April 2024

Camera Control PTP Example Instruction Manual

SONY

First edition (November 2023)

Contents

About

The following four types of example programs are provided to help your understanding and implementation of the Camera Control PTP:

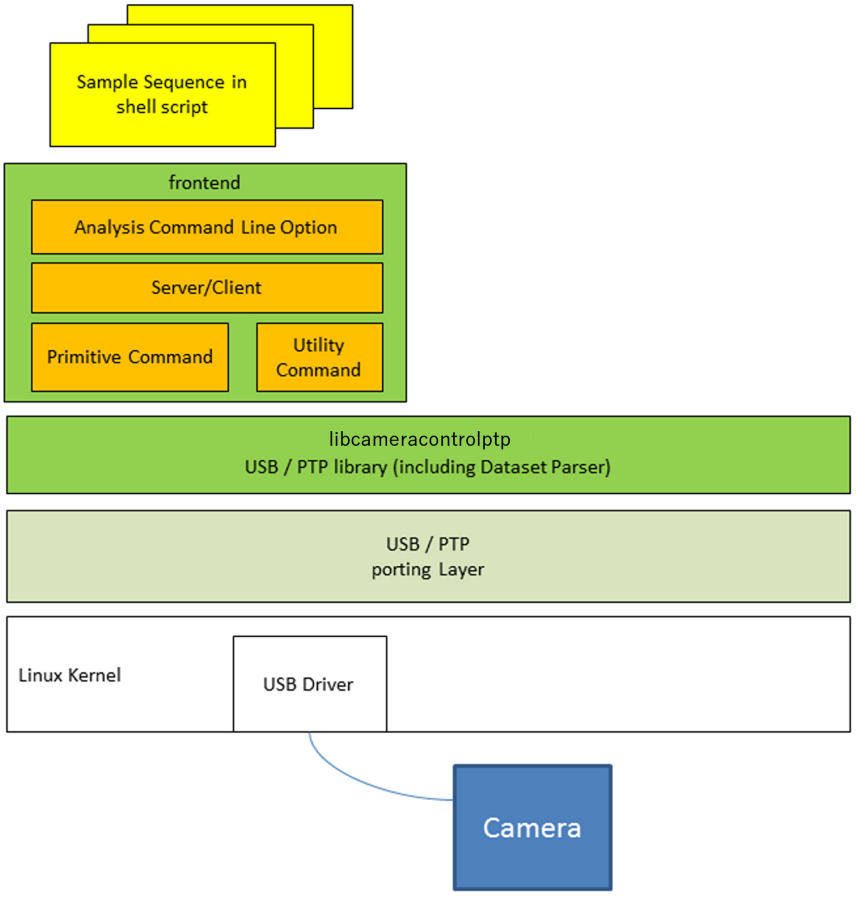
- Camera Control PTP (for 2020 Models or Later) Example Program for Linux
- Camera Control PTP (for 2020 Models or Later) Example Program for Windows
- Camera Control PTP (for Models Earlier than 2020) Example Program for Linux
- Camera Control PTP (for Models Earlier than 2020) Example Program for Windows

Note that these programs are objected to only to explain how to implement the protocol. Therefore, please do not use them in your products.

Example Program for Linux

The example program for Linux consists of:

- `libcameracontrolptp`: a library of the Camera Control PTP implementations
- `control`: a CLI program that provides primitive commands corresponding to PTP operations using the library
- CLI wrapper scripts that represent high-level scenarios (found in `frontend/scripts`)



System Requirements

This example program is tested on the following system:

- Linux distribution: Ubuntu 22.04.3 LTS
- Linux kernel: 6.2.0-34-generic x86_64 GNU/Linux
- Libraries:
  - libusb-1.0-0: 2:1.0.25-1ubuntu2
  - libusb-1.0-0-dev: 2:1.0.25-1ubuntu2
  - libc6: 2.35-0ubuntu3.4
  - libc6-dev: 2.35-0ubuntu3.4

- libstdc++6: 12.3.0-1ubuntu1~22.04
- libstdc++-11-dev: 11.4.0-1ubuntu1~22.04

How to Build

1. Unpack the sources
2. Move to the top-level directory
3. Type `make`

To run the example programs, please refer to the CLI section.

Example Library (libcameracontrolptp)

This library is an example library to perform PTP/USB connection and PTP transfer. It is used from the CLI front-end program. It is designed for libusb-1.0 as a USB back end. If your system has your USB back end or PTP back end, please use it instead. If you use libusb-1.0, please follow its license.

API Reference

If a method has a return value, it will be zero on success and non-zero on failure.

`com::sony::imaging::remote::socc_ptp::connect()`

`connect()` connects to the target device and makes it possible to start a PTP transaction. The bus number and device number of the target device should be given with the constructor's parameters. This example tries to connect automatically for the first PTP device found in the enumeration if the bus number and device number are zero.

`com::sony::imaging::remote::socc_ptp::disconnect()`

`disconnect()` should disconnect from the target device and release every resource if needed.

`com::sony::imaging::remote::socc_ptp::send(uint16_t code, uint32_t* params, uint8_t nparam, Container& response, void* data, uint32_t size)`

`send()` is a method to perform PTP transactions.

- `code`: OpperationCode in OperationRequest dataset
- `params`: uint32_t array for parameter fields in the request phase
- `nparam`: The number of parameters
- `response`: The response dataset that is received in the response phase
  `Container` is defined in `socc_types.h` as follows:

```
typedef struct _Container{
  uint16_t code;
  uint32_t session_id;
  uint32_t transaction_id;
  uint32_t param1;
  uint32_t param2;
  uint32_t param3;
  uint32_t param4;
```

```
  uint32_t param5;
  uint8_t  nparam; // Number of used parameters
} Container;
```

`send()` sends operations with/without a data phase. If the operation has a data phase, the `data` parameter should be a pointer to the buffer to send, and the `size` parameter should be the length of the buffer in bytes. If the operation has no data phase, the `data` parameter should be `NULL`, and the `size` parameter should be zero.

`com::sony::imaging::remote::socc_ptp::receive(uint16_t code, uint32_t* params, uint8_t nparam, Container& response, void** data, uint32_t& size)`

`receive()` is a method to perform PTP transactions.

Some parameters are shared with `send()`.

`receive()` receives operations with a data phase. The buffer that is needed to receive data from a device is allocated in this method. It will be disallocated by `dispose()`. The transferred size in bytes will be set in the `size` parameter on success.

`com::sony::imaging::remote::socc_ptp::wait_event(Container& container)`

`wait_event()` waits for a PTP event and copies the acquired PTP event container to the `container` parameter when your back end detects a PTP event. The format of `Container` is different from the PTP Event dataset format. Please translate each field.

`com::sony::imaging::remote::socc_ptp::dispose_data(void** data)`

`dispose_data()` should free allocated buffer by `receive()`, and set its pointer to `NULL`.

`com::sony::imaging::remote::socc_ptp::set_hotplug_callback(socc_hotplug_callback_func_t callback_func, void* vp)`

`set_hotplug_callback()` registers a callback function for USB hot plug detection. The type of callback function is `socc_hotplug_callback_func_t`, defined in `socc_types.h`. The first parameter of callback stands for USB connection status. Call this callback with `socc_hotplug_event_t::SOCC_HOTPLUG_EVENT_ARRIVED` when your back end detects a USB bus connection, `socc_hotplug_event_t::SOCC_HOTPLUG_EVENT_REMOVED` for disconnection. These enum values are defined in `socc_types.h` as a type. The second parameter of callback is user data. Set the same pointer given by `set_hotplug_callback()`. `socc_hotplug_callback_func_t` and `socc_hotplug_event_` are defined in `socc_types.h`. In this example, the library detects a hotplug event after calling `connect()` successfully, only for a device claimed to be used.

```
typedef enum {
  SOCC_HOTPLUG_EVENT_UNKNOWN = 0,
  SOCC_HOTPLUG_EVENT_ARRIVED = 1,
  SOCC_HOTPLUG_EVENT_REMOVED = 2
} socc_hotplug_event_t;
typedef void(*socc_hotplug_callback_func_t)(socc_hotplug_event_t, void*);
```

`com::sony::imaging::remote::socc_ptp::clear_halt(int what = 0)`

If your back end needs to clear the HALT/STALL status of USB endpoints from the front end, implement this method. In this example, the library clears BULK IN, BULK OUT, and Interrupt IN endpoints at the same time if the parameter `what` is zero.

