Canon EOS cameras - Principles of interfacing and library description.

1. Introduction

Throughout it's history, Canon DSLR cameras came with several different control protocols. The interface described on this page is implemented in all current (as of Fall 2010) Canon EOS cameras, as well as some past models. While camera data is being collected, current knowledge is published on PTP/USB control camera data page. The cameras known to work with current library code are 400D, 450D, 500D, 550D, as well as 5D Mark II, 7D and some others. This list will be expanded as soon as new data become available.

Canon control protocol extends standard PTP set with a number of vendor-specific commands, properties and events. Data exchange follows PIMA 15740-2000, with minor differences (for example, Canon EOS cameras send events via bulk-IN pipe instead of interrupt pipe as it is defined in PTP specification). PTP exchanges are hidden from the user – you do not have to worry about opening PTP session or camera initialization, these actions are performed by PTP library automatically when appropriate.

2. Basic principles of EOS camera control

Typical data exchange between application and Canon EOS camera goes like this:

1. A PTP session to the camera is open
2. Camera is set to PC Connection mode
3. `CanonEOS::SetExtendedEventInfo` command is sent to the camera
4. `CanonEOS::EventCheck` is used to poll camera for events. At this point, camera sends back rather large packet containing specific type of event where it returns information about all device property values and their value enumerators. This packet can be parsed to find out which camera properties can be changed and which values are available – for example, 7D has exposure compensation range from -5 to +5, whereas many other cameras have Ev compensation range from -2 to 2. On the other hand, if an application is designed to work with certain type of camera only or doesn't need to change properties, this packet can be ignored.

At this point, application may start changing camera properties and/or send shooting commands to the camera. At the same time, camera generates events, which can be received using `CanonEOS::EventCheck`. Some events are generated in response to commands (good example is "Object Created" when picture is taken), some happen asynchronously (like "Low Battery"). Since many events carry important information about changes in camera capabilities for the current shooting mode (as well as lens, zoom, etc.), processing event information takes significant part of an application intended to modify camera settings.

The event information is returned in a standard PTP data packet as a number of records followed by an empty record at the end of the packet. Each record consists of multiple four-byte fields and always starts with record length field. Further structure of the record depends on the device property code which always goes in the third field. The empty record consists of the size field and four byte empty field, which is always zero.

To be able to work with your EOS camera you have to parse that event packet for necessary device property values and monitor all the changes of those properties later in your program. Here is the

dump generated by `EOSEventMonitor` sample sketch when changing exposure time (AKA shutter speed):

```
0000:2C 00 00 00 02 00 16 91 0D 00 00 00 08 00 00 00
0010:8E C1 00 00 10 00 00 00 89 C1 00 00 02 D1 00 00
0020:5B 00 00 00 08 00 00 00 00 00 00 00
```

The first 12 bytes is a standard PTP header followed by event records. The second event record, starting from 0×0014 offset holds information about exposure time change. The first field is the record length 0×10 (16 bytes), the second one is `EOS_EC_DevPropChanged` event code, the third is device property code which is exposure time and the fourth is the new value 0x5B which corresponds to 1/20 sec. The last is the empty record consisting of the length and zero fields. If changing device property affects the values accepted for the property, (for example, manual zoom change which changes aperture value from 3.5 to 5.6 because the former is not available for the selected zoom range and therefore the range of possible aperture values accepted for selected lens zoom value), `EOS_EC_DevPropValuesAccepted` is generated too. So if you going to deal with several lenses you have to parse not only `EOS_EC_DevPropChanged`, but also `EOS_EC_DevPropValuesAccepted` to be able to set proper value of aperture at any time for any lens.

It is not possible to poll for specific events; In reply to `CanonEOS::EventCheck` camera sends whatever events have happened since the last poll. It is up to the application to parse events it needs and ignore the others. It is also important not to poll too often – the camera may hang. While maximum accepted polling rate varies from one camera model to the other, a 200ms pause between polls seems not be causing issues on any of the cameras that we tested.

I want to say a few words about how commands and device property operations are implemented in EOS camera protocol in comparison to standard PTP commands and properties defined by PIMA. As you might know, PTP protocol transaction consists of three stages: command stage, response stage and optional data stage. PTP protocol defines a set of commands and properties. Each PTP command starts with PTP packet header holding size, type of packet, operation code and transaction ID, followed by a set of optional four-byte parameters which are transferred as a part of the command block. Most PTP standard commands pass their arguments in optional parameters in command block while some of the EOS protocol commands, such as `PTP_OC_EOS_SetDevicePropValue`, pass their command arguments in EOS specific data block inside of the standard PTP data block. Each EOS data block consists of one or more four-byte fields. The first field always holds EOS data block length in bytes, while values of the other fields depend on the command type.

