ABCanon7 library, v2.03

Programming guide and function reference

Rev G, January, 4th 2021

Revision history:

Rev A, April, 26th 2016
First release

Rev B, July, 10th 2016
Documented library version 1.01

Rev C, July, 24th 2016
Documented library version 1.02

Rev D, December, 11th 2016
Documented library version 2.00

Rev E, March, 16th 2017
Documented library version 2.01

Rev F, February, 10th 2020
Documented library version 2.02

Rev G, January, 4th 2021
Documented library version 2.03

Index

1 Introduction notes

I used the original Canon library (and in particular the Canon Utilities software) on my old Acer Ferrari PC with Windows XP for nearly 10 years without any problem, and I was very happy about that. But a couple of years ago I moved to a more modern Windows 7 PC and have also in the meantime acquired a second 350D camera. This prompted me to find a way to use both camera on the same PC at the same time. Not having found a suitable solution I decided to write my own, and soon (well, not that soon), this library was born.
Since the kind of photography I normally do is (deep sky) astrophotography, this library has been designed and tested with that kind of use in mind. While other use of this library are possible, these weren't as tested as Manual mode.

This library also tries to avoid sending incorrect commands to the camera by limiting the allowed operations to those that can be done using the original Canon Utilities software. The only exception to this is the capability to set the shutter speed to BULB, because several third party software do this without any problem.

As a general rule for this library, all `int`-returning functions return an error when the return value is less than 0. Regular return values (not errors) are greater than or equal to zero, while `char *`-returning functions return `NULL` when an error occurs, without any specific indication about which error occurred.
There are a few exception to these rules, and they are indicated in the specific function description.
For a complete list of the errors that may be returned refer to `ABCanon.h`,

The `ABUTestDll` program is a very simple reference example about using this library. Please refer to `ABUTestDll.c` for examples.
A different example (more complex but less complete) can be found in the (VB2010) .NET source code for the `ABUtilites` program that mimics the original Canon Utilities software distributed with the Canon 350D.
With version 2.03 the `ABUtilites` program has been replaced by `ABUtilites2`, (written in VB.NET 2013), which adds some sort of "timer" to automate shooting (and it is also an example I used to test async/await with the library).

Version 2.00 adds support for the Canon 5D.

2 Theory of operation

This chapter introduce some basic concepts about using this library, but is not intended to be a complete reference for the functions used. Some functions are not described at all, while other have an incomplete parameter list description. For a complete reference of the available functions, refer to *Alphabetical function reference* on page 21.

2.1 Step #1: Retrieving the list of connected cameras

To retrieve the list of connected cameras, call the `ABCanonBuildCameraList (struct sCameraData **ppCameraData)` function.
This function compiles an array of the available cameras (using an internal memory buffer), returns a pointer to this list in the `ppCameraData` parameter and the number of cameras found as the return value.
When this list is no longer needed, call the `ABCanonReleaseCameraList (struct sCameraData *pCameraData)` function to release the memory.

The returned array is an array of:

```
struct sCameraData
{
 uint32_t                   Idx;
 uint32_t                   Status;
 uint32_t                   SerialNumber;
 unsigned char              CameraName[32];
 uint8_t                    FWRelease[3];
 uint8_t                    CameraType;
};
```

The fields of this structure are:

`Idx`	The index. Starts from 0.
`Status`	The status of this camera. You should normally see only `ABC_STS_OK` indicating that the camera is available or `ABC_STS_INUSE` when the camera has been already open (connected) by some other process. There are other possible values but you should not normally see them (they only occurs in case of errors). Refer to `ABCanon.h` for the complete list.

The following fields will be compiled only when `Status == ABC_STS_OK`.

`SerialNumber`	The camera serial number.
`CameraName`	The ASCIIZ camera name, as returned by the camera.
`FWRelease[3]`	The camera firmware release, as FWRelease[0].FWRelase[1].FWRelease[2]
`CameraType`	The camera type. In this release it can be `ABCTYPE_350D` or `ABCTYPE_5D`.

An example about how to use these function can be found in the "scan" command in ABUTestDLL.c.

This first step is somehow optional, because the "Connect" function can be forced to connect a specific camera (designated by its serial number, which should be known beforehand) or just the first camera available.

2.2 Step #2: connecting to a camera

To connect to a camera, call the ABCanonConnectToCamera (int ConnectMethod, int Filter, uint32_t Parm, ABCAMERA_HANDLE *phCH) function.
There are three ways to connect to a camera (with the ConnectMethod parameter):

- CONNECT_FIRST_AVAILABLE: just connect the first available camera. Cameras already in use will not be counted, as cameras that don't satisfy the Filter parameter. Parameter Parm is unused.
- CONNECT_BY_INDEX: connect the camera with the Idx parameter (in the camera list, see step #1) equal to parameter Parm. Note that if the list changes between the call to ABCanonBuildCameraList() and the call to ABCanonConnectToCamera() you may end up connecting to the wrong camera. Note also that if the Idx camera does not satisfy the Filter parameter the function will fail.
- CONNECT_BY_SERIAL_NUMBER: Connect to the camera with the serial number equal to the Parm parameter and satisfying the Filter parameter. If no camera can be found, the function will fail.

The Filter parameter can be ABCTYPE_ALL indicating that no filtering should be performed, or an ABCTYPE_xxx value (either ABCTYPE_350D or ABCTYPE_5D in this release).

If the connection has been successful the return value will be ABC_RETVAL_OK and the camera handle will be returned in phCH, otherwise the return value will be an error indicating why the camera cannot be connected.

See the commands "open1", "openidx" and "opensn" in ABUTestDLL.c for examples.

2.3 Step #3: Selecting the image destination

When an image is taken (both using the shutter button or calling the ABCanonShoot() function) the camera can save the image to the CF and/or to the PC. This can be chosen with the ABCanonSetSaveTarget(ABCAMERA_HANDLE hCH, int SaveTarget) function. You can use ABC_SAVETGT_PC to save to the PC only, ABC_SAVETGT_CF to save to the CF only or (ABC_SAVETGT_PC + ABC_SAVETGT_CF) to save to both destinations at the same time. Note that when the camera is first connected, the destination will be ABC_SAVETGT_CF and should be changed if needed, while when the connection to the camera is closed (by calling ABCanonDisconnectFromCamera()) the destination will be automatically reverted to ABC_SAVETGT_CF.

When the camera has the ABC_SAVETGT_PC target active, the image will be downloaded to the PC via the USB connection.
This download can be handled in four different modes, selected with the ABCanonSetDownloadMode(ABCAMERA_HANDLE hCH, int DownloadMode) function:

- ABC_IMGDL_DISCARD: the image will be downloaded and discarded.
- ABC_IMGDL_SAVE_TO_FILE or ABC_IMGDL_OVERRIDE_OR_DISCARD: the image will be automatically saved to a file. There are functions (see below) to set the path/filename/image counter where the image should be saved.
- ABC_IMGDL_SAVE_TO_MEMORY: the image will be downloaded to memory.

The event that an image has been taken, discarded, downloaded to a file or to a memory buffer will be notified via the notification call back or the event notification queue. See *Notifications/events* below.
For more information about saving, refer to chapter *2.10 Saving the image*.

See the command "autosavefn", "overridefn" and "downloadmode" in ABUTestDLL.c for examples.

2.4 Step #4: Changing the image settings

This library can get the following image settings and can set some of them, depending on the mode the camera is in:

ABC_IMGSETTING_MODE_DIAL	Always read-only
ABC_IMGSETTING_SHUTTER_SPEED	
ABC_IMGSETTING_APERTURE	
ABC_IMGSETTING_ISO	
ABC_IMGSETTING_EXP_COMPENSATION	
ABC_IMGSETTING_METERING	
ABC_IMGSETTING_AUTOFOCUS	Always read-only
ABC_IMGSETTING_WHITE_BALANCE	
ABC_IMGSETTING_PICTURE_QUALITY	
ABC_IMGSETTING_FLASH_EXP_COMP	
ABC_IMGSETTING_AUTO_POWER_OFF	Always read-only
ABC_IMGSETTING_DRIVE_MODE	Always read-only
ABC_IMGSETTING_AEB	Always read-only
ABC_IMGSETTING_COLOR_TEMPERATURE	5D in K White Balance only

The current value for one setting can be retrieved using int ABCanonGetImgSetting (ABCAMERA_HANDLE hCH, int ImgSetting) or all settings can be retrieved at once with int ABCanonGetAllSettings (ABCAMERA_HANDLE hCH, int32_t *pSettingsArray, int SettingsArraySize) where pSettingsArray is an user allocated array of SettingsArraySize items (not bytes) to be filled.
You can check if any setting is writable by calling int ABCanonCheckImgSettingWritable(ABCAMERA_HANDLE hCH, int

`ImgSetting)` or check all of them with `int ABCanonGetWritableSettings(ABCAMERA_HANDLE hCH, int32_t *pSettingsArray, int SettingsArraySize)` used in the same way as `ABCanonGetAllSettings()`.