`com::sony::imaging::remote::socc_ptp::reset()`

If your back end needs to reset the USB bus from the front end, implement this method. In this example, the library calls the reset function of the back end.

CLI

The CLI executes PTP commands and outputs the logs and an object (content) to a file. It is easy to confirm the camera behavior and to compare the behavior with your code because, one by one, the PTP command can be executed from a shell.

The source code parts of the CLI are described below:

- Analysis Command Line Option: `main.cpp`
- Server/Client: The server and client model is employed for keeping a USB connection. See `serverclient.h`
- Primitive and Utility command: `command.h`
- Dataset Parser: `parser.h`

Command Reference

The CLI needs to be executed in the root user for the permission of the USB device node. The environment variable `LD_LIBRARY_PATH` should contain a path to the `out/lib/` directory.

`control send --op=OperationCode [--p1=param1] [--p2=param2] [--p3=param3] [--p4=param4] [--p5=param5] [--size=size] [--data=data] [--log=logfile] [--bus=busN] [--dev=devN]`

This command is used for an Initiator-to-Responder operation. The operation consists of sending an operation code and five parameters, sending data, and receiving a response.

The communication log is appended to the end of the file specified with the parameter `--log`. The log includes the time, sent command, and response. If the parameter `--log` is set to `-` or omitted, the log will be written to the stdout.

The operation is sent to the bus number specified with the parameter `--bus` and the device number specified with the parameter `--dev`. Use the `lsusb` command to obtain these numbers. If these numbers are omitted, the CLI finds a device automatically.

If the parameter `--size` is set to `string`, this command sends the data specified with the parameter `--data` as a string.

If the parameter `--size` is set to `file`, this command recognizes the parameter `--data` as a file name and sends the file content.

If the parameters `--data` and `--size` are omitted, this command executes an operation without data.

`control recv --op=OperationCode [--p1=param1] [--p2=param2] [--p3=param3] [--p4=param4] [--p5=param5] [--of=outfile] [--log=logfile] [--bus=busN] [--dev=devN]`

This command is used for a Responder-to-Initiator operation. The operation consists of sending an operation code and five parameters, receiving data, and receiving a response. The data will be written into the file specified with the parameter `--of`. If the parameter `--of` is set to `-` or omitted, the data outputs to the stdout.

`control wait [--log=logfile] [--bus=busN] [--dev=devN]`

Waits an event. The event code and the parameters are written in the file specified with the parameter `--log`.

`control clear [--bus=busN] [--dev=devN]`

When Bulk-in and Bulk-out endpoints of the USB are stalled, recover from it by calling this. STALL occurs when sending an invalid operation (e.g., wrong operation code or parameters). If this command succeeds, commands such as `send` and `recv` can be executed.

`control reset [--bus=busN] [--dev=devN]`

When an unrecoverable error occurs, for example, a `clear` command returns an error, the device may recover by executing this command. Then, the USB connection is down, and another `open` command is needed to be executed.

`control open [--log=logfile] [--bus=busN] [--dev=devN]`

Opens a new session.

`control close [--log=logfile] [--bus=busN] [--dev=devN]`

Closes the open session.

`control auth [--log=logfile] [--bus=busN] [--dev=devN]`

Executes an authentication sequence.

`control getall [--if=infile] [--of=outfile] [--log=logfile] [--bus=busN] [--dev=devN]`

Executes the operation SDIO_GetAllExtDevicePropInfo, parses the result, and writes all data into the file specified with the parameter `--of`. If the parameter `--if` is given, this command only parses the specified file as the output file of a `getall` command. If the parameter `--if` is set to `-`, this command reads the data from the stdin instead of a file.

`control get DevicePropertyCode [--if=infile] [--of=outfile] [--log=logfile] [--bus=busN] [--dev=devN]`

Executes the operation SDIO_GetAllExtDevicePropInfo, parses the result, and writes the SDIDevicePropInfo dataset into the file specified with the parameter `--of`.

`control getobject handle [--of=outfile] [--log=logfile] [--bus=busN] [--dev=devN]`

Executes the GetObject operation for handle and writes the object data into the file specified with the parameter `--of`.

`control getliveview [--of=outfile] [--log=logfile] [--bus=busN] [--dev=devN]`
Obtains a live-view image and saves it to the file specified with the parameter `--of`.

Example Program for Windows

The example program for Windows is a GUI-based program that can control a camera using the Camera Control PTP.

System Requirements

This example program is tested on the following system:

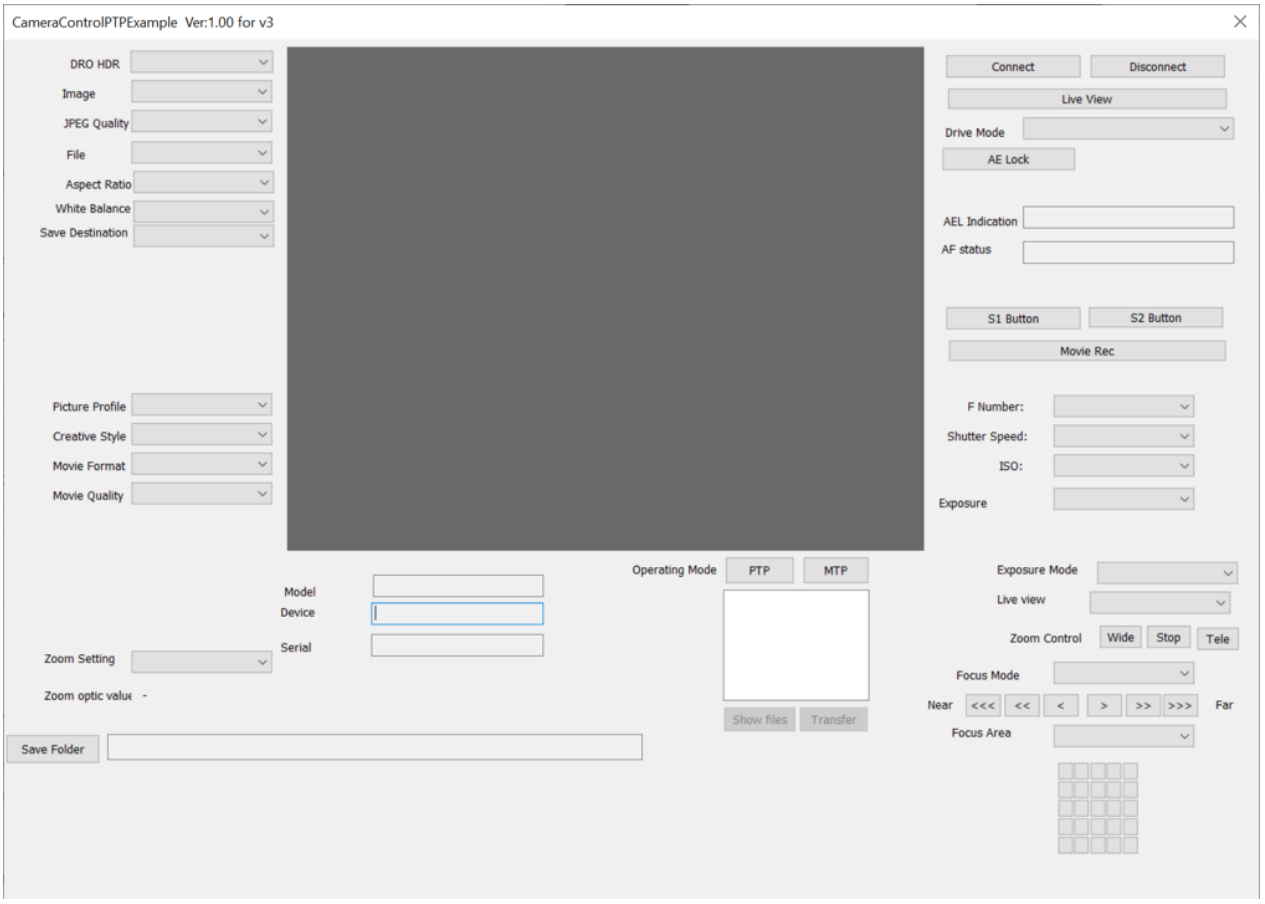
- Microsoft Windows 10
- Microsoft Visual Studio 2022
- Microsoft Windows SDK Version 10.0

