vxgproxyclient

1.0.4

Generated by Doxygen 1.8.17

Chapter 1

VXG Uplink Client Library

Chapter 2

Build System

2.0.1 Overview

VXG Uplink Client library uses `Meson` build system as a modern, fast and flexible build system that supports easy to set up and maintain a cross-compilation process.

It's recommended to refer to the `Meson` guide.

2.0.2 Build system installation

IMPORTANT: This projects requires Meson version >= 0.56.0

It's recommended to use `Ubuntu 20.04 LTS` distribution in development process but other distributions or operation systems are also supported by `Meson`.

Please refer to `Meson installation guide` to get and install `Meson`, preferable way to install `Meson` is `pip` method.

Quick install guide for Ubuntu 20.04. If you have an old version of meson already installed please remove it first.

```
sudo apt-get update
sudo apt-get install -y python3-pip git ninja-build curl tzdata python3-tz
pip3 install git+https://github.com/mesonbuild/meson@0.56.0
# pip3 puts meson main script into the $HOME/.local/bin/ directory, you need to
# add $HOME/.local/bin/ into your PATH environment variable, for bash shell you
# can run the following command and restart the shell session.
echo 'export PATH=$HOME/.local/bin:$PATH' » $HOME/.bashrc
# Check currently installed meson version
meson -v
```

Chapter 3

Application Development

3.1 Overview

An application that uses VXG Uplink Client Library should implement the Uplink::Proxy class derived from the base classes provided by the library:

- Uplink::Proxy - common implementation class, used for obtaining camera information such as serial number and MAC Address.

Any Proxy implementation should implement the get_serial_number, get_mac_address, and get_camera_info functions.

The library provides the stub implementation for most of the virtual methods of these classes, the stub implementation prints a log message about this method is not implemented and returns an error, the final application should implement all virtual methods on its own.

3.1.1 Linking application against the VXG Uplink Client Library

There are 3 possible ways of how to build and link your application

1. Building the application inside the VXG Uplink Client library's `Meson project`, the app will be assembled during the library project compilation in this case.
   You need to add a new executable target into the main `meson.build` file, please refer to the example app build target declaration:

   User must declare own executable target with a list of sources and dependencies, user may need to declare own dependencies if application requires it.

   This method is not recommended as it makes updating of the VXG Uplink Client library mostly not possible or very difficult for application developer

2. Building your app using your own build system and linking against the installed library.
   Running the `install` step from the compile section installs the binary libraries and headers into the directory you specified during the `setup` step, it also puts the `pkg-config`'s `.pc` files into the prefix directory which could be used by your own build system.

3. Preferred and recommended way of application development is to hold the app as a separate `Meson project` and use the VXG Uplink Client library as a `Meson subproject` of the application's `Meson project`.
   Using this approach gives the most flexible and convenient workflow for updating the VXG Uplink Library, all library dependencies will be promoted to the main project and will be also accessible by the application.
   How does it work

   - Assuming you have a `Meson` build system installed
   - Start a new `Meson project` with a following command:
     ```
     meson init -l cpp -n your-project-name
     ```
   - As a result of this command you should have the following files tree:
     ```
     |-- meson.build
     |-- your_project_name.cpp
     ```
   - Add VXG Uplink Client library as a `Meson subproject`
     All subprojects should be located in the `subprojects` directory so you have to create it first
     ```
     mkdir subprojects
     ```

   Now you have 2 options depending on how you want to store the VXG Uplink Client library sources:

   (a) If you want to store the VXG Uplink Client library as a files tree locally.

   - Create a symlink to the library path inside the subprojects dir:
     ```
     ln -s path/to/vxgproxyclient subprojects/vxgproxyclient
     ```

   Or you can just move vxgproxyclient directory inside the subprojects dir.