To change the value of writable settings, call `int ABCanonSetImgSetting(ABCAMERA_HANDLE hCH, int ImgSetting, int SetType, int Value)`. Different types of change can be done:

- `ABC_SETTYPE_FIRST`: set to first (lowest) setting. `Value` is ignored.
- `ABC_SETTYPE_CHANGE`: relative change. Value is the (signed) change to be done. The change is automatically limited to the maximum or minimum value.
- `ABC_SETTYPE_SET`: set the image setting to `Value`. If `Value` is not an allowed value the function returns an error.
- `ABC_SETTYPE_LAST`: set to last (highest) setting. `Value` is ignored.
- `ABC_SETTYPE_PREV`: Previous setting. Similar to `ABC_SETTYPE_CHANGE -1`, but wraps from the first element of the list to the last one when required. `Value` is ignored.
- `ABC_SETTYPE_NEXT`: Next setting. Similar to `ABC_SETTYPE_CHANGE +1`, but wraps from the last element of the list to the first one when required. `Value` is ignored.

The list of allowed values for each (writable) setting can be retrieved with `int ABCanonGetList(ABCAMERA_HANDLE hCH, int ImgSetting, uint32_t pList[], size_t ListSize)`. The return value is the number of values returned in the list, or `ABC_RETVAL_BUFFER_TOO_SMALL` if `ListSize` is too small.
Since the size of the list is not necessary known, you can either allocate a bigger than necessary buffer, pass it to the function and use the return value to limit the actual list size, or you can call this function first with `pList = NULL` (`ListSize` will be ignored) to query for the actual list size (returned value) and then call this function again with a correct sized list.
Note that if you call first `ABCanonGetList()` to query for the size of the list (with `pList[] = NULL`) and then call `ABCanonGetList()` again with a buffer sized with the value returned by the previous call, you should anyway check the returned value because the list size may have changed in between the two calls (and so you can get fewer items if the list shrunk, or `ABC_RETVAL_BUFFER_TOO_SMALL` if the list grew).

2.5 Step #5: Image shooting

To take a picture, call `int ABCanonShoot(ABCAMERA_HANDLE hCH)`. Unfortunately, it is not always possible to take a picture using an USB command. In these cases the `ABCanonShoot()` function returns an error (`ABC_RETVAL_DENIED`). It is also possible call the `int ABCanonIsShootingAllowed(ABCAMERA_HANDLE hCH, int *Reason)` function to explicitly query for the reason why the shot is not allowed. Currently supported reasons are:

- Mirror lock is enabled (`ABC_SHOOTINGDENIED_MIRROR_LOCK`)
- Self timer is active (`ABC_SHOOTINGDENIED_SELF_TIMER`)

- Shutter speed/Mode dial is set to BULB (`ABC_SHOOTINGDENIED_BULB`)

There may be other conditions not currently known when shooting is not allowed. In these cases it's likely that the `ABCanonShoot()` function will return a Canon untranslated error (`0x80??????`).

2.6 Miscellaneous functions

Other miscellaneous functions that can be useful are (refer to the function reference chapter for a complete description of the parameters and returned values):

- `ABCanonGetVersion()`: returns the library version and build number
- `ABCanonGetSerialNumber()`: returns the camera serial number (to be interpreted as an unsigned 32 bit long number)
- `ABCanonGetOwnerName()`: returns the camera owner name.
- `ABCanonGetPictureTypes()`: returns a list of image types (Fine/Normal/Raw) with their resolution and a text describing the image type itself.
- `ABCanonGetPQImageCount()`: returns the number of images to be downloaded for a given picture quality (normally 1, but 2 if it's one of the RAW+JPG formats).
- `ABCanonGetCameraClock()` and `ABCanonSetCameraClock()` to get/set the camera internal clock
- `ABCanonGetBatteryStatus()`: returns the battery level. Note that if the level changes, an event is triggered too (so you can use the event call back routine or the event queue instead of continuously calling this function).
- `ABCanonGetRemainingShots()`: returns the number of available shots on the CF. Note that the camera limits the displayed number to 999, while the returned value may be higher.
- `ABCanonGetCFStatus()`: returns the total size and the free space of the CF in kbytes. If no CF is inserted, both values will be zero.
- `ABCanonGetCameraName()`: returns the Canon camera name ("Canon EOS 350D DIGITAL" for example).
- `ABCanonGetFWVersion()`: returns the camera firmware version.

2.7 Library options

Some library behaviors can be changed based on the user needs using the `ABCanonSetOptions(ABCAMERA_HANDLE hCH, uint32_t NewOptions)` and the `ABCanonGetOptions()` function.
Available options are:

- `ABC_OPTIONS_BULB_IN_LIST`: (350D only, ignored otherwise). The list for the shutter speed allowed values returned by the camera does never contain the BULB value (`ABC_SHUTTERSPEED3_BULB`), even if the BULB setting CAN be selected both from the camera and the PC (by calling `ABCanonSetImgSetting(..., ABC_SETTYPE_SET, ABC_SHUTTERSPEED3_BULB)`. If this option is set, the library

will add the BULB value to the shutter speed list when appropriate and BULB can be selected with any ABC_SETTYPE_xxx option. If this option is not set, the list returned will be exactly the list returned by the camera (and BULB can then only be selected with ABC_SETTYPE_SET).

- ABC_OPTIONS_HANDLE_THREAD_BUSY: With this option set, calling a library function within a callback function will cause the function to fail with ABC_RETVAL_THREAD_BUSY if the call **may** cause deadlocking. See *Threading* on page 20 for more information.
- ABC_OPTIONS_STANDARD_BULB: (5D only, ignored otherwise) When the Canon 5D mode dial is set to BULB (ABC_DIALMODE_BULB), force the shutter speed to return BULB (ABC_SHUTTERSPEED3_BULB) too. The camera will otherwise return meaningless (random?) values. This can be useful to check when the camera is in BULB mode in a camera independent way.
- ABC_OPTIONS_ALLOW_OVERWRITE: When saving the image to disk, it is normally an error if the target file already exists. Setting this option allows the old file to be overwritten without any error.
- ABC_OPTIONS_PERMANENT_OVERRIDE: The filename set with the ABCanonSetSaveOverrideA/W() will normally be used only once. Setting this option will keep the override filename active until the override itself is cleared. Not that removing this option will NOT automatically clear the override filename.
- ABC_OPTIONS_BUSYPOLL: The library will generate an ABC_EVENT_BUSY (begin) event every time the camera returns a ABC_RETVAL_BUSY error, then it will start polling the camera and will finally generate a ABC_EVENT_BUSY (end) event when the camera is no longer busy. Previous library versions required the user code to poll the camera to detect this condition.
- ABC_OPTIONS_PCT2: Additional information are passed in the (otherwise NULL) EventPtr pointer in the ABC_EVENT_DOWNLOADING event.

2.8 Notifications/events

The camera will notify an event when there is some change in the camera settings or in the camera status, when a picture has been taken or when some other event occurs.
There are two ways to handle these events, either by registering a call-back function, or by letting the library put all events in a queue and reading that queue. Both methods will be described below.
The first method (using a call back) will not automatically exclude the second one (the queue) but gives you the flexibility to select which events will be inserted in the queue and which will be discarded. Moreover, only events marked with "A" in the event list below will be normally put in the queue.

The notified events will have this structure:

```
struct sABCEvent
{
 uint32_t     EventID;
 uint32_t     EventSize;
 void        *EventPtr;
};
```

The fields of this structure are:

`EventID` The event identification

`EventSize` A parameter related to the event size. For a more specific description, refer to the event list below.

`EventPtr` Data associated to the event. For a more specific description, refer to the event list.

