Brightway Vision Camera API User guide

For API Version 3.1.20

Contents

1. General

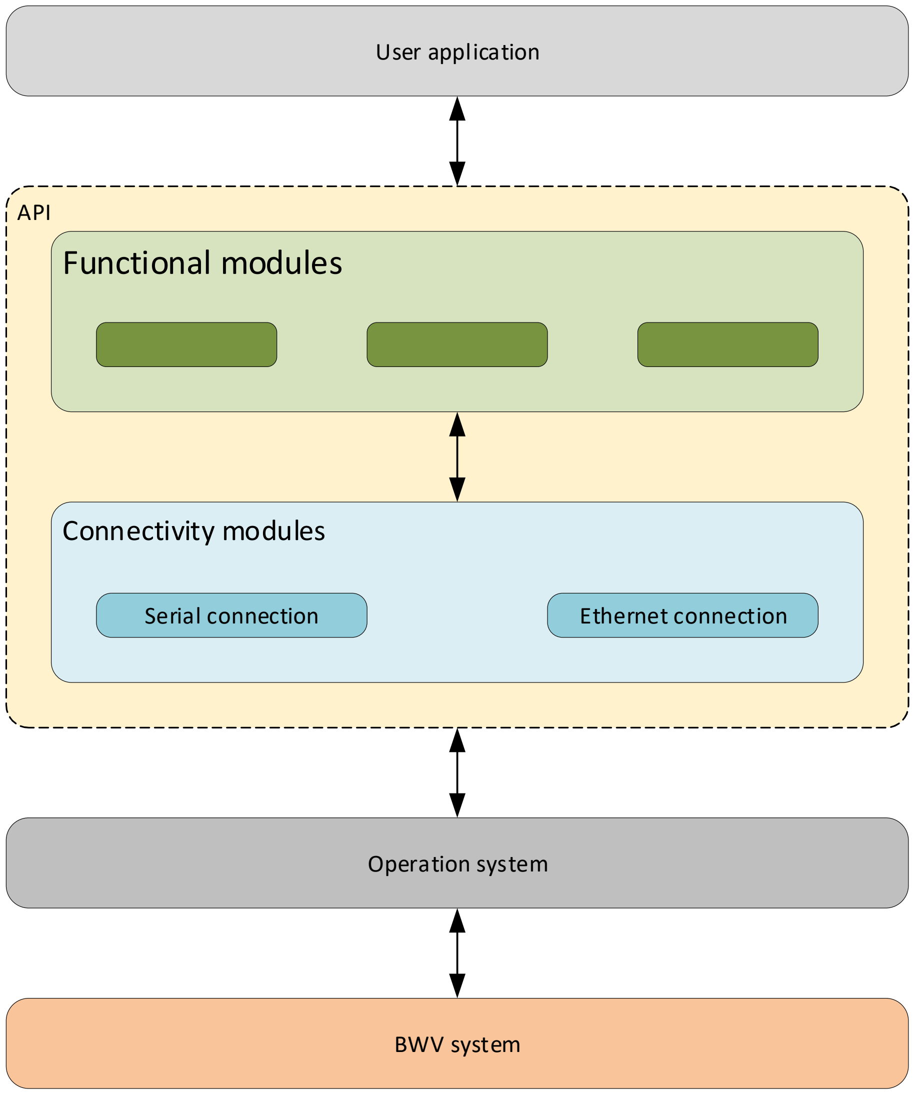
This document explains BWV API top level architecture and use scenarios.

BWV cameras connect to PC either through Communication box/Control box by means of USB-to-Serial interface or directly through ethernet connection.

BWV API is shared library and contains set of functions that provides connectivity with BWV camera system and easy access to camera/control box units. It uses threading system to maintain real time asynchronous control access.