How to Build

1. Open the solution file `CameraControlPTP.sln`
2. Execute the build command using the IDE
3. The built application will be located in `Release\CameraControlPTP.exe`

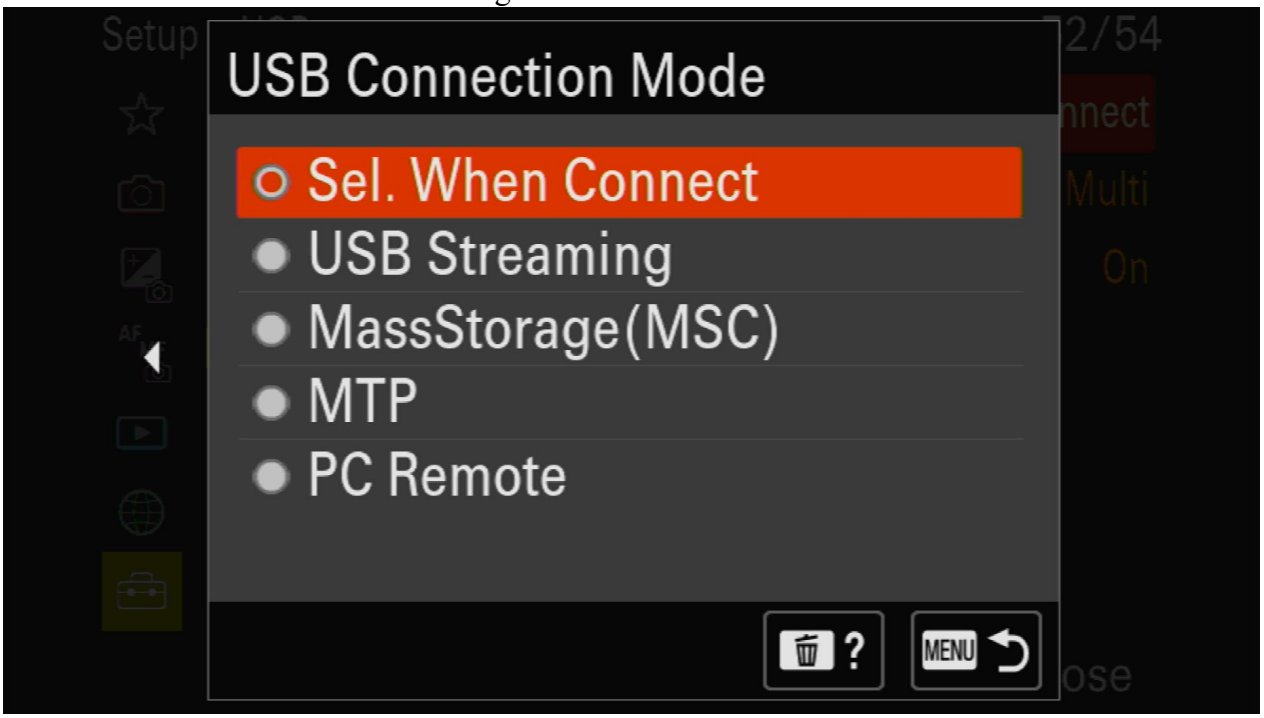
Usage

This example program has a GUI below:



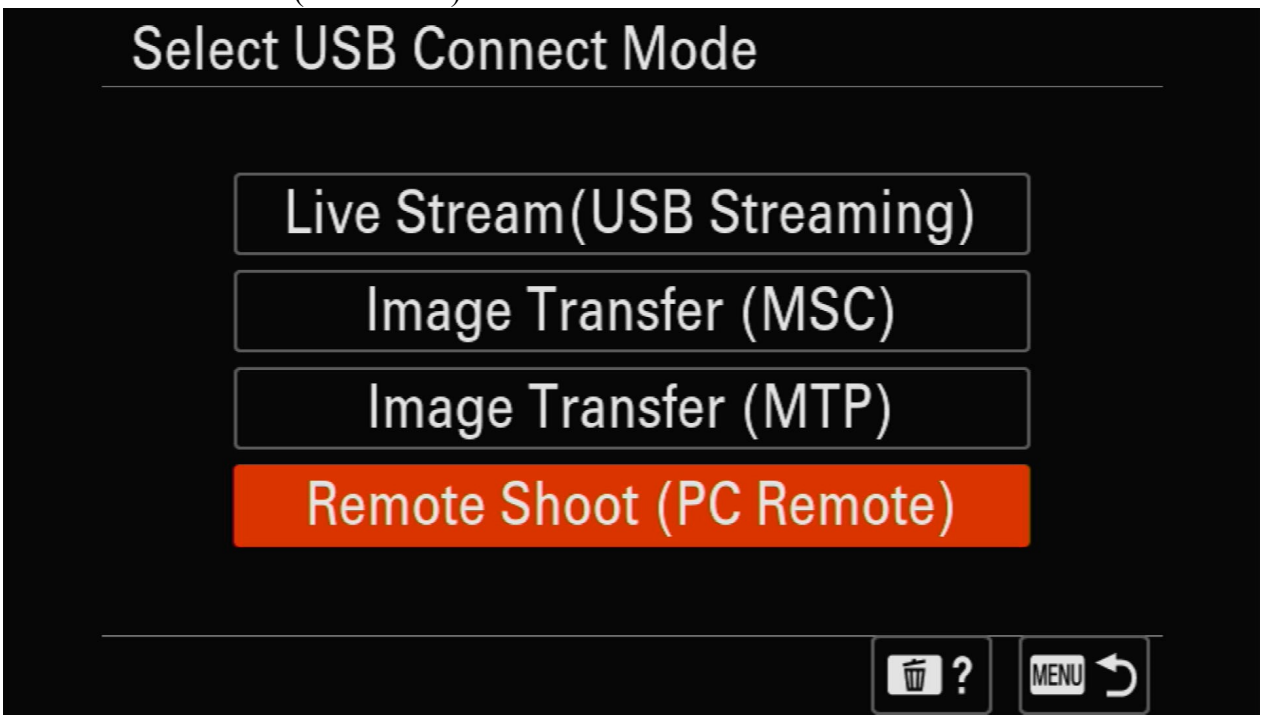
Connect Camera to PC

1. Select “Sel. When Connect” in “Setting > USB > USB Connection Mode”

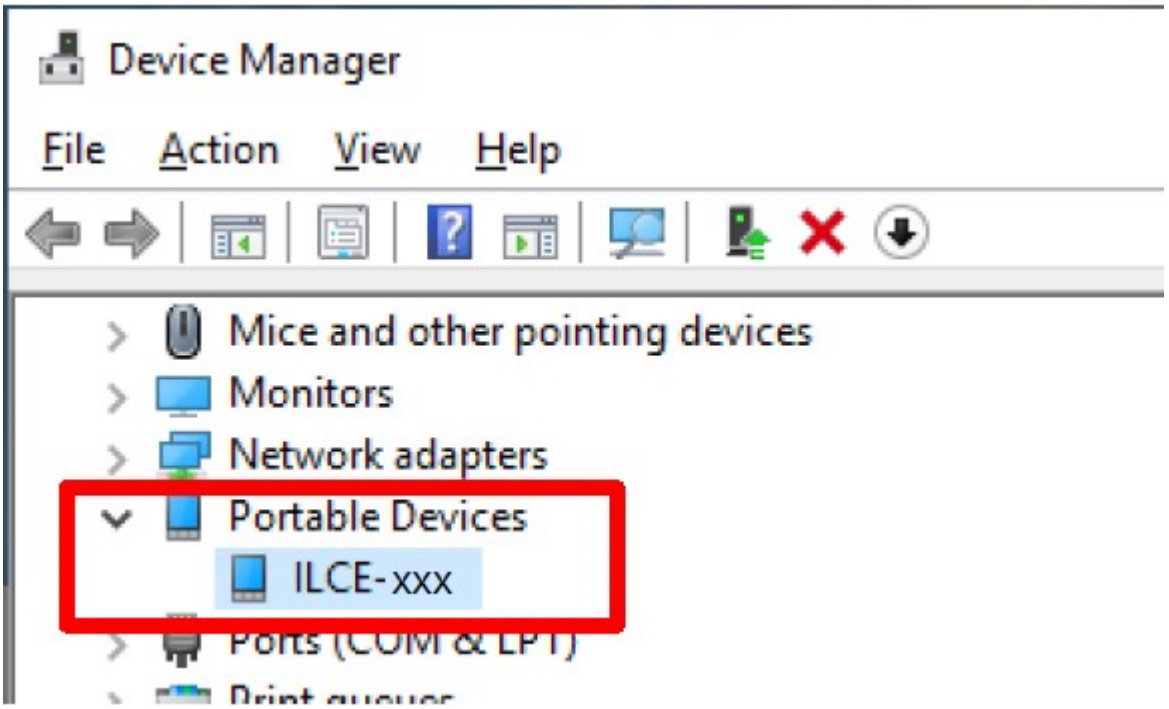


2. Connect your camera to your PC with a USB cable

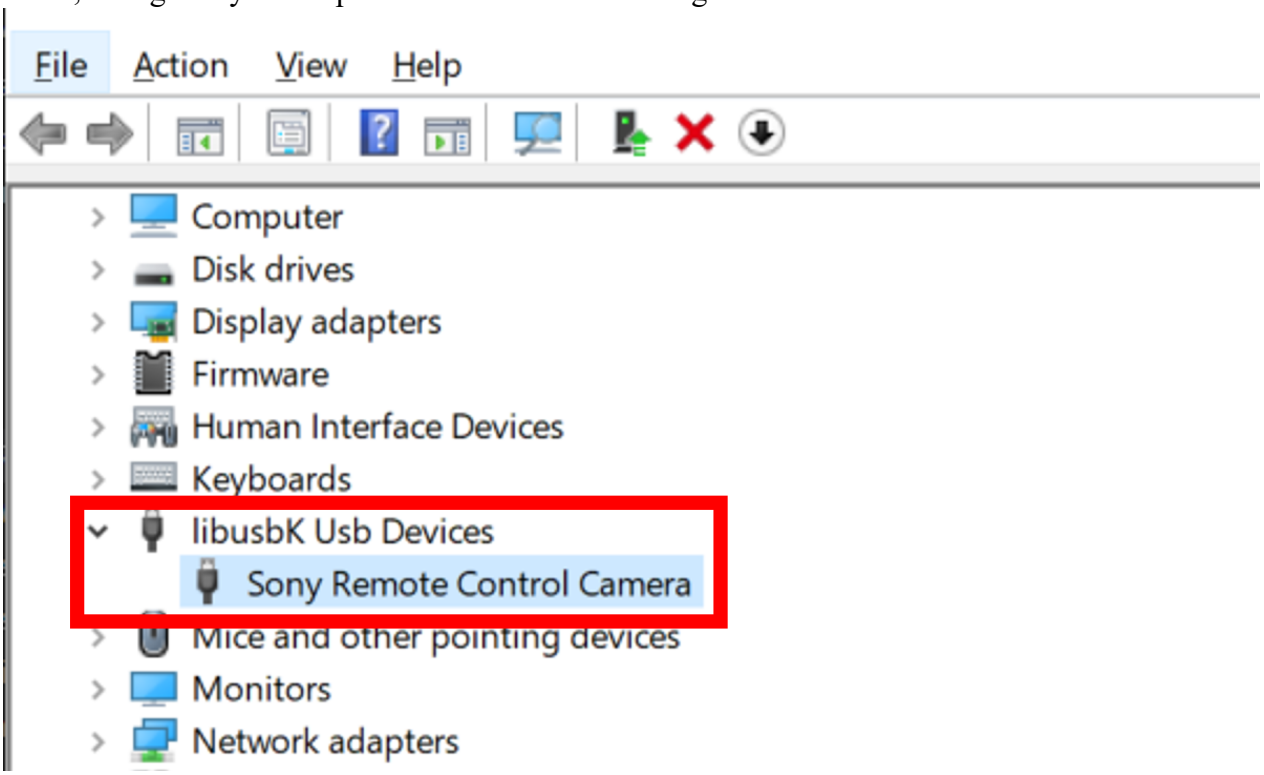
3. Select “Remote Shoot (PC Remote)”