2.8.1 Event list

EventID	Description	Rv
`ABC_EVENT_BATTERY_STATUS_CHANGED (1)`	The battery status (level) has changed. `EventSize`: New battery level `EventPtr`: Unused. **NOTE**: it occurred to me a couple of time during testing that the camera generated an error while shooting an image (like, for example, an ERR 05). In all these cases the camera locked up reporting a LOW BATTERY event to the PC, even if the battery was still fully charged.	A
`ABC_EVENT_SETTINGS_CHANGED (2)`	Some image setting has changed. There is no indication about what setting changed. `EventSize`: Unused `EventPtr`: Unused	A
`ABC_EVENT_SHOT_COMPLETED (3)`	A shot has been completed. It happens at the END of the shot. Just an informational event, occurs only once regardless of the picture quality setting, and after the ImgCounter variable has been incremented (so you can change it if you want). You can also call the `ABCanonSetSaveOverrideA/W()` function to set an explicit filename to be used for this image only. `EventSize`: The "session" image number `EventPtr`: Unused	B
`ABC_EVENT_JPG_READY (4)`	The JPG image is ready to be downloaded. This event occurs only if the picture quality setting is not **RAW**. If the return value from the call back function is zero, the image will be discarded! `EventSize`: The "session" image number	C

EventID	Description	Rv
	EventPtr: Unused	
ABC_EVENT_CR2_READY (5)	The RAW image is ready to be downloaded. This event occurs only if the picture quality setting is **RAW** or any **RAW+JPG**. If the return value from the call back function is zero, the image will be discarded! EventSize: The "session" image number EventPtr: Unused	C
ABC_EVENT_DOWNLOADING (6)	Download progress percentage, 0% to 100%. In case of **RAW+JPG** this event occurs twice (0 to 100 for the CR2 and then 0 to 100 again for the JPG) EventSize: 0 to 100 EventPtr: Unused. When option ABC_OPTIONS_PCT2 has been set, EventPtr will be a pointer to a struct sDownloadPct variable (see note on page *15* below)	B
ABC_EVENT_JPG_SAVED (7)	The JPG image has been saved, accordingly to the save filename set with either ABCanonSetSaveFilenameA/W() or ABCanonSetSaveOverrideA/W(). This event occurs only if the download mode has been set to ABC_IMGDL_SAVE_TO_FILE or ABC_IMGDL_OVERRIDE_OR_DISCARD. EventSize: File size EventPtr: File name (WSTR)	A
ABC_EVENT_CR2_SAVED (8)	The CR2 image has been saved, accordingly to the save filename set with either ABCanonSetSaveFilenameA/W() or ABCanonSetSaveOverrideA/W(). This event occurs only if the download mode has been set to ABC_IMGDL_SAVE_TO_FILE or ABC_IMGDL_OVERRIDE_OR_DISCARD. EventSize: File size EventPtr: File name (WSTR)	A
ABC_EVENT_JPG_SAVE_FAILED (9)	The JPG image was not saved because of some error (the error itself is not reported). This event occurs only if the download mode has been set to ABC_IMGDL_SAVE_TO_FILE or ABC_IMGDL_OVERRIDE_OR_DISCARD. EventSize: Image size	A

EventID	Description	Rv
	EventPtr: Image data	
ABC_EVENT_CR2_SAVE_FAILED (10)	The CR2 image was not saved because of some error (the error itself is not reported). This event occurs only if the download mode has been set to ABC_IMGDL_SAVE_TO_FILE or ABC_IMGDL_OVERRIDE_OR_DISCARD. EventSize: Image size EventPtr: Image data	A
ABC_EVENT_JPG_IMAGE (11)	The JPG image has been downloaded from the camera and is now in memory. This event occurs only if the download mode has been set to ABC_IMGDL_SAVE_TO_MEMORY. EventSize: Image size EventPtr: Image data	A
ABC_EVENT_CR2_IMAGE (12)	The CR2 image has been downloaded from the camera and is now in memory. This event occurs only if the download mode has been set to ABC_IMGDL_SAVE_TO_MEMORY. EventSize: Image size EventPtr: Image data	A
ABC_EVENT_AF_FAILED (13)	A shot has been requested but not executed because an AF lock was not possible EventSize: Unused EventPtr: Unused	A

EventID	Description	Rv
ABC_EVENT_CAMERA_DISCONNECTED (14)	The camera has been disconnected either by disconnecting the USB cable or powering the camera off EventSize: Unused EventPtr: Unused	A
ABC_EVENT_JPG_DISCARDED (15)	The JPG image has been discarded by the library. EventSize: Unused EventPtr: Unused	A
ABC_EVENT_CR2_DISCARDED (16)	The CR2 image has been discarded by the library. EventSize: Unused EventPtr: Unused	A
ABC_EVENT_BUSY (17)	EventSize: 1 Busy, 0 No longer Busy EventPtr: Unused. Requires ABC_OPTIONS_BUSYPOLL.	A
ABC_EVENT_ALL_DONE (18)	All shoot related operations have been completed. Note that this event does not occur if no images are saved by the camera (discarded images are ok, saving to CF without any CF is not). EventSize: Unused EventPtr: Unused	A

EventPtr for the ABC_EVENT_DOWNLOADING event with the ABC_OPTIONS_PCT2 option will be a pointer to the following structure:

```
struct sDownloadPct
{
 uint32_t    TotalImages;
 uint32_t    CurrentImage;
 uint32_t    FullPct;
};
```

The fields of this structure are:

TotalImages	Total images to download for this shot: 1 or 2 in case of RAW+JPG
CurrentImage	0/1. The current image
FullPct	The 0-100% percentage of the full download. Does always 0-100% regardless of the number of images to download.

2.8.2 Event call back function

This method is just a bit more complex than the event queue but it is more powerful because not all events will be queued (the download percentage, for example).

Call int ABCanonRegisterNotificationCB() to register the notification call back. Passing NULL as the call back function will disable any notification.

The registered function will be called (from a background thread) every time there is a new event. See *Threading* below for implications of using a different thread.

The call back function has the following prototype (`ABC_NOTIFICATION_CB`):
`int _stdcall cbFunction (ABCAMERA_HANDLE Handle, struct sABCEvent *pEvent)`
`pEvent` is a pointer to the event structure defined above, containing the event ID and the event data.
The meaning of the return value of this call back function depends on the "Rv" column of the event list table above:

A Event can be discarded (and not inserted in the event queue) by returning 1
B The return value will be ignored and the event will never be inserted in the event queue
C The picture taken can be discarded by returning 0 (note that the image will be downloaded anyway and then discarded). The event itself will never be inserted in the event queue.

So a generic rule if you decide to use just the call back function is to return 1 to all events. By doing so the event queue will stay empty.

Any buffer/memory allocated by the library and passed as `EventPtr` to the call back function will automatically be released upon returning.

2.8.3 The event queue

The second method to handle events is to read them from the event queue using the `int ABCanonGetEvent(ABCAMERA_HANDLE hCH, struct sABCEvent *pEvent, uint32_t Timeout)` function, which waits `Timeout` milliseconds for an event to occur then returns `ABC_RETVAL_OK` if there is an event or `ABC_RETVAL_TIMEOUT` if the timeout lapsed without any event (but other errors can occur too, for example in case of invalid parameters or internal errors). `Timeout` can also be 0 (does not wait at all) or -1 (waits indefinitely).

If no call back function has been defined, all events will be queued except the ones marked "B" and "C" in the table events above. If there is a call back function, queued events depends on the call back function.

Any buffer/memory allocated by the library to hold the event itself and any possible event data should be released by calling `ABCanonReleaseEvent(struct sABCEvent *pEvent)` when the event data is no longer needed.

2.9 Raw CANON notifications

While the most important notifications can be handled via the call back function or the event queue described above, there can also be some other events or event parameter that are currently ignored.
If you want, therefore, you can have a call back function (this too, from another thread) with exactly the event notification as it came from the camera.

