frames to a file named video_stream.raw.

Please see the file main.cpp for the steps to initialize live streaming. First, instantiate the four handler objects:

```
SinkVideo *sinkVideo = new SinkVideo(„video_stream.raw“);
SinkAudio *sinkAudio = new SinkAudio(„audio_stream.raw“);
AuthorizationHandler *authHandler = new AuthorizationHandler();
NotificationListener *notificationListener = new NotificationListener();
```

Then pass these objects to the constructor of the MxPEG_SDK class. Note that you also need to specify the format for the decoded video frames at this point.

```
MxPEG_SDK *client = new MxPEG_SDK(sinkAudio,sinkVideo,authHandler, notificationListener,
MxPEG_ImageMode::im_YUV);
```

Once the SDK object is created you can connect to a camera by passing the EventStream service URL to the function **stream_setup()**:

```
client->stream_setup(url);
```

This will create a new session with the EventStream service of the camera. Initially, the session opens only the control channel. To also activate audio and video streaming for the session, use:

```
client->setVideoActive(true);
client->setAudioActive(true);
```

The command **startLive()** finally requests the live stream from the camera:

```
client->startLive();
```

The SDK is single-threaded, therefore you need to invoke the SDK loop() function from your application's main loop continuously in order to send and receive data.

```
while(true) {
  client->loop();
}
```

Please note: The SDK is not thread-safe! If you want to use the SDK in a multi-threaded application, you need to protect all SDK invocations accordingly.

3.2. Playback

You can find this example in the folder "samples/player".

This example uses the SDL2 library to display the received video data. This library is not included in the SDK package and needs to be installed separately. Please see the file ReadMe.txt in the "doc" folder for further information.

The implementation of the abstract classes are in the files:

- **AuthorizationHandler.cpp**
  Sample implementation of the **MxPEG_AuthorizationHandler class**. This provides the default **admin** user credentials **admin/meinsm**.
- **NotificationListener.cpp**
  Sample implementation of the **MxPEG_EventNotificationListener** class. Writes all received messages to stdout.
- **SinkAudio.cpp**
  Sample implementation of the **MxPEG_SinkAudio** class. The implementation in this example simply deletes all data that is passed to its consume function. Since this example does not request audio data, the consume function should never be called.
- **SinkVideo.cpp**
  Uses SDL2 to create a window in which the received video frames a displayed.

The initialization is similar to the first example, only the SDL2 library initialization code is added:

```
SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO);
```

And the example only requests video frames and no audio:

```
g_client->setVideoActive(true);
g_client->setAudioActive(false);
```

In addition to calling the SDKs loop() function, this example also calls a function named poll_sdl_events().

```
while(g_active){
  g_client->loop();
  poll_sdl_events();
}
```

The function **poll_sdl_events()** uses the SDL library to get user input from the keyboard and mouse. Based on this input, the SDK functions are invoked to switch between live and playback modes, to control the player or to start a PTZ operation.

For a detailed description on the API provided by the EventStream Client SDK, please see in the file "include/MxPEG_SDK_API.hpp"