4. Ensure that the connected camera (e.g., “ILCE-xxx”) is under “Portable Devices” in the “Device Manager” window

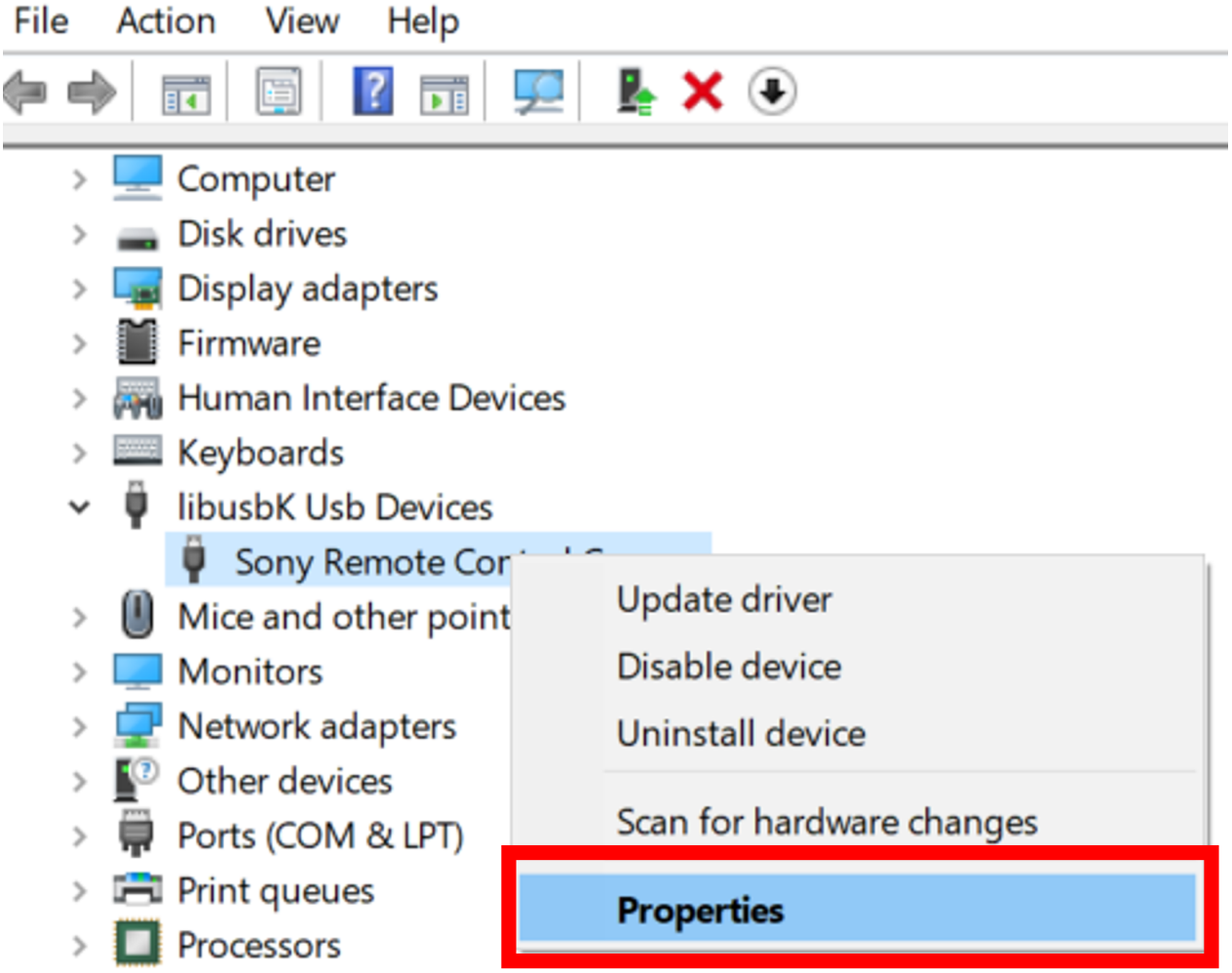


If not, change it by the steps described in the following section

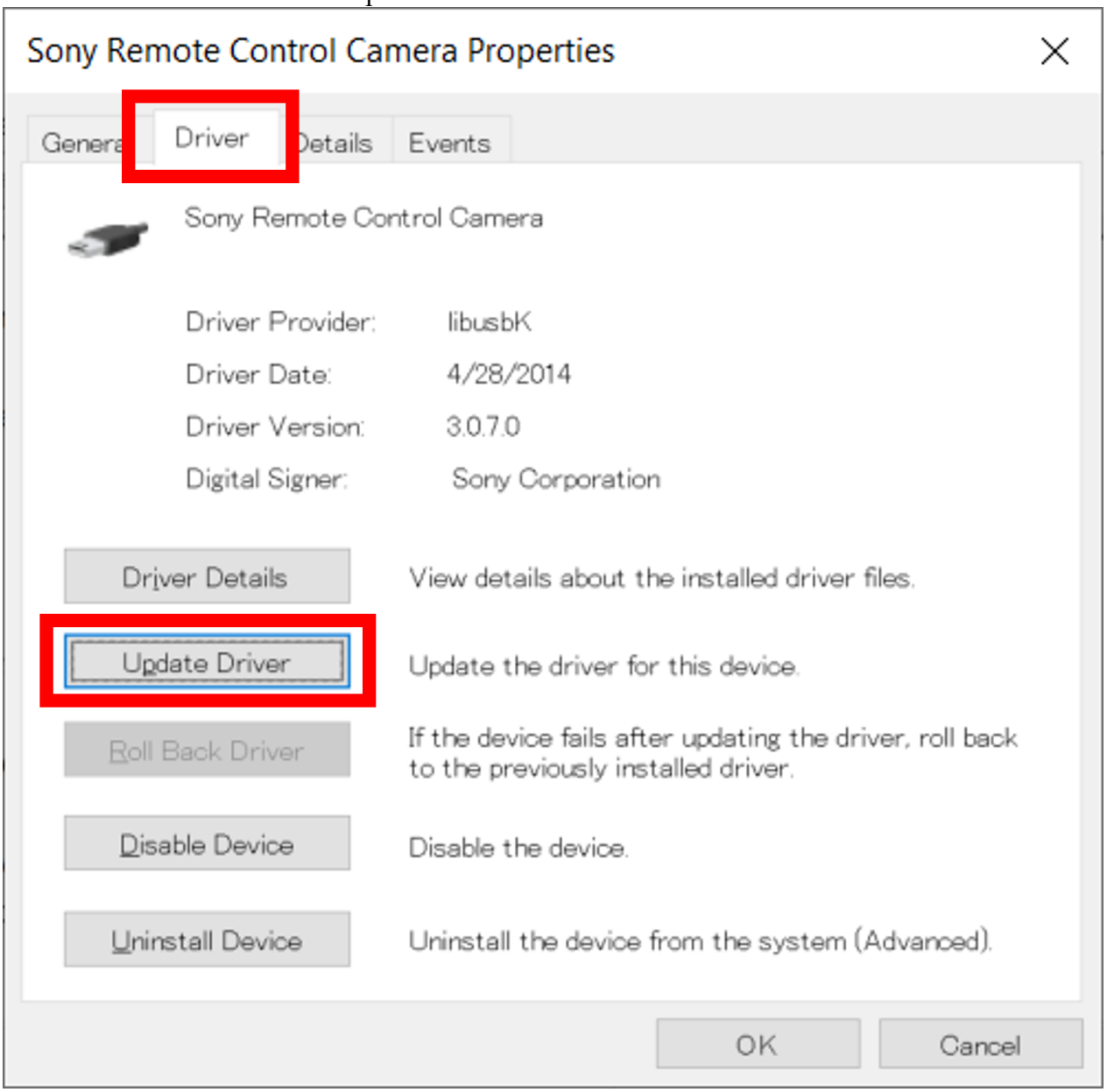


Change Camera Driver

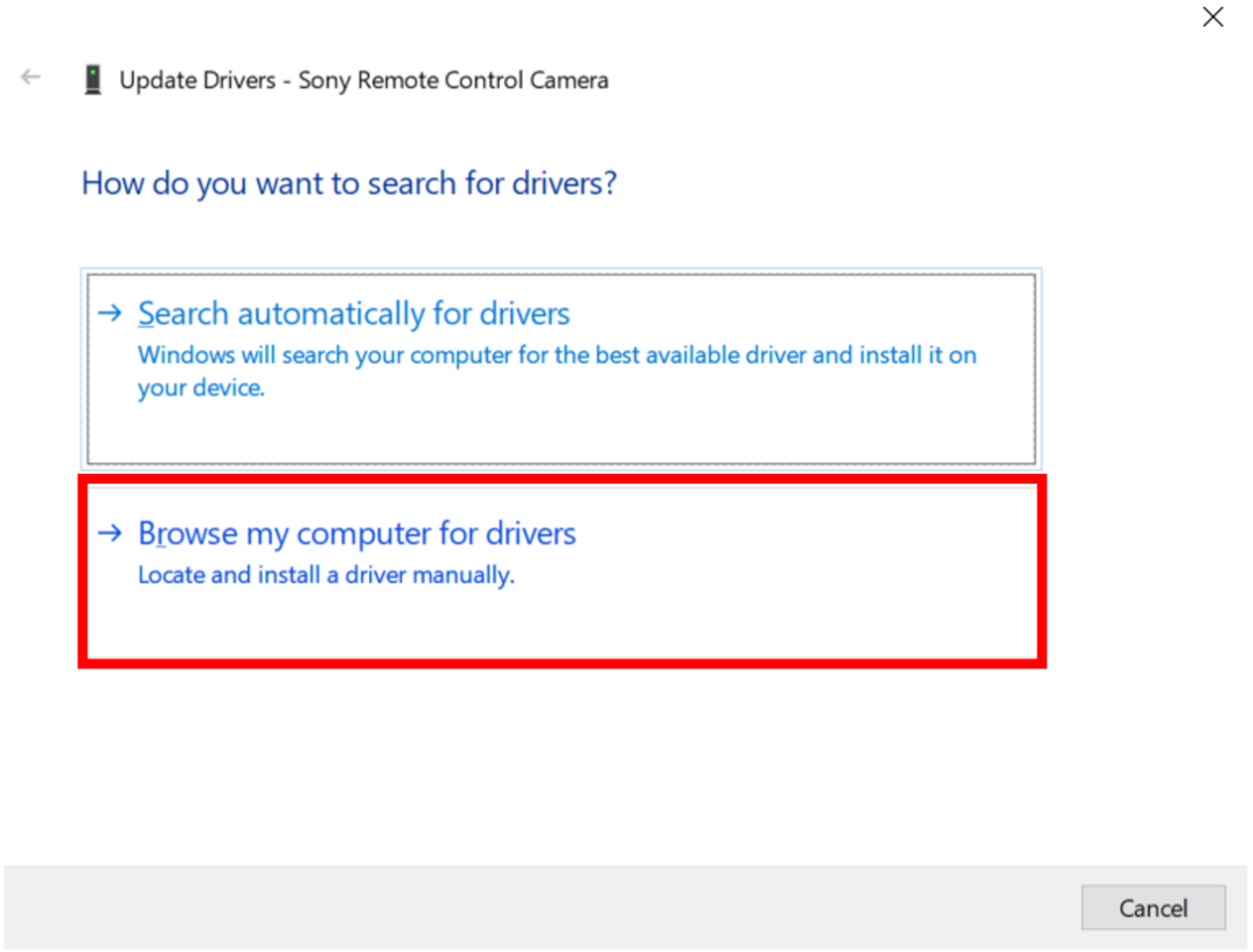
1. Open “Properties”