Create a function with a prototype like `void _stdcall MyRAWNotificationFunction (ABCAMERA_HANDLE Handle, uint16_t EventID, uint16_t Severity, uint8_t *pData, int DataSize)` and register it by calling `int ABCanonRegisterRawNotificationCB(ABCAMERA_HANDLE hCH, ABC_RAW_NOTIFICATION_CB lNotificationFunction)`

2.10 Saving the image

Saving the image is not a simple task because you have to handle also the cases where the image has been taken out of your control: you normally expect the image to be taken with the `ABCanonShoot()` function or using some kind of remote shooting device, but you will get images also when the user press the shutter button or when the remote shooting device has a spurious activation like it may happen when using the serial port DTR, for example.
And these "spurious" images are a problem, because when the camera reports that there is an image ready, the library has to download it, otherwise the camera will freeze with a "BUSY" indication on the display.
The way the image is downloaded from the camera and saved is selected with `ABCanonSetDownloadMode()`. There are 4 available modes:

- `ABC_IMGDL_DISCARD`: The image will be downloaded and then discarded. You can use this option if you don't want/expect any image.
- `ABC_IMGDL_SAVE_TO_MEMORY`: The image will be downloaded to a memory buffer and then passed to the application. You can then decide if save it to disk, process it, or discard it.
- `ABC_IMGDL_SAVE_TO_FILE`: In this mode the image will be automatically saved with a schema like this:

  `<pathname\basename><ImageCounter>.<Extension>`

  The `<Extension>` part will be automatically set to the correct image type (JPG or CR2), while `<pathname\basename>` and the number of digits in `<ImageCounter>` can be set with the `int ABCanonSetSaveFilenameA (ABCAMERA_HANDLE hCH, char *FileName, int DigitsCount)` function (or its `ABCanonSetSaveFilenameW()` widechar counterpart).
  When either one of these functions is called, the destination directory will be checked to see if there are already images present, and the starting number for `<ImageCounter>` will be automatically set to the correct value (last image number + 1).
  A few other notes:
  - If no filename is set, the image will be discarded
  - The target directory will be read (to set to the correct `<ImageCounter>`) during the call to `ABCanonSetSaveFilenameA / ABCanonSetSaveFilenameW` function **ONLY**.
  - The number will be incremented just after having saved an image and ONLY if using the filename set with `ABCanonSetSaveFilenameA / ABCanonSetSaveFilenameW`.
  - You can get/change this number with `ABCanonGetImageCounter() / ABCanonSetImageCounter()`.

    - The save operation will never overwrite a file (but you can change this with the ABC_OPTIONS_ALLOW_OVERWRITE option).
    - If you set multiple cameras to the same directory and naming schema things are not going to work.
    - If you set both the normal filename (with ABCanonSetSaveFilenameA / ABCanonSetSaveFilenameW) and an override filename (with ABCanonSetSaveOverrideA / ABCanonSetSaveOverrideW) the override filename will be used (and the number for the standard filename will not be incremented).
    - The override filename will be used only once (but you can change this with the ABC_OPTIONS_PERMANENT_OVERRIDE option).
- ABC_IMGDL_OVERRIDE_OR_DISCARD: In this mode the image will be saved only if an override filename has been set (with ABCanonSetSaveOverrideA / ABCanonSetSaveOverrideW) or discarded otherwise.
    - The normal filename set with ABCanonSetSaveFilenameA / ABCanonSetSaveFilenameW will not be used in this mode.
    - The override filename will be used only once (but you can change this with the ABC_OPTIONS_PERMANENT_OVERRIDE option).
    - The save operation will never overwrite a file (but you can change this with the ABC_OPTIONS_ALLOW_OVERWRITE option).
    - You can clear the override filename by passing NULL as the filename to the ABCanonSetSaveOverrideA / ABCanonSetSaveOverrideW function.

So, to summarize:

- If you want to use a simple naming schema with a fixed part and a number (just like the camera does on the CF), use ABC_IMGDL_SAVE_TO_FILE and set the filename with ABCanonSetSaveFilenameA / ABCanonSetSaveFilenameW.
- If you want to use a totally different naming schema, and you want to save all the images (also the ones that you don't expect), you can either use ABC_IMGDL_SAVE_TO_MEMORY and handle the saving by yourself or you can set an override filename during the ABC_EVENT_SHOT_COMPLETED event callback and let the library handle the saving.
- If you don't want to save the images you don't expect, use ABC_IMGDL_OVERRIDE_OR_DISCARD and set an override filename just before taking the picture.

2.11 Errors

You should not encounter errors in the normal operations of the library, except for the ABC_RETVAL_TIMEOUT value for the ABCanonGetEvent() function and a few other specific errors.
One error that is not really an error and that should be specifically handled is ABC_RETVAL_BUSY. This error is returned by some functions, specifically those that set or get settings to or from the camera, when the function has to talk to the camera because buffered data are

no longer up to date and the function itself is called while the menu is displayed on the camera screen. In this case there is no possibility to communicate with the camera, so a good thing to do is to notify this condition to the user (so he/she can exit the camera menu) and to start polling the camera (asking for any image setting, for example) to discover when the menu has been exited (unfortunately there is no specific notification for the menu enter/exit, and you cannot count on the `ABC_EVENT_SETTINGS_CHANGED` event occurring on menu exit because it happens only when an image setting has been changed while in the menu).

For a complete list of possible errors and cause, se the *Error reference* chapter below.

2.12 Debugging aids

The library has a "stdout-like" output that can be useful to debug issues that may happen during development. The `ABUTestDll` program uses this interface to output debugging messages to the console.
To get this output, just create a function with this prototype: `void _stdcall myDebugCb (ABCAMERA_HANDLE Handle, void *Msg, uint32_t MsgSize)` and pass it to `int ABCDebugSetStdout(int Flags, ABC_DEBUG_STDOUT_CB StdOutFn)`.
`Flags` is a parameter which controls how the debug text is passed to the function:

- `ABC_STDOUT_LF_ONLY`: line breaks are LFs (0x0A) only
- `ABC_STDOUT_CR_LF`: line breaks are CRs (0x0D) followed by LFs (0x0A)
- `ABC_STDOUT_ANSI`: text is 8-bit ASCII
- `ABC_STDOUT_WIDE`: text is widechar.

The parameters passed to the call back functions are:

- `Handle`: Camera handle this text refers to. Note that this handle CAN be `NULL` when the text does not refer to a camera
- `Msg`: the ANSI or WIDECHAR text (depending on setup) to be displayed. Both types are 0-terminated.
- `MsgSize`: The size (in characters) of `Msg`, not counting the ending '\0'.

Note that internally the debugging text is always 7-bit ASCII, except for the filenames which are converted to ANSI from the internal WIDECHAR values.
This means that:

1. You can ignore codepages and the output should be always readable
2. Filenames MAY be displayed incorrectly, but will be correctly used.

This debugging feature is also not meant to be used in a release build of the final software, because it may have problem if used from different processes.

Another couple of debugging functions are:

- `ABCDebugMemoryUsed(0)`: returns the memory (in bytes) used by the library. It should start at about 7K and not significantly increase in "idle" conditions (i.e. with an empty queue and no buffered images).
- `ABCDebugDumpHandle()`:Dumps the content of a camera handle to stdout.

2.13 Text Helper functions

The library has also a few functions to help translating camera setting values to a readable text. These function are quite basic and not sophisticated at all.

- `ABCanonGetImgSettingText()`: Similar to `ABCanonGetImgSetting()` but returns a text instead of a value
- `ABCanonImgSettingValueToText()`: "static" function (does not refer to a camera handle) to translate a setting value to its corresponding text
- `ABCanonGetBatteryStatusText()`: Return the text describing the current battery level (in English) but can also be used to translate a generic battery value to the corresponding text.

2.14 Threading

All functions are thread safe, since there is an internal mutex protecting the low level communication with the camera.
There are anyway a couple of issues to know.

1. Call backs are done on different threads created by the library, and this means that .NET events raised from this call backs requires an Invoke() call to handle the event in the UI thread. You can use the code in `ABUtilites` as an example on how to do so. Handling the call backs in a different language may require different precautions.
2. When the call back function is called, the internal mutex is normally not owned, so there are no problems in handling the call back. This is valid both for the standard notification call back (`ABC_NOTIFICATION_CB`) and for the raw notification call back (`ABC_RAW_NOTIFICATION_CB`).
   There is only one exception to this and it is when the event ID is `ABC_EVENT_DOWNLOADING`: during this call back the internal mutex is owned by the thread which is downloading the image (and calling the call back function). This requires special attentions because deadlocking may otherwise occur (a typical scenario could be this: you receive the `ABC_EVENT_DOWNLOADING` event, and therefore `Invoke()` to the UI thread to update a download progressbar but the UI thread, for some timer tick or for user input calls any `ABCanon...()` function that requires the mutex).
   Version 2.03 of the library uses an helper thread to deliver some of the notifications, avoiding deadlocks altogether. "Partial" deadlocks may still occur, but since these will not block the downloading thread, the download will complete anyway, though some `ABC_EVENT_DOWNLOADING` notifications may be lost.
   You may try to avoid this situation by activating the `ABC_OPTIONS_HANDLE_THREAD_BUSY` option. With this option active, you will get the `ABC_RETVAL_THREAD_BUSY` return value from any function that may deadlock. Note that in complex multithread scenarios you may also have false positives (function that return this error when deadlocking is NOT going to occur) and it's for this reason that this option is not active by default.