   - Create a library's `Meson` wrap file inside the subprojects dir, the name of the file should be the same as symlink you created in 1.1 and the content of the file should be:
     ```
     [wrap-file]
     directory = vxgproxyclient
     [provide]
     vxgproxyclient = vxgproxyclient_dep
     ```

   (b) If you want to store the library in a git repository you just need to create a wrap file with the content like below:
   ```
   [wrap-git]
   url=https://your-git-repo-url.com/path/vxgproxyclient.git
   # You can specify tag, branch or commit hash as revision
   revision=master
   [provide]
   vxgproxyclient = vxgproxyclient_dep
   ```

   You can find the example app `Meson project` in the example/app directory of the VXG Uplink library sources package.

Chapter 4

Library Compilation Guide

4.0.1 Library build process

Here is a compilation quickstart guide:

- First of all you need to have a build system and toolchain installed
- **Setup** the build directory
```
meson setup --prefix=path/to/install --strip -Dbuildtype=debug builddir/
# --prefix=path specifies the installation path
# --strip indicates that final binaries should be stripped
# -Dbuildtype= specifies the debug/release build type, please check the Meson docs about full list of
 the build types.
```
- **Build**
```
meson compile -C builddir
# Or
ninja -C builddir
```
- **Install**
```
meson install -C builddir
# Or
ninja -C builddir/ install
```
  As a result of the `install` step you should have the library compiled and installed into the prefix directory you specified during the `setup` step.
- **Clean**
```
ninja -C builddir clean
```
  Or you can just delete the builddir, you will need to `setup` it again in this case.
```
rm -rf builddir
```

4.0.2 Cross-compilation

- By default `Meson` builds project for the host platform, but it's also possible to cross-compile the library and your application using `Meson`.
- Full `Meson` cross-compilation documentation can be found `here`.
- The difference between the host compilation described above and the cross-compilation is the additional `--cross-file=path/to/cross-file.txt` flag for the Meson `Setup` step, the `Setup` command should look like below:
```
meson setup --prefix=path/to/install --strip -Dbuildtype=debug --cross-file=path/to/cross-file.txt
 builddir/
```
  `cross-file.txt` is the target platform description which in terms of Meson called a `cross-file`.
- `cross-file` example below is for the Debian provided `arm-linux-gnueabihf` toolchain installable using the Ubuntu's package manager command
```
sudo apt install g++-arm-linux-gnueabihf
```
- Example of the ARMv7 `cross-file` can be found in `/cross` directory:

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 6

Data Structure Index

6.1 Data Structures

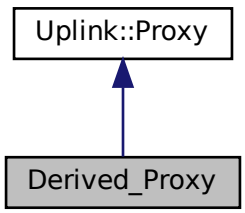
Here are the data structures with brief descriptions:

Chapter 7

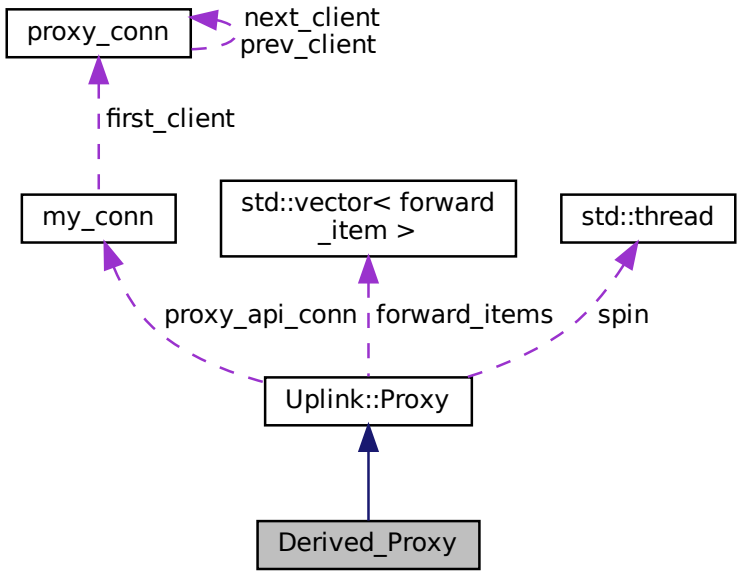
Data Structure Documentation

7.1 Derived_Proxy Class Reference

Inheritance diagram for Derived_Proxy:



Collaboration diagram for Derived_Proxy:



Public Member Functions

- int get_serial_number (char *ser_number) override

  [Get serial number implementation]
- int get_mac_address (char *mac_address) override

  [Get mac address implementation]
- int get_camera_info () override

  Get camera info function, responsible for retrieving camera S/N and MAC address.