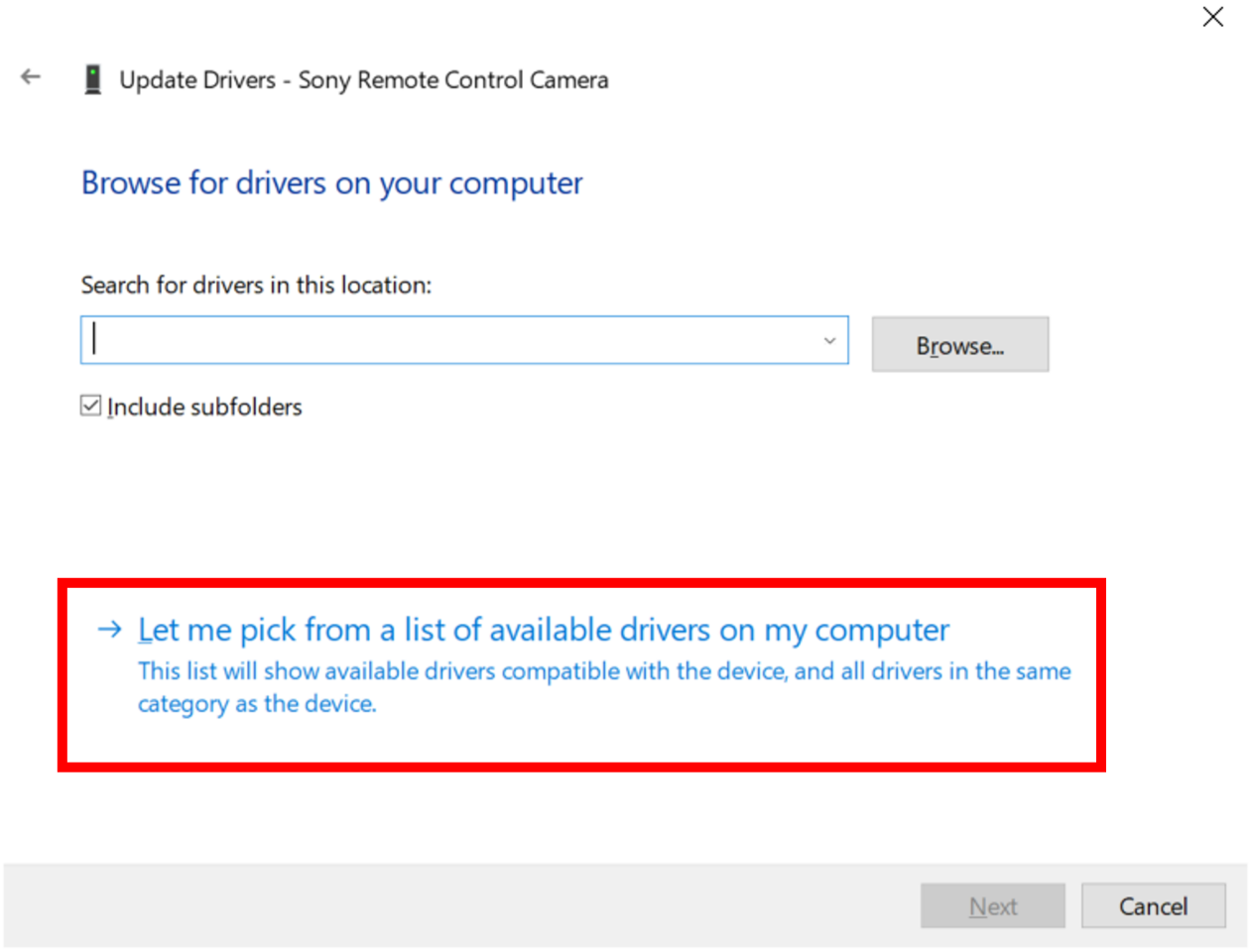
2. Select “Driver” tab and click “Update Driver”



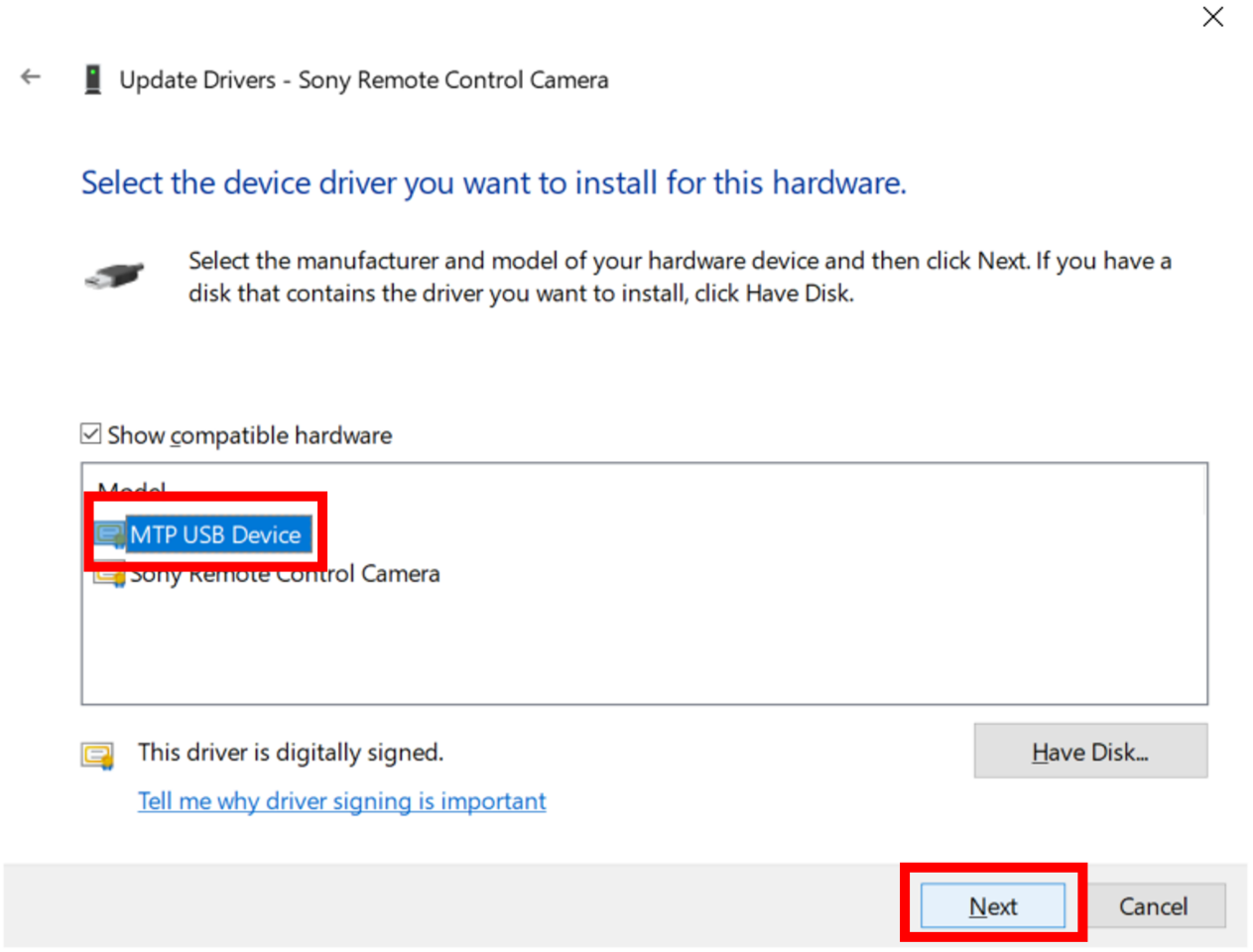
3. Select “Browse my computer for drivers”