3 Alphabetical function reference

3.1 ABCanonBuildCameraList

Build and returns the list of supported cameras connected to the system

Syntax:

```
int ABCAPI ABCanonBuildCameraList(
  struct sCameraData **ppCameraData
);
```

Parameters:

`ppCameraData`
: A pointer to a struct sCameraData that receives the list of supported cameras connected to the system.

Return value:

The number of supported cameras found or an error

Remarks:

The memory allocated by the library and returned in the `ppCameraData` pointer should be released by calling `ABCanonReleaseCameraList()`

3.2 ABCanonCheckImgSettingWritable

Check if an image setting is writable or read only in the current camera mode.

Syntax:

```
int ABCAPI ABCanonCheckImgSettingWritable(
  ABCAMERA_HANDLE hCH,
  int ImgSetting
);
```

Parameters:

`hCH`
: The camera handle

`ImgSetting`
: The image setting to retrieve, one of:
`ABC_IMGSETTING_MODE_DIAL`
`ABC_IMGSETTING_SHUTTER_SPEED`
`ABC_IMGSETTING_APERTURE`

ABC_IMGSETTING_ISO
ABC_IMGSETTING_EXP_COMPENSATION
ABC_IMGSETTING_METERING
ABC_IMGSETTING_AUTOFOCUS
ABC_IMGSETTING_WHITE_BALANCE
ABC_IMGSETTING_PICTURE_QUALITY
ABC_IMGSETTING_FLASH_EXP_COMP
ABC_IMGSETTING_AUTO_POWER_OFF
ABC_IMGSETTING_DRIVE_MODE
ABC_IMGSETTING_AEB
ABC_IMGSETTING_COLOR_TEMPERATURE

Return value:

Not error return values are ABC_RETVAL_OK if the setting is writable, or ABC_RETVAL_INVALID_PARM if it is not. Other errors can be returned too.

3.3 ABCanonConnectToCamera

Connect to a camera and returns an handle that can be used to operate the camera

Syntax:

```
int ABCAPI ABCanonConnectToCamera(
  int ConnectMethod,
  int Filter,
  uint32_t Parm,
  ABCAMERA_HANDLE *phCH
);
```

Parameters:

ConnectMethod

How to select the camera to connect to. Available methods are:

- CONNECT_FIRST_AVAILABLE: connect the first camera available (cameras already in use are not counted).
- CONNECT_BY_INDEX: connect to the nth camera available.
- CONNECT_BY_SERIAL_NUMBER: connect to the camera with a specific serial number.

Filter

Filter only the cameras of a certain type. Currently available values are ABCTYPE_350D, ABCTYPE_5D and ABCTYPE_ALL (no filtering).

`Parm`
: An auxiliary parameter for the connection. Unused when `ConnectionMethod` is `CONNECT_FIRST_AVAILABLE`, it is the camera index for `CONNECT_BY_INDEX` and the camera serial number for `CONNECT_BY_SERIAL_NUMBER`.

`phCH`
: A pointer to a `ABCAMERA_HANDLE` variable that receives the handle to the camera.

Return value:

The error code. When the return value is not `ABC_RETVAL_OK` the camera has NOT been opened.

Remarks:

Use `ABCanonDisconnectFromCamera()` to disconnect from the camera and release the handle.

3.4 ABCDebugDumpHandle

Dumps the content of a camera handle to stdout

Syntax:

```
void ABCAPI ABCDebugDumpHandle(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
: The camera handle

Return value:

The error code.

Remarks:

Output is done to stdout, so you have to have set a stdout call back function with `ABCDebugSetStdout()` to see it.

3.5 ABCDebugMemoryUsed

Returns the memory used by the library

Syntax:

```
int ABCAPI ABCDebugMemoryUsed(
  int Reserved
);
```

Parameters:

`Reserved`
: Reserved parameter. Set to 0.

Return value:

The memory used in bytes, or the error code.

Remarks:

See *Debugging aids* on page 19 for more information.

3.6 ABCDebugSetStdout

Sets the call back function for the library stdout text.

Syntax:

```
int ABCAPI ABCDebugSetStdout(
  int Flags,
  ABC_DEBUG_STDOUT_CB StdOutFn
);
```

Parameters:

`Flags`
: StdOut options:
  - `ABC_STDOUT_LF_ONLY`: Line breaks are LFs (0x0A) only
  - `ABC_STDOUT_CR_LF`: Line breaks are CRs (0x0D) + LFs (0x0A)
  - `ABC_STDOUT_ANSI`: Text is 8-bit ASCII
  - `ABC_STDOUT_WIDE`: Text is widechar.

`StdOutFn`
: The call back function (`ABC_DEBUG_STDOUT_CB`)

Return value:

The error code.

Remarks:

The call back function has the following prototype:

```
void _stdcall StdOutFunction(
  ABCAMERA_HANDLE Handle,
  void *Msg,
  uint32_t MsgSize
);
```
See *Debugging aids* on page 19 for more information.

3.7 ABCanonDisconnectFromCamera

Disconnects from a camera.

Syntax:

```
int ABCAPI ABCanonDisconnectFromCamera(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
: The camera handle

Return value:

The error code.

Remarks:

The disconnect operation release all the resources allocated by the library but also reprogram the camera to save the image to the CF and not to the PC. The Canon library did this too, but I have no idea on what happens if you just pull the USB connection.

3.8 ABCanonGetAllSettings

Get all camera settings at one time

Syntax:

```
int ABCAPI ABCanonGetAllSettings(
  ABCAMERA_HANDLE hCH,
  int32_t *pSettingsArray,
  int SettingsArraySize
);
```

Parameters:

`hCH`
The camera handle
`pSettingsArray`
A pointer to user allocated memory to receive the settings
`SettingsArraySize`
The `pSettingsArray` size in items (not bytes)

Return value:

The number of items actually used (which is `SettingsArraySize` at maximum) or an error code.

Remarks:

You can use `ABC_IMGSETTING_...` (see `ABCanonGetImgSetting()` below) as an index in `pSettingsArray`. If `SettingsArraySize` is smaller than required only `SettingsArraySize` items will be compiled.

3.9 ABCanonGetBatteryStatus

Returns the current battery status.

Syntax:

```
int ABCAPI ABCanonGetBatteryStatus(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
The camera handle

Return value:

The battery level that can be either `ABC_BATTERY_NORMAL`, `ABC_BATTERY_WEAK`, `ABC_BATTERY_LOW` or `ABC_BATTERY_LB`.

Remarks:

Changes in the battery level are also notified and put in the event queue.

3.10 ABCanonGetBatteryStatusText

Return the current battery level in a readable text format, or converts a generic battery status code to a readable text format.

Syntax:

```
char * ABCAPI ABCanonGetBatteryStatusText(
  ABCAMERA_HANDLE hCH,
  int BattLevel
);
```

Parameters:

`hCH`
: The camera handle. Can be `NULL` if the function is called to convert a generic battery status code to text. In this case, `BattLevel` cannot be `-1`.

`BattLevel`
: Use `-1` to return the text for the current camera level, or the battery status code to be converted.

Return value:

A pointer to a constant ASCII text with the readable version for the battery status or NULL in case of error.

Remarks:

It is an error to have both `hCH == NULL` and `BattLevel == -1`.

3.11 ABCanonGetCameraClock

Returns the camera internal clock.

Syntax:

```
int ABCAPI ABCanonGetCameraClock(
  ABCAMERA_HANDLE hCH,
  time_t *pTime
);
```

Parameters:

`hCH`
: The camera handle

`pTime`
: A pointer to receive the current camera clock.

Return value:

The error code.

Remarks:

The clock is in "unix" time (for example, 0x562823AC is Oct 21st 2015, 23:45:48)

3.12 ABCanonGetCameraName

Gets the Canon camera name ("Canon EOS 350D DIGITAL" for example)

Syntax:

```
char * ABCAPI ABCanonGetCameraName(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
: The camera handle

Return value:

A pointer to an internal buffer containing the camera name as an ASCIIz string, or `NULL` in case of error (invalid handle).