EOS utilizes PTP standard device properties with vendor defined property codes along with EOS specific device properties which are set differently. PTP-like device properties are set using standard PTP mechanism via `PTP_OC_SetDevicePropValue` where property code is passed as a first parameter in command block and the value itself is passed in the data block.

On the other hand, EOS specific device properties are set via `PTP_OC_EOS_SetDevicePropValue` command where arguments are passed in EOS specific data block inside PTP data block as follows: EOS data block size, EOS device property code and value. Standard properties are used to handle parameters common for all Canon cameras such as human readable camera name. EOS specific properties are used mainly to handle EOS specific parameters.

3. Application interface

The main camera control loop is implemented as a state machine in method `CanonEOS::Task()`. This method takes care of all actions necessary when camera is connected or disconnected. The application code starts when camera is initialized and ready to accept commands. To place your code in a control loop, you have to supply derived class address. `EOSStateHandlers` provides methods for handling different camera states. With this class, all you have to do is add your code in virtual function `OnDeviceInitializedState` and optionally `OnDeviceDisconnectedState` if you want to handle camera disconnected state. `CanonEOS` class instance gets control when function `CanonEOS::Task` is called from loop function. `OnDeviceInitializeState` in turn is called in every loop of the cycle while the camera is connected and initialized. You have to keep it in mind that your code should not be taking control for extended periods of time otherwise the rest of the code called from the loop function will not get control until it is done. Your code called from `OnDeviceInitialized` state should be written as a state machine and return control to the caller function as soon as a small portion of code is performed and be capable of performing another operation next time `OnDeviceInitialized` is called.

Here is a small sample sketch which captures images every five second. An explanation of important parts is given after the source text.

```
#include <Spi.h>
#include <Max3421e.h>
#include <Max3421e_constants.h>
#include <Max_LCD.h>
#include <Usb.h>
#include <Ptp.h>
#include <Canoneos.h>

#define DEV_ADDR        1

// Canon EOS 400D
#define DATA_IN_EP      1
#define DATA_OUT_EP     2
#define INTERRUPT_EP    3
#define CONFIG_NUM      1

void setup();
void loop();

class CamStateHandlers : public EOSStateHandlers
{
      bool stateConnected;
public:
      CamStateHandlers() : stateConnected(false) {};
      virtual void OnDeviceDisconnectedState(PTP *ptp);
      virtual void OnDeviceInitializedState(PTP *ptp);
} CamStates;

CanonEOS  Eos(DEV_ADDR, DATA_IN_EP, DATA_OUT_EP, INTERRUPT_EP, CONFIG_NUM, &C

void CamStateHandlers::OnDeviceDisconnectedState(PTP *ptp)
{
    if (stateConnected)
    {
        stateConnected = false;
        Notify(PSTR("Camera disconnected\r\n"));
    }
}

void CamStateHandlers::OnDeviceInitializedState(PTP *ptp)
{
    if (!stateConnected)
        stateConnected = true;

        uint16_t rc = Eos.Capture();

        if (rc != PTP_RC_OK)
            Message(PSTR("Error: "), rc);

```

- **Lines 6,7** These two headers are necessary to access library functions.
- **Lines 12-15** Endpoint numbers and configuration number. Theoretically, this information is camera-specific, however, in practice, all Canon EOS and Powershot cameras that we've seen have identical endpoint numbers, configuration number, as well as maximum packet size. Therefore, this information is hard coded for the time being.
- **Lines 20-27** Derivation of `CamStateHandlers` class with name `CamStates`. The address of the class is given to the constructor on the next line.
- **Lines 31-38** A handler for camera disconnected state.
- **Lines 40-51** A handler for "camera ready" state. Shooting occurs on **line 45**. The delay on **line 50** is an example of bad coding - the code execution will be blocked for 5 seconds and all events would have to wait that long to be handled. It is advised to always program delays in non-blocking manner.
- **Line 57** `Eos.Setup()` has to be called once.
- **Line 62** This is the place from which `OnDeviceDisconnectedState` and `OnDeviceInitializedState` are called. Therefore, `Eos.Task()` should be allowed to run from time to time in order to be able to react to camera state changes.

This is probably the simplest example of EOS camera control - more advanced examples are contained in `examples/` directory of library code.