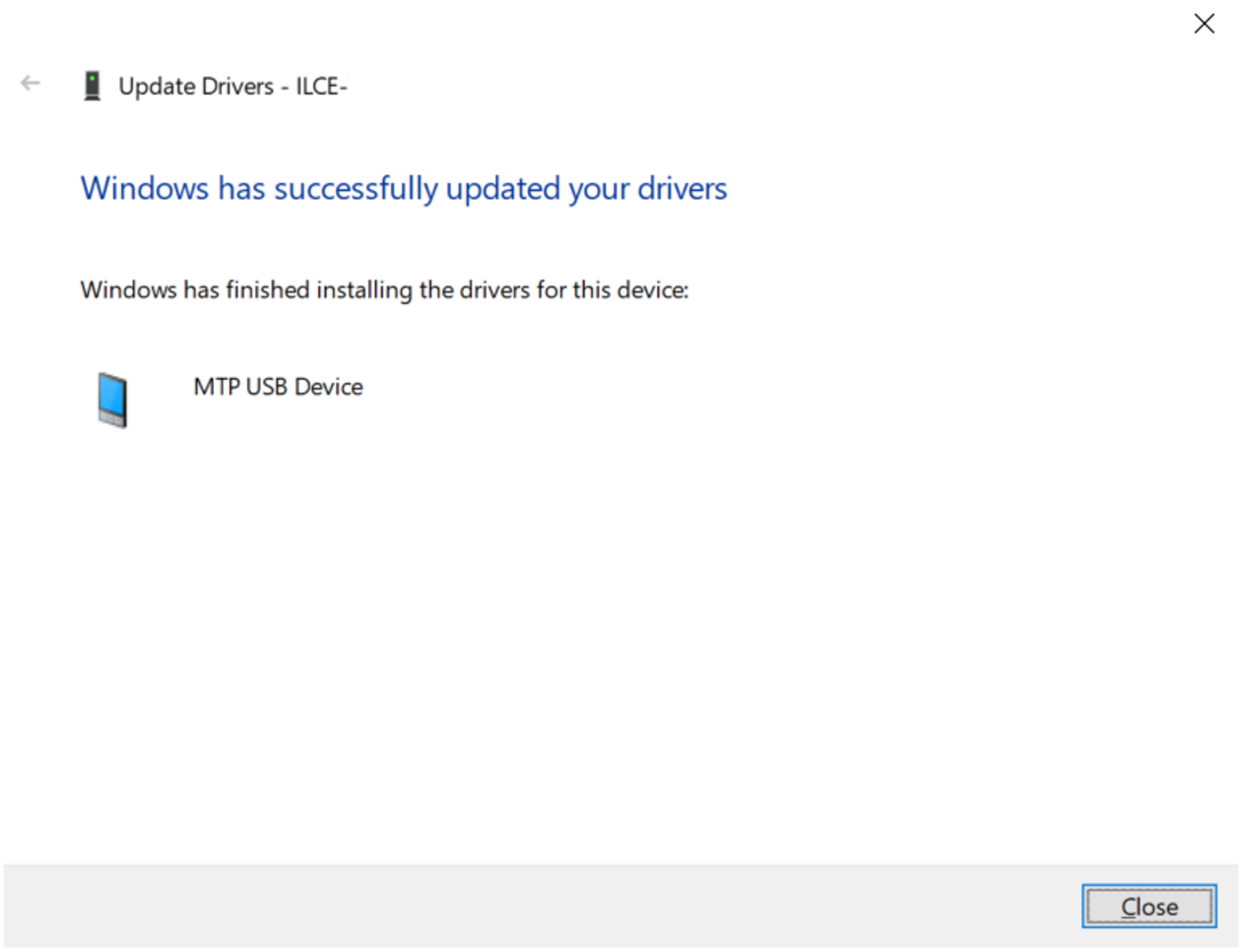
4. Select “Let me pick from a list of available drivers on my computer”