3.13 ABCanonGetCameraType

Gets the camera type

Syntax:

```
int ABCAPI ABCanonGetCameraType(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
: The camera handle

Return value:

The camera type (see the `ABCTYPE_xxxx` values) or an error.

3.14 ABCanonGetCFStatus

Gets the size and the available space on the compact flash.

Syntax:

```
int ABCAPI ABCanonGetCFStatus(
  ABCAMERA_HANDLE hCH,
  uint32_t *TotalKB,
  uint32_t *FreeKB
);
```

Parameters:

`hCH`
: The camera handle

`TotalKB`
: A pointer to receive the Compact Flash size in KiloBytes.

`FreeKB`
: A pointer to receive the Compact Flash free space in KiloBytes.

Return value:

The error code.

Remarks:

If no Compact Flash is inserted into the camera, both values will be 0.

3.15 ABCanonGetCustomFunction

Get a "Custom function" setting from the camera. This function replaces the `ABCanonGetCustomOption()` function which is now deprecated.

Syntax:

```
int ABCAPI ABCanonGetCustomFunction(
  ABCAMERA_HANDLE hCH,
  int CustomFunctionNum
);
```

Parameters:

`hCH`
: The camera handle

`CustomFunctionNum`
: The custom function to get, one of:
`ABC_CF_SETBUTTON_FUNCT`
`ABC_CF_LONGEXP_NOISE_RED`

```
ABC_CF_FLASH_SYNC
ABC_CF_SHUTTER_AE_LOCK_BTN
ABC_CF_AF_ASSIST_BEAM
ABC_CF_EXP_LEVEL_INCREMENT
ABC_CF_FLASH_FIRING
ABC_CF_ISO_EXPANSION
ABC_CF_AEB_SEQUENCE
ABC_CF_SUPERIMPOSED_DISPLAY
ABC_CF_MENU_BUTTON_DISPLAY
ABC_CF_MIRROR_LOCKUP
ABC_CF_AF_POINT_SELECTION
ABC_CF_E_TTL_II
ABC_CF_SHUTTER_CURTAIN
ABC_CF_SAFETY_SHIFT
ABC_CF_AF_POINT_ACTIVATION
ABC_CF_RETURN_TO_SHOOT
ABC_CF_LENS_AF_STOP_BUTTON
ABC_CF_ADD_ORIGINAL_DATA
```

Return value:

The error code, which will be `ABC_RETVAL_INVALID_PARM` if the requested custom function is not defined for the camera.

Remarks:

This function replaces and enhance the `ABCanonGetCustomOption()` introduced in version 1.02.
For compatibility reasons, `CustomFunctionNum` can also be a 1 to 9 number corresponding to the 350D custom functions numbers which will be translated to the corresponding custom function if the camera is a 5D.

3.16 ABCanonGetCustomOption

This function has been deprecated and replaced by the `ABCanonGetCustomFunction()`.

3.17 ABCanonGetEvent

Retrieve an event from the event queue

Syntax:

```
int ABCAPI ABCanonGetEvent(
  ABCAMERA_HANDLE hCH,
  struct sABCEvent *pEvent,
  uint32_t Timeout
);
```

Parameters:

`hCH`
: The camera handle

`pEvent`
: A pointer to receive the event

`Timeout`
: The timeout (in ms) to wait for an event. Can also be 0 (do not wait) or -1 (wait indefinitely)

Return value:

The error code. ABC_RETVAL_TIMEOUT means that the timeout lapsed and no event has been returned.

Remarks:

The returned event should be released with `ABCanonReleaseEvent()`.

3.18 ABCanonGetFWVersion

Gets the camera firmware version.

Syntax:

```
int ABCAPI ABCanonGetFWVersion(
  ABCAMERA_HANDLE hCH,
  uint8_t FWRelease[3]
);
```

Parameters:

`hCH`
: The camera handle

`FWRelease`
: A pointer to receive the 3 bytes firmware version

Return value:

The error code.

Remarks:

The 3 version bytes are high, medium and low, in order. So camera with firmware version, for example, 1.0.3, will have 1 in `FWRelease[0]`, 0 in `FWRelease[1]` and 3 in `FWRelease[2]`.

3.19 ABCanonGetImgSetting

Get an image setting from the camera.

Syntax:

```
int ABCAPI ABCanonGetImgSetting(
  ABCAMERA_HANDLE hCH,
  int ImgSetting
);
```

Parameters:

`hCH`
: The camera handle

`ImgSetting`
: The image setting to retrieve, one of:
`ABC_IMGSETTING_MODE_DIAL`
`ABC_IMGSETTING_SHUTTER_SPEED`
`ABC_IMGSETTING_APERTURE`
`ABC_IMGSETTING_ISO`
`ABC_IMGSETTING_EXP_COMPENSATION`
`ABC_IMGSETTING_METERING`
`ABC_IMGSETTING_AUTOFOCUS`
`ABC_IMGSETTING_WHITE_BALANCE`
`ABC_IMGSETTING_PICTURE_QUALITY`
`ABC_IMGSETTING_FLASH_EXP_COMP`
`ABC_IMGSETTING_AUTO_POWER_OFF`
`ABC_IMGSETTING_DRIVE_MODE`
`ABC_IMGSETTING_AEB`
`ABC_IMGSETTING_COLOR_TEMPERATURE`

Return value:

The requested setting or an error. Errors are values < 0, settings are >= 0

3.20 ABCanonGetImageCounter

Get the internal image counter, used to create the filename for automatic image saving.

Syntax:

```
int ABCAPI ABCanonGetImageCounter(
  ABCAMERA_HANDLE hCH,
  uint32_t *pCounter
);
```

Parameters:

`hCH`
: The camera handle

`pCounter`
: A pointer to receive the current image counter.

Return value:

The error code.

Remarks:

The counter is automatically set to the correct value during the call to `ABCanonSetSaveFilenameA / ABCanonSetSaveFilenameW`, based on the images already present in the directory.

3.21 ABCanonGetImgSettingText

Get an image setting as readable text.

Syntax:

```
char * ABCAPI ABCanonGetImgSettingText(
  ABCAMERA_HANDLE hCH,
  int ImgSetting
);
```

Parameters:

`hCH`
: The camera handle

`ImgSetting`
: The image setting to retrieve, one of:
`ABC_IMGSETTING_MODE_DIAL`
`ABC_IMGSETTING_SHUTTER_SPEED`
`ABC_IMGSETTING_APERTURE`
`ABC_IMGSETTING_ISO`
`ABC_IMGSETTING_EXP_COMPENSATION`
`ABC_IMGSETTING_METERING`
`ABC_IMGSETTING_AUTOFOCUS`

ABC_IMGSETTING_WHITE_BALANCE
ABC_IMGSETTING_PICTURE_QUALITY
ABC_IMGSETTING_FLASH_EXP_COMP
ABC_IMGSETTING_AUTO_POWER_OFF
ABC_IMGSETTING_DRIVE_MODE
ABC_IMGSETTING_AEB
ABC_IMGSETTING_COLOR_TEMPERATURE

Return value:

A pointer to a constant ASCII text with the readable version for the requested setting or NULL in case of error.

Remarks:

This function is a helper function that can be used to ease debugging and it is not meant to be directly used because all returned values are in English and will not be localized.

3.22 ABCanonGetList

Get a list of admitted values for an image setting

Syntax:

```
int ABCAPI ABCanonGetList(
  ABCAMERA_HANDLE hCH,
  int ImgSetting,
  uint32_t pList[],
  size_t ListSize
);
```

Parameters:

`hCH`
: The camera handle

`ImgSetting`
: The image setting to retrieve, one of:
ABC_IMGSETTING_MODE_DIAL
ABC_IMGSETTING_SHUTTER_SPEED
ABC_IMGSETTING_APERTURE
ABC_IMGSETTING_ISO
ABC_IMGSETTING_EXP_COMPENSATION
ABC_IMGSETTING_METERING
ABC_IMGSETTING_AUTOFOCUS
ABC_IMGSETTING_WHITE_BALANCE
ABC_IMGSETTING_PICTURE_QUALITY
ABC_IMGSETTING_FLASH_EXP_COMP

ABC_IMGSETTING_AUTO_POWER_OFF
ABC_IMGSETTING_DRIVE_MODE
ABC_IMGSETTING_AEB
ABC_IMGSETTING_COLOR_TEMPERATURE

`pList[]`
: Pointer to a user allocated array for the list item. Can be NULL when just requesting the list size.

`ListSize`
: The size (in items) of the memory pointed by `pList[]`. Ignored if `pList[]` is NULL.

Return value:

The number of items of the requested list or an error. `ABC_RETVAL_BUFFER_TOO_SMALL` is returned if the provided list is too small.

Remarks:

Note that if you call first this function to query for the size of the list (using `pList[] == NULL`) and then call it again with a buffer sized with the value returned by the previous call, you should anyway check the returned value because the list size may have changed in between the two calls (and so you can get fewer items if the list shrunk, or `ABC_RETVAL_BUFFER_TOO_SMALL` if the list grew).
A simpler way to use this function is to allocate a big list (100 items, for example) and then use the return value to limit the list.

3.23 ABCanonGetOptions

Get the current library options.

Syntax:

```
int ABCAPI ABCanonGetOptions(
  ABCAMERA_HANDLE hCH,
  uint32_t *pOptions
);
```

Parameters:

`hCH`
: The camera handle

`pOptions`
: A pointer to receive the current options.

Return value:

The error code.

Remarks:

See *Library options* on page 10 for more information.

3.24 ABCanonGetOwnerName

Returns the camera owner name.

Syntax:

```
char * ABCAPI ABCanonGetOwnerName(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
: The camera handle

Return value:

A pointer to an internal buffer containing the camera owner as an ASCIIz string, or `NULL` in case of error (invalid handle).