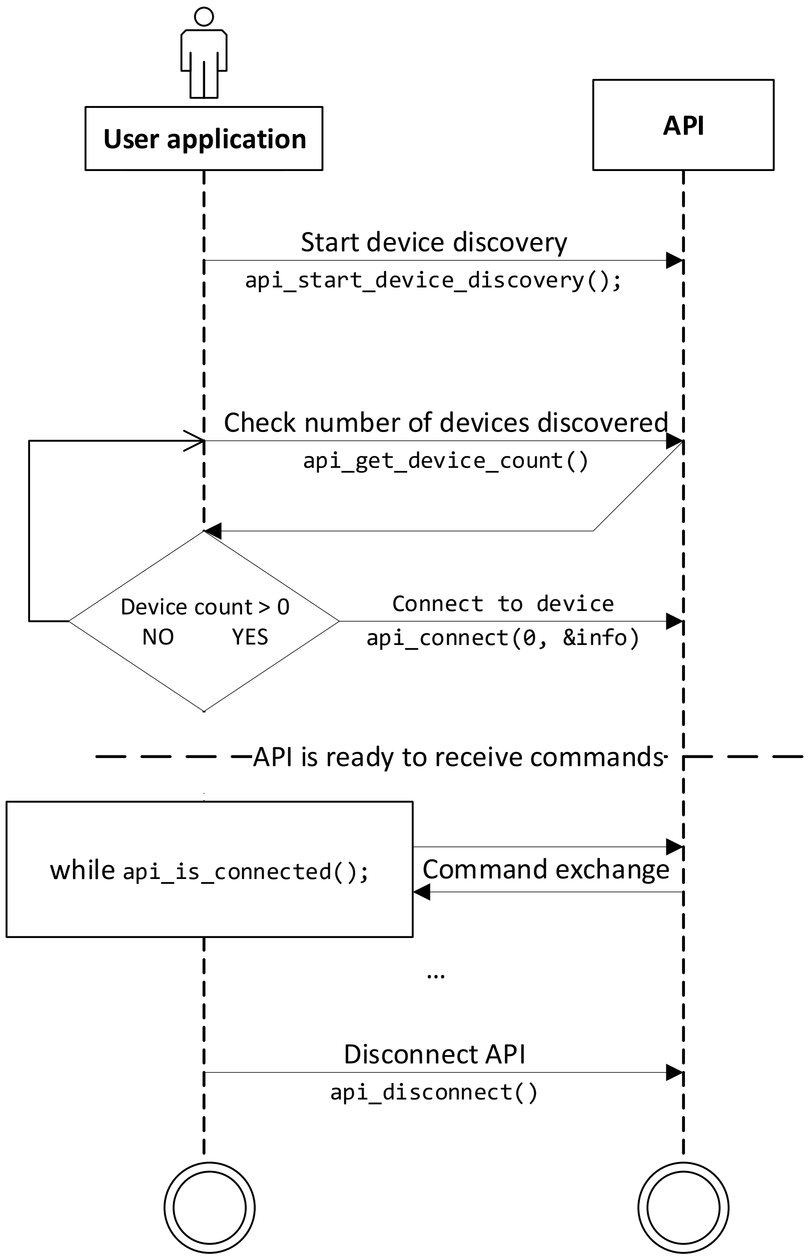
Command responses are callback-based. That means for every read operation user application should register its function as callback to API. This function will be called by API when corresponding response will be received from camera system

API contains several function modules that use connectivity subsystem for sending and receiving commands.


User application
API
Functional modules
Connectivity modules
Serial connection
Ethernet connection
Operation system
BWV system

2. User application lifecycle

User application should take into account that API is multithreaded and command responses are asynchronous. That means that user application should implement internal loop which prevent application from exiting before API could respond.



3. Connection sequence

These are the steps user application have to implement to connect API to camera system

1. Load API dynamic library
2. Call api_init() to obtain handle to API instance. This handle should be provided for all API calls
3. Call to `api_start_device_discovery()`
4. Wait till `api_get_device_count()` is non-zero
    5. Connect to desired device, providing its zero-based index (e.g. 0) using `api_connect();`
6. Starting this point further user application can exchange commands with API. It should not exit till it calls to api_disconnect(); and api_deinit();

4. Typical command workflow

In case of application wants to write data to system the flow is the following:

1) Application initialized corresponding data structure
2) Application fills required fields of structure
3) Application calls write function of corresponding functional module passing prepared data structure

In case of application wants to read data from system the flow is the following:

1) Application registers callback with “add…callback” function of corresponding module
2) Application calls read function of corresponding functional module
3) When data arrives, registered callback will be called with module data structure as parameter