7.1.1 Detailed Description

Definition at line 49 of file vxg_proxy_client.cc.

7.1.2 Member Function Documentation

7.1.2.1 get_camera_info()

```
int Derived_Proxy::get_camera_info ( )  [inline], [override], [virtual]
```

Get camera info function, responsible for retrieving camera S/N and MAC address.

Returns

0 if successful

Reimplemented from Uplink::Proxy.

Definition at line 70 of file vxg_proxy_client.cc.

The documentation for this class was generated from the following file:

- vxg_proxy_client.cc

7.2 forward_item Struct Reference

Data Fields

- char **name** [MAX_FORWARD_ITEM_NAME_LEN+1]
- char **host** [MAX_FORWARD_ITEM_HOST_LEN+1]
- Proto **proto**
- uint16_t **port**

7.2.1 Detailed Description

Definition at line 32 of file Proxy.h.

The documentation for this struct was generated from the following file:

- Proxy.h

7.3 msg Struct Reference

Data Fields

- void * **payload**
- size_t **len**

7.3.1 Detailed Description

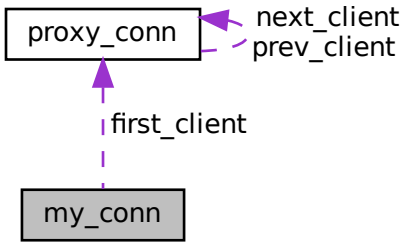
Definition at line 39 of file Proxy.h.

The documentation for this struct was generated from the following file:

- Proxy.h

7.4 my_conn Struct Reference

Collaboration diagram for my_conn:



Data Fields

- lws_sorted_usec_list_t **sul**
- struct lws ∗ **wsi**
- uint16_t **retry_count**
- struct lws_ring ∗ **ring**
- uint32_t **tail**
- char **flow_controlled**
- uint8_t **write_consume_pending**:1
- struct proxy_conn ∗ **first_client**
- uint32_t **total_msgs_in_client_rings**
- void ∗ **obj**

7.4.1 Detailed Description

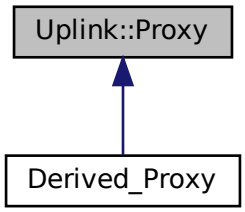
Definition at line 60 of file Proxy.h.

The documentation for this struct was generated from the following file:

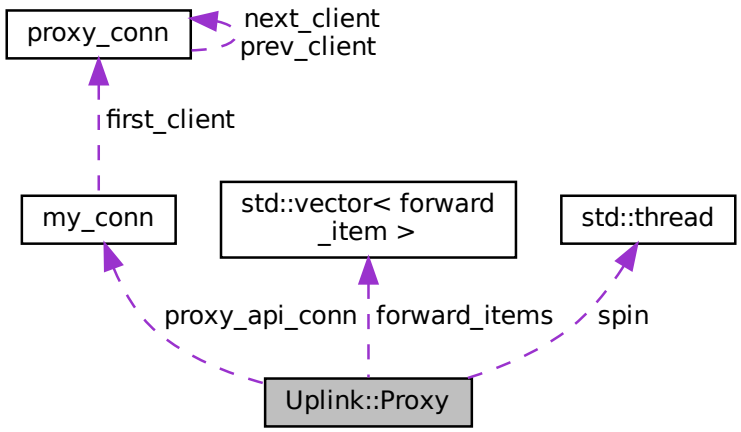
- Proxy.h

7.5 Uplink::Proxy Class Reference

Inheritance diagram for Uplink::Proxy:



Collaboration diagram for Uplink::Proxy:



Public Member Functions

- virtual int **get_serial_number** (char ∗ser_number)
- virtual int **get_mac_address** (char ∗mac_address)
- virtual int **get_camera_info** ()
- int start ()