3.25 ABCanonGetPictureTypes

Returns a list of picture types the camera can take.

Syntax:

```
int ABCAPI ABCanonGetPictureTypes(
  ABCAMERA_HANDLE hCH,
  struct sImageTypes **pSIT
);
```

`hCH`
: The camera handle

`pSIT`
: A pointer that receives a pointer to a `struct sImageTypes` array containing the images types supported by the camera.

Return value:

The number of entries in the `struct sImageTypes` array, or an error.

Remarks:

`struct sImageTypes` is:

```
{
 char     Description[20];
 uint32_t Height;
 uint32_t Width;
 uint32_t Type;
};
```

- `Description` is an ASCIIz string containing the image type description;
- `Height` and `Width` are the dimensions in pixels;
- `Type` is one of `ABC_PT_FINE`, `ABC_PT_NORMAL`, `ABC_PT_RAW`.

The memory used for this array does not require to be freed.

3.26 ABCanonGetPQImageCount

Gets the number of images that will be downloaded for a given picture quality.

Syntax:

```
int ABCAPI ABCanonGetPQImageCount (
  ABCAMERA_HANDLE hCH,
  int32_t PictureQuality
);
```

Parameters:

`hCH`
: The camera handle

`PictureQuality`
: The picture quality or `-1` for the picture quality currently in use.

Return value:

The number of images that will be downloaded (1 or 2) or an error.

Remarks:

An error will be returned if the `PictureQuality` parameter is not valid value for the current camera configuration (for example, passing `ABC_PICTUREQUALITY_RAW` when the camera is in basic mode).

3.27 ABCanonGetRemainingShots

Gets the number of images that can be saved to the compact flash.

Syntax:

```
int ABCAPI ABCanonGetRemainingShots(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
: The camera handle

Return value:

The number of images that can be saved to the compact flash or an error.

Remarks:

The camera limits the displayed number to 999, while the returned value may be higher.

3.28 ABCanonGetSerialNumber

Returns the camera serial number.

Syntax:

```
uint32_t ABCAPI ABCanonGetSerialNumber(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
: The camera handle

Return value:

The camera serial number (as a 32 bit unsigned number) or 0 in case of error (invalid handle).

3.29 ABCanonGetVersion

Returns the library version and build number.

Syntax:

```
void ABCAPI ABCanonGetVersion(
  uint32_t *pVersionH,
  uint32_t *pVersionL,
  uint32_t *pBuild
);
```

Parameters:

`pVersionH`
: A pointer to a uint32_t variable that receives the major version number

`pVersionL`
: A pointer to a uint32_t variable that receives the minor version number

`pBuild`
: A pointer to a uint32_t variable that receives the build number

3.30 ABCanonGetWritableSettings

Returns an array of boolean values indicating what image settings are writable in the current camera mode.

Syntax:

```
int ABCAPI ABCanonGetWritableSettings(
  ABCAMERA_HANDLE hCH,
  int32_t *pSettingsArray,
  int SettingsArraySize
);
```

Parameters:

`hCH`
: The camera handle

`pSettingsArray`
: A pointer to user allocated memory to receive the settings

`SettingsArraySize`
: The `pSettingsArray` size in items (not bytes)

Return value:

The number of items actually used (which is `SettingsArraySize` at maximum) or an error code.

Remarks:

You can use `ABC_IMGSETTING_...` (see `ABCanonGetImgSetting()` above) as an index in `pSettingsArray`. If `SettingsArraySize` is smaller than required only `SettingsArraySize` items will be compiled. If a value in the array is 0, the corresponding setting is read only.

3.31 ABCanonImgSettingValueToText

Convert an image setting value to the corresponding text.

Syntax:

```
char * ABCAPI ABCanonImgSettingValueToText(
  int ImgSetting,
  int Value
);
```

Parameters:

`ImgSetting`

The image setting to retrieve, one of:
`ABC_IMGSETTING_MODE_DIAL`
`ABC_IMGSETTING_SHUTTER_SPEED`
`ABC_IMGSETTING_APERTURE`
`ABC_IMGSETTING_ISO`
`ABC_IMGSETTING_EXP_COMPENSATION`
`ABC_IMGSETTING_METERING`
`ABC_IMGSETTING_AUTOFOCUS`
`ABC_IMGSETTING_WHITE_BALANCE`
`ABC_IMGSETTING_PICTURE_QUALITY`
`ABC_IMGSETTING_FLASH_EXP_COMP`
`ABC_IMGSETTING_AUTO_POWER_OFF`
`ABC_IMGSETTING_DRIVE_MODE`
`ABC_IMGSETTING_AEB`
`ABC_IMGSETTING_COLOR_TEMPERATURE`

`Value`

The value to be converted to text.

Return value:

A pointer to a constant ASCII text with the readable version for the requested setting or NULL in case of error.

Remarks:

This function is a helper function that can be used to ease debugging and it is not meant to be directly used because all returned values are in English and will not be localized.

3.32 ABCanonIsShootingAllowed

Checks if shooting is allowed, or some camera setting prevents it.

Syntax:

```
int ABCAPI ABCanonIsShootingAllowed(
  ABCAMERA_HANDLE hCH,
  int *Reason
);
```

Parameters:

`hCH`
: The camera handle

`Reason`
: A pointer to receive whether the shoot is allowed or the reason it is not.

Return value:

The error code.

Remarks:

The returned reason can be:

- `ABC_SHOOTINGDENIED_ALLOWED` Shooting is actually allowed.
- `ABC_SHOOTINGDENIED_MIRROR_LOCK` Mirror lock is active.
- `ABC_SHOOTINGDENIED_SELF_TIMER` Self timer is active.
- `ABC_SHOOTINGDENIED_BULB` The shutter speed is set to BULB.

3.33 ABCanonRegisterNotificationCB

Register a call back function to receive notifications

Syntax:

```
int ABCAPI ABCanonRegisterNotificationCB(
  ABCAMERA_HANDLE hCH,
  ABC_NOTIFICATION_CB lNotificationFunction
);
```

Parameters:

`hCH`
The camera handle
`lNotificationFunction`
The call back function

Return value:

The error code.

Remarks:

The call back function has the following prototype:

```
int _stdcall Function(
  ABCAMERA_HANDLE Handle,
  struct sABCEvent *pABCEvent
);
```

Refer to Notifications/events on page 11 for more information about events.

3.34 ABCanonRegisterRawNotificationCB

Register a call back function to receive raw Canon notifications.

Syntax:

```
int ABCAPI ABCanonRegisterRawNotificationCB(
  ABCAMERA_HANDLE hCH,
  ABC_RAW_NOTIFICATION_CB lNotificationFunction
);
```

Parameters:

`hCH`
The camera handle
`lNotificationFunction`
The call back function

Return value:

The error code.

Remarks:

The call back function has the following prototype:

```
void _stdcall Function (
  ABCAMERA_HANDLE Handle,
  uint16_t CanonEventID,
  uint16_t Severity,
  uint8_t *pData,
  int DataSize
);
```

3.35 ABCanonReleaseCameraList

Releases the memory allocated by the `ABCanonBuildCameraList()` function.

Syntax:

```
void ABCAPI ABCanonReleaseCameraList(
  struct sCameraData *pCameraData
);
```

Parameters:

`pCameraData`

The pointer returned by the `ABCanonBuildCameraList()` function.

3.36 ABCanonReleaseEvent

Release an event returned by `ABCanonGetEvent()`.