4. Method summary of the `CanonEOS` class

```
CanonEOS(uint8_t addr, uint8_t epin, uint8_t epout, uint8_t epint, uint8_t
    addr        Device address
    epin        IN end point number
    epout       OUT end point number
    epint       INTERRUPT end point number
    nconf       Configuration number
    s   Address of PTPStateHandlers derived class

uint16_t SetPCConnectMode(uint8_t mode)
    Sets PC connection mode. 1 – switches the camera to PC connect mode. 0

uint16_t SetExtendedEventInfo(uint8_t mode)
    This command sets the camera extended event info mode. When turned on
    mode  Integer
    mode value. 1 - turns extended event mode on, 0 – off.

uint16_t Capture()
    Initiates capture with current camera settings.

uint16_t StartBulb()
    Initiate bulb capture. Use StopBulb method to stop bulb capture.

uint16_t StopBulb()
    Stops bulb capture initiated by StartBulb method.

uint16_t SwitchLiveView(bool on)
    Turns Live View on/off.
    on  Turns Live View on if true, off otherwise.

uint16_t MoveFocus(uint16_t step)
    Moves focus on some EOS cameras capable of performing this operation.

uint16_t SetProperty(uint16_t prop, uint32_t val)
    Sets EOS specific device property value.
    prop        Property code.
    val Value to be set.

uint16_t SetImageQuality(uint32_t format)
    Sets image format and compression ratio.
    format  Four-byte unsigned integer value, defined in canoneos.h header

virtual uint16_t EventCheck(PTPReadParser *parser)
    Retrieves event info and passes it to a parser specified by *parser pc
    parser  Pointer to PTPReadParser – derived class.
```

5. Code examples

The EOS-specific code example demonstrating sending "Capture" command to the camera

can be found in section 3 above. This is a complete sketch, it can be copied from this page and pasted into Arduino IDE window. Several other examples (2 at present) can be found in the example directory of the gitHub PTP library repo.

PTPDevInfo

This sketch connects to PTP device, sends PTP::GetDeviceInfo command, and prints the output in human-readable form. This sketch is not camera-specific, it can be used to explore any device which talks PTP. Examples of output generated by the sketch can be seen in **GetDeviceInfo** output collection.

EOSEventLab

This sketch is useful for exploring events generated by EOS camera. It connects to the camera, sets proper event-generating parameters, then goes on to continuously poll the camera for events while printing event output in hex format. Below you can see this sketch outputting events from Canon EOS 7D camera in Av mode while exposure compensation has been changed using camera control buttons. Four changes has been made.

```
    44 00 00 00
1   10 00 00 00 89 C1 00 00 04 D1 00 00 FD 00 00 00
2   10 00 00 00 89 C1 00 00 04 D1 00 00 FD 00 00 00
3   10 00 00 00 89 C1 00 00 02 D1 00 00 48 00 00 00
4   08 00 00 00 00 00 00 00
5
6
7   44 00 00 00
8   10 00 00 00 89 C1 00 00 04 D1 00 00 00 00 00 00
9   10 00 00 00 89 C1 00 00 04 D1 00 00 00 00 00 00
10  10 00 00 00 89 C1 00 00 02 D1 00 00 45 00 00 00
11  08 00 00 00 00 00 00 00
12
13  44 00 00 00
14  10 00 00 00 89 C1 00 00 04 D1 00 00 03 00 00 00
15  10 00 00 00 89 C1 00 00 04 D1 00 00 03 00 00 00
16  10 00 00 00 89 C1 00 00 02 D1 00 00 43 00 00 00
17  08 00 00 00 00 00 00 00
18
19
20  44 00 00 00
21  10 00 00 00 89 C1 00 00 04 D1 00 00 05 00 00 00
22  10 00 00 00 89 C1 00 00 04 D1 00 00 05 00 00 00
23  10 00 00 00 89 C1 00 00 02 D1 00 00 40 00 00 00
    08 00 00 00 00 00 00 00
```

- **Line 1** Total PTP packet length, in this case, 68 bytes
- **Line 2** First four bytes - length (16 bytes), next four bytes - `EOS_EC_DevPropChanged` event (C189), next four bytes - `EOS_DPC_ExposureCorrection` property (D104), last four bytes - value (FD, which corresponds to -1/3)

- **Line 3** Same as Line 2
- **Line 4** Third four-byte field - `EOS_DPC_Exposure` property (D102), fourth - property value (48).
- **Line 5** Last (empty) record of EOS packet.

Parsing of lines 7 - 23 is left as an exercise for the reader. Note how exposure time (AKA shutter speed) goes down while exposure compensation goes up.

Attention! This sketch requires Time library to be installed in order to compile.

6. Links

- GitHub repo
- Introductory article to USB camera control