  Start internal workflow, this is the main function which starts all internal connections.
- void stop ()

  Stop internal workflow, this is the main function which stops lws connection.
- void **set_parameters** (char ∗api_host, char ∗api_path, char ∗api_password, char ∗ws_host, char ∗ws_path, char ∗device_ser, char ∗token, int conn_port, int ssl_conn, **std::vector**< forward_item > ∗fwd_items)
- volatile int **get_force_exit** ()
- volatile int **get_restart** ()

Private Member Functions

- void **_websocket_connect** (lws_sorted_usec_list_t *sul)
- void **_authtoken_connect** (lws_sorted_usec_list_t *sul)
- void **_proxy_api_connect** (lws_sorted_usec_list_t *sul)
- void **_vxg_token_api_connect** (lws_sorted_usec_list_t *sul)
- int **_proxy_client_connection_closed_notification** (struct proxy_conn *client)
- void **_destroy_proxy_client** (struct proxy_conn *client)
- int **_proxy_client_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- int **_proxy_api_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- int **_websocket_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- int **_authtoken_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- int **_vxg_token_api_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- void **_wait_for_exit** ()

Static Private Member Functions

- static void **_destroy_message** (void *_msg)
- static void **websocket_connect** (lws_sorted_usec_list_t *sul)
- static void **proxy_api_connect** (lws_sorted_usec_list_t *sul)
- static void **vxg_token_api_connect** (lws_sorted_usec_list_t *sul)
- static int **proxy_client_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- static int **proxy_api_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- static int **websocket_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- static int **vxg_token_api_callback** (struct lws *wsi, enum lws_callback_reasons reason, void *user, void *in, size_t len)
- static void **wait_for_exit** (void *user_data)

Private Attributes

- char **vxg_api_host** [128]
- char **vxg_api_path** [128]
- char **vxg_api_password** [128]
- char **proxy_api_host** [256]
- char **proxy_api_path** [256]
- char **proxy_ws_host** [256]
- char **proxy_ws_path** [256]
- unsigned short **proxy_ws_port**
- char **device_serial** [256]
- char **auth_token** [1024 *8]
- int **port**
- int **ssl_connection**
- const char * **pro**
- const uint32_t **backoff_ms** [5]
- lws_retry_bo_t **ws_retry**

- lws_retry_bo_t **vxg_token_api_retry**
- lws_retry_bo_t **proxy_api_retry**
- int **vxg_token_api_request_status**
- char **auth_res_json_buffer** [1024 ∗10]
- int **proxy_api_request_status**
- char **proxy_api_res_json_buffer** [1024 ∗10]
- int **websocket_rcv_buffer_len**
- char ∗ **websocket_rcv_buffer**
- volatile int **force_exit**
- volatile int **restart**
- volatile int **is_provisioning**
- **std::thread spin**
- struct lws_context ∗ **context**
- **std::vector**< forward_item > **forward_items**
- struct lws ∗ **client_wsi**
- struct lws_context_creation_info **info**
- struct my_conn ws_conn authtoken_conn vxg_token_api_conn **proxy_api_conn**

7.5.1 Detailed Description

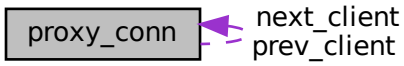
Definition at line 75 of file Proxy.h.

The documentation for this class was generated from the following files:

- Proxy.h
- Proxy.cpp

7.6 proxy_conn Struct Reference

Collaboration diagram for proxy_conn:



Data Fields

- struct lws ∗ **wsi_raw**
- struct lws_ring ∗ **ring**
- uint8_t **forward_index**
- client_id_t **client_id**
- uint32_t **tail**
- char **flow_controlled**
- char **close_notification_sent**
- uint8_t **write_consume_pending**:1
- struct proxy_conn ∗ **next_client**
- struct proxy_conn ∗ **prev_client**
- void ∗ **obj**

7.6.1 Detailed Description

Definition at line 46 of file Proxy.h.

The documentation for this struct was generated from the following file:

- Proxy.h

Index