5. Select “MTP USB Device” and click “Next”

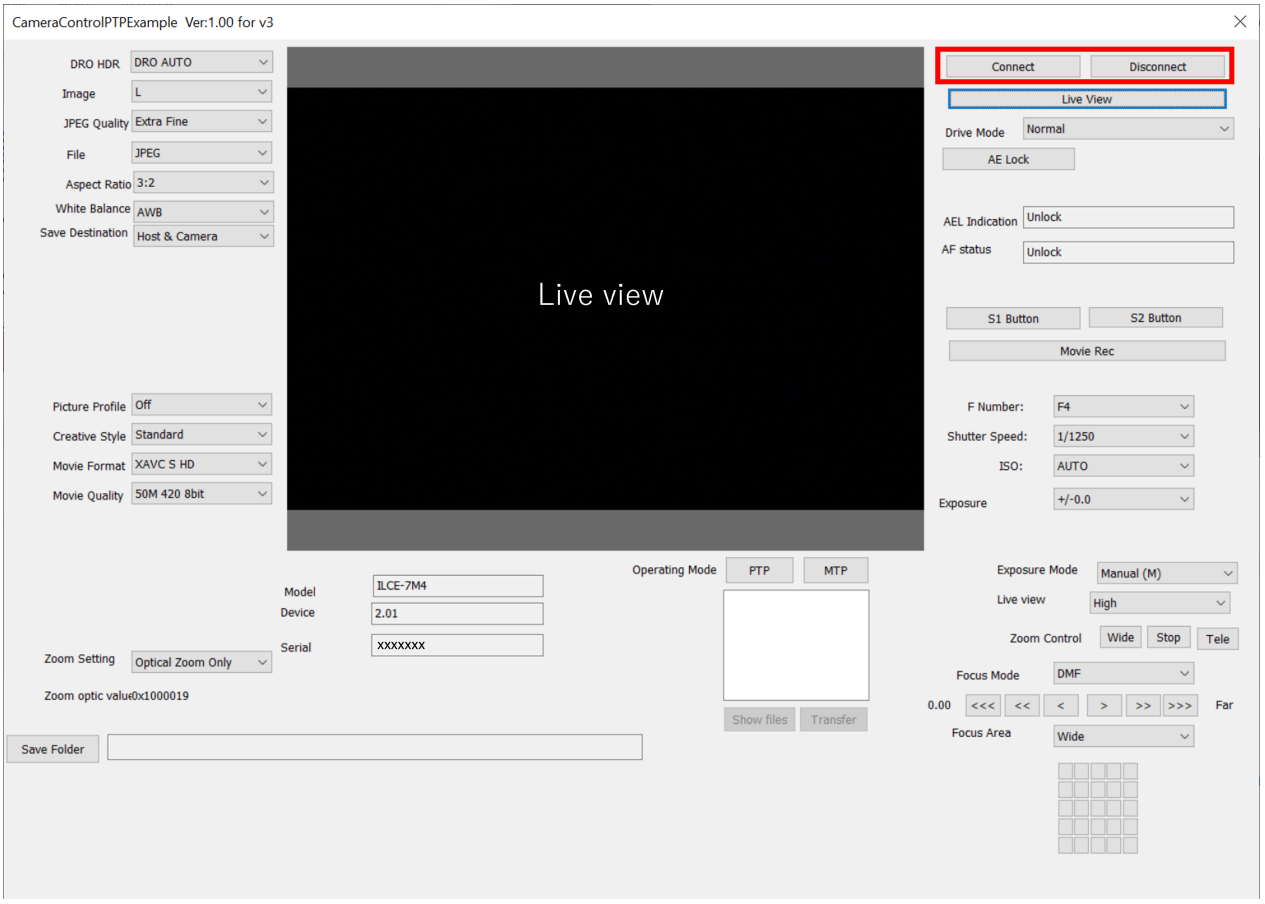


6. Change driver completed



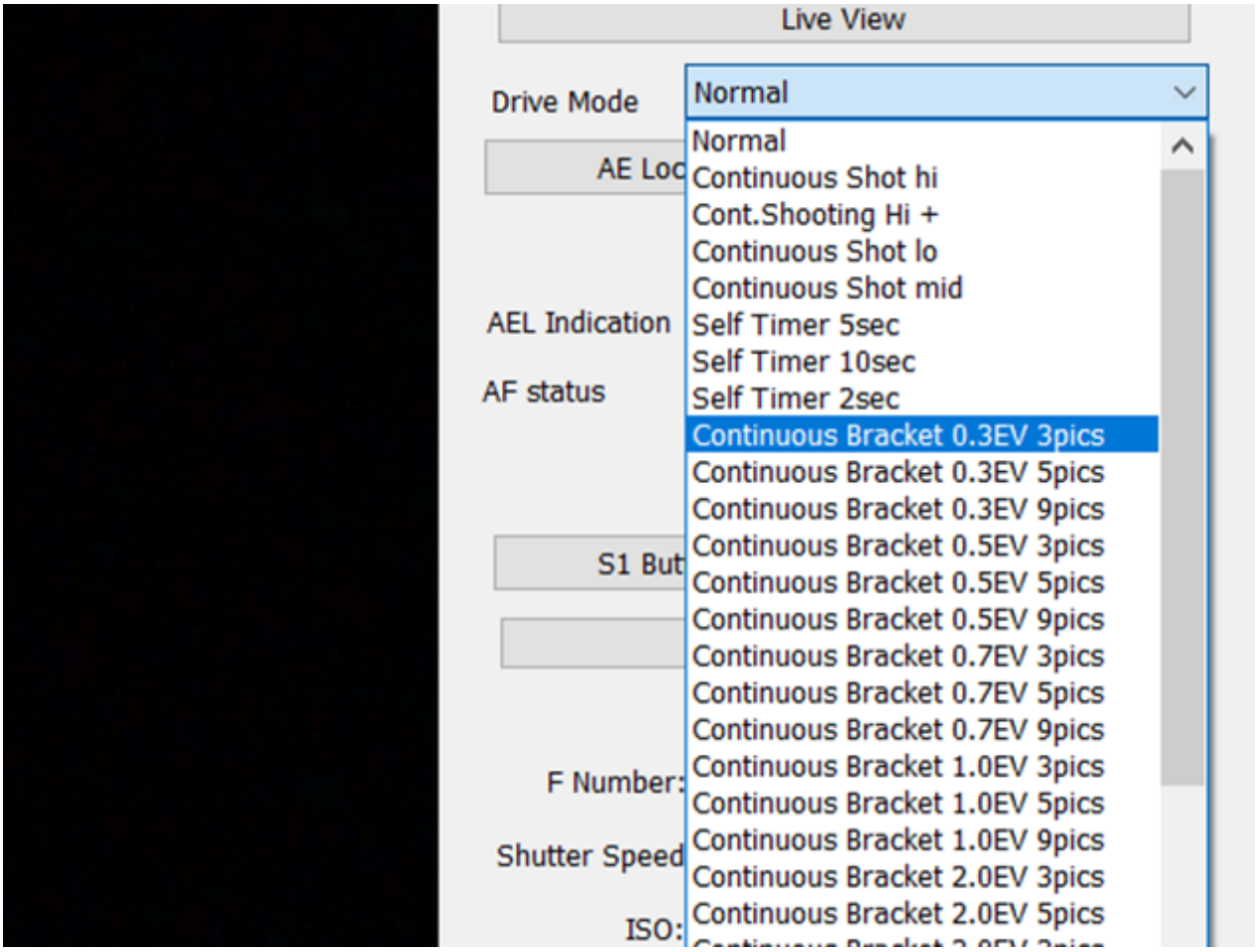
Open and Close Session

To open a Camera Control PTP session to the camera, press the “Connect” button. If the connection is successful, the values of device properties will be displayed. Press the “Disconnect” button to close the session.



Display and Change Device Properties

When the “isEnabled” state of a device property is enabled, its value can be obtained and set. To set a new value, change the value of the corresponding control, such as the “Drive Mode” combobox:



Shutter Release

1. Press “S2 Button”
2. The camera will take a photo
3. If “Save Destination” was set to “Host & Camera,” the image will be saved in the folder you did in “Save Folder”

Shutter Half-Release in AF Mode

1. Press “S1 Button” (shutter half-press down)
2. The camera will make AF control with a half-release
3. Press “S1 Button” (shutter half-press up)

Shutter Half- and Full-Release in AF Mode

1. Press “S1 Button” (shutter half-press down)
2. The camera will make AF control with a half-release
3. Press “S2 Button”
4. The camera will take a photo
5. If “Save Destination” was set to “Host & Camera,” the image will be saved in the folder you did in “Save Folder”
6. Press “S1 Button” (shutter half-press up)

Movie Recording

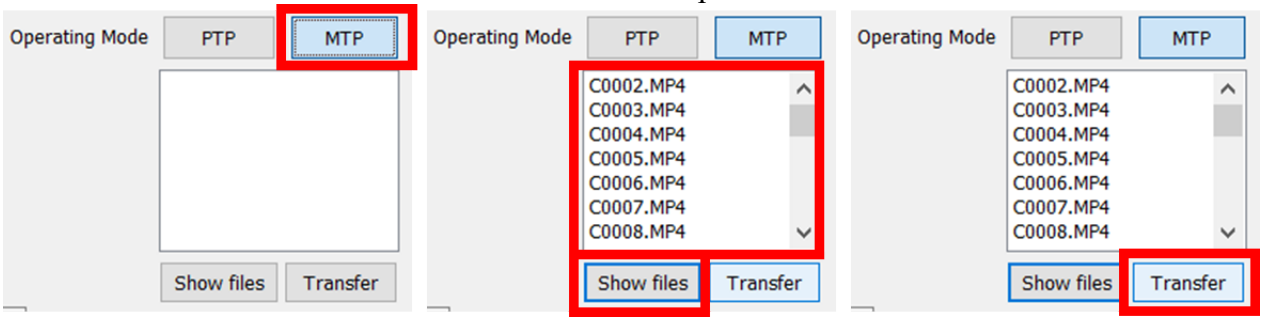
1. Press “Movie Rec” to start recording

2. Press again to stop

Content Transferring

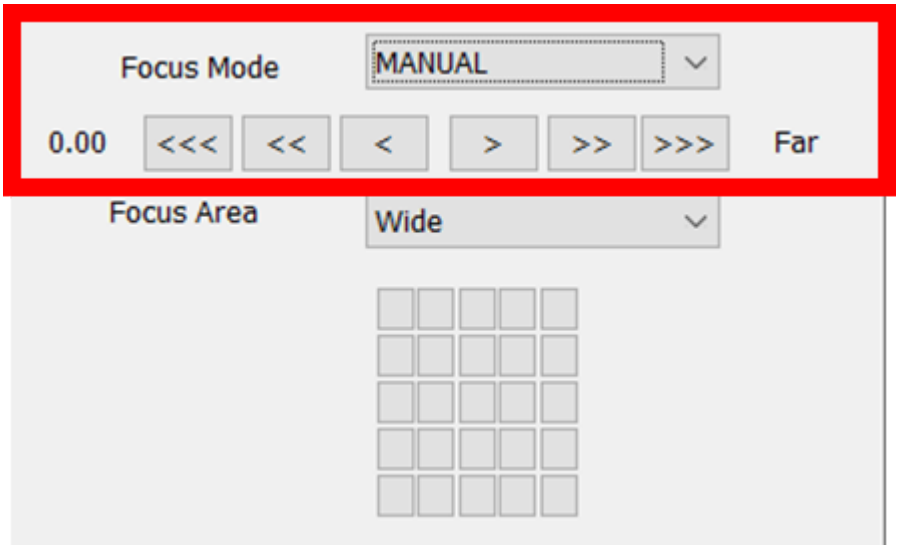
To download images and videos stored in the camera, use the Contents Transfer Mode by the following steps:

1. Press “MTP” to enter the Contents Transfer Mode
2. Press “Show Files” to retrieve and show the content list
3. Press “Transfer” to save all the contents to the folder specified in “Save Folder”

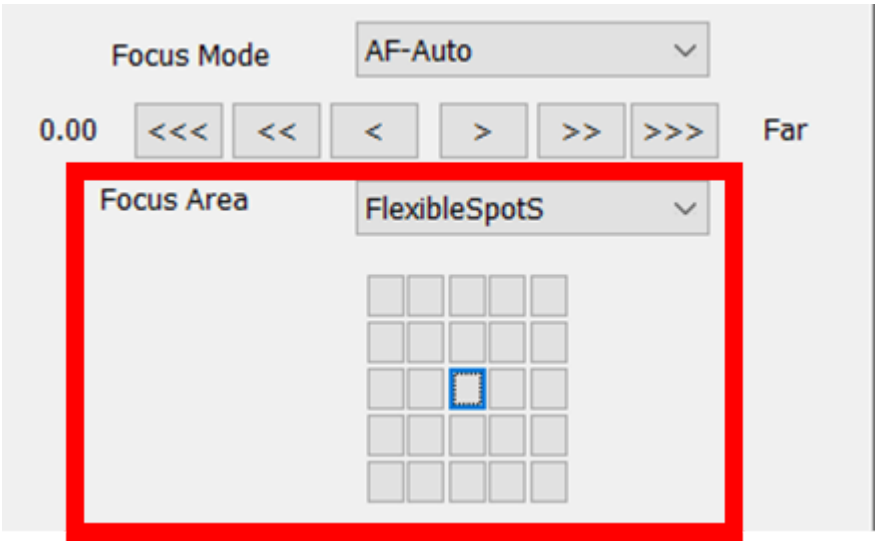


Focus Mode

- When in the manual-focus mode, the focus can be controlled near/far by pressing the “>” and “<” buttons.



- (Only in Version 3) When in the auto-focus mode, you can select “Focus Area Mode” and control “Focus Area Position.”



White Balance (only in version 2)

When the white balance is “C.Temp.,” the color temperature can be set using the slider.