Syntax:

```
void ABCAPI ABCanonReleaseEvent(
  struct sABCEvent *pEvent
);
```

Parameters:

`pEvent`

The event pointer returned by the `ABCanonGetEvent()` function.

3.37 ABCanonSetCameraClock

Sets the camera internal clock.

Syntax:

```
int ABCAPI ABCanonSetCameraClock(
  ABCAMERA_HANDLE hCH,
  time_t Time
);
```

Parameters:

`hCH`
: The camera handle

`Time`
: The time to set the camera

Return value:

The error code.

Remarks:

The clock is in "unix" time (for example, 0x562823AC is Oct 21st 2015, 23:45:48)

3.38 ABCanonSetDownloadMode

Selects what to do with the downloaded images.

Syntax:

```
int ABCAPI ABCanonSetDownloadMode(
  ABCAMERA_HANDLE hCH,
  int DownloadMode
);
```

Parameters:

`hCH`
: The camera handle

`DownloadMode`
: The destination of the downloaded image:
  - `ABC_IMGDL_DISCARD` Discard the image
  - `ABC_IMGDL_SAVE_TO_FILE` Save to a file, with the filename set with either `ABCanonSetSaveFilenameA/W()` or `ABCanonSetSaveOverrideA/W()`.
  - `ABC_IMGDL_SAVE_TO_MEMORY` Put the image in a memory buffer and pass it to the application.
  - `ABC_IMGDL_OVERRIDE_OR_DISCARD` Save to file if there is a filename set with `ABCanonSetSaveOverrideA/W()`, discard otherwise. Note that

filename set with `ABCanonSetSaveFilenameA/W()` are ignored in the download mode.

Return value:

The error code.

Remarks:

For this to work, `ABCanonSetSaveTarget()` should be set to PC o CF+PC.

3.39 ABCanonSetImageCounter

Set the internal image counter, used to create the filename for automatic image saving.

Syntax:

```
int ABCAPI ABCanonSetImageCounter(
  ABCAMERA_HANDLE hCH,
  uint32_t Counter
);
```

Parameters:

`hCH`
: The camera handle

`hCH`
: The value to set the image counter to.

Return value:

The error code.

Remarks:

The counter is automatically set to the first available value during the call to `ABCanonSetSaveFilenameA()` or `ABCanonSetSaveFilenameW()`, based on the images already present in the directory. So if you want to change the number, you should call this function AFTER having set the directory and the base filename.
If you want to set the value just before the image is saved, do it in the `ABC_EVENT_SHOT_COMPLETED` event.

3.40 ABCanonSetImgSetting

Changes an image setting

Syntax:

```
int ABCAPI ABCanonSetImgSetting(
  ABCAMERA_HANDLE hCH,
  int ImgSetting,
  int SetType,
  int Value
);
```

Parameters:

`hCH`

The camera handle

`ImgSetting`

The image setting to set, one of:

`ABC_IMGSETTING_MODE_DIAL`
`ABC_IMGSETTING_SHUTTER_SPEED`
`ABC_IMGSETTING_APERTURE`
`ABC_IMGSETTING_ISO`
`ABC_IMGSETTING_EXP_COMPENSATION`
`ABC_IMGSETTING_METERING`
`ABC_IMGSETTING_AUTOFOCUS`
`ABC_IMGSETTING_WHITE_BALANCE`
`ABC_IMGSETTING_PICTURE_QUALITY`
`ABC_IMGSETTING_FLASH_EXP_COMP`
`ABC_IMGSETTING_AUTO_POWER_OFF`
`ABC_IMGSETTING_DRIVE_MODE`
`ABC_IMGSETTING_AEB`
`ABC_IMGSETTING_COLOR_TEMPERATURE`

`SetType`

How to change the setting:

`ABC_SETTYPE_FIRST`	Change to lowest value
`ABC_SETTYPE_CHANGE`	Relative change of `Value` (with sign)
`ABC_SETTYPE_SET`	Set image setting to `Value`
`ABC_SETTYPE_LAST`	Change to highest value
`ABC_SETTYPE_PREV`	Switch to the previous value
`ABC_SETTYPE_NEXT`	Switch to the next value

`Value`

The relative change for `ABC_SETTYPE_CHANGE` and the value to set for `ABC_SETTYPE_SET`. Ignored for all others `SetType` values.

Return value:

The error code.

3.41 ABCanonSetOptions

Sets the library options.

Syntax:

```
int ABCAPI ABCanonSetOptions(
  ABCAMERA_HANDLE hCH,
  uint32_t NewOptions
);
```

Parameters:

`hCH`
: The camera handle

`NewOptions`
: The new library options.
See *Library options* on page 10 for more information.

Return value:

The error code.

Remarks:

See *Library options* on page 10 for more informations.

3.42 ABCanonSetSaveFilenameA / ABCanonSetSaveFilenameW

Sets the path/filename to use when download mode has been set to `ABC_IMGDL_SAVE_TO_FILE`.

Syntax:

```
int ABCAPI ABCanonSetSaveFilenameA(
  ABCAMERA_HANDLE hCH,
  char *FileName,
  int DigitsCount
);
```

or

```
int ABCAPI ABCanonSetSaveFilenameW(
  ABCAMERA_HANDLE hCH,
  LPWSTR FileName,
  int DigitsCount
);
```

Parameters:

`hCH`
: The camera handle

`FileName`
: The filename to save to, either ANSI for `ABCanonSetSaveFilenameA` or WIDECHAR for `ABCanonSetSaveFilenameW`.

`DigitsCount`
: The number of digits to use.

Return value:

The error code.

Remarks:

The path/filename to save to is:

`<pathname\basename><ImageCounter>.<Extension>`.

The `<Extension>` part will be automatically set to the correct image type (JPG or CR2), while `FileName` sets `<pathname\basename>` and `DigitsCount` sets the number of digits in `<ImageCounter>`.

For example, setting `FileName` to `"C:\Users\Angelo\Pictures\IMG_"` and `DigitsCount` to `4` will save images to:

- `C:\Users\Angelo\Pictures\IMG_0001.JPG`
- `C:\Users\Angelo\Pictures\IMG_0002.JPG`
- ...

When this function is called, it automatically searches the directory for images and will set the starting value of `ImageCounter` to the last image found +1.

3.43 ABCanonSetSaveOverrideA / ABCanonSetSaveOverrideW

Sets the path/filename to use **once** when download mode has been set to `ABC_IMGDL_SAVE_TO_FILE`.

Syntax:

```
int ABCAPI ABCanonSetSaveOverrideA(
  ABCAMERA_HANDLE hCH,
  char *FileName
);
```

or

```
int ABCAPI ABCanonSetSaveOverrideW(
  ABCAMERA_HANDLE hCH,
  LPWSTR FileName
);
```

Parameters:

`hCH`
: The camera handle

`FileName`
: The filename to save to, either ANSI for `ABCanonSetSaveOverrideA` or WIDECHAR for `ABCanonSetSaveOverrideW`.
Use NULL to clear a previously set override filename.

Return value:

The error code.

Remarks:

The filename set with this function will be used without any modification/expansion (except for adding the correct extension) just once for the next image download and then discarded.

3.44 ABCanonSetSaveTarget

Sets where the camera saves the image.

Syntax:

```
int ABCAPI ABCanonSetSaveTarget(
  ABCAMERA_HANDLE hCH,
  int SaveTarget
);
```

Parameters:

`hCH`
: The camera handle

`SaveTarget`
: The destination of the image:
- `ABC_SAVETGT_PC`: The camera downloads the image to the PC via the USB connection.
- `ABC_SAVETGT_CF`: The camera saves the image to the compact flash
- `ABC_SAVETGT_BOTH`: The camera both saves the image to the compact flash and downloads it to the PC.

Return value:

The error code.

Remarks:

If you want to download the images to the PC, you should call `ABCanonSetSaveTarget(...,ABC_SAVETGT_PC)` after having connected the camera.
When the camera is disconnected, the library will automatically set the destination to `ABC_SAVETGT_CF`.

3.45 ABCanonShoot

Takes an image.

Syntax:

```
int ABCAPI ABCanonShoot(
  ABCAMERA_HANDLE hCH
);
```

Parameters:

`hCH`
The camera handle

Return value:

The error code, which is typically ABC_RETVAL_DENIED when shooting is not possible. Call `ABCanonIsShootingAllowed()` to find the reason.

4 Error reference

4.1 ABC_RETVAL_OK

No error.

4.2 ABC_RETVAL_ERROR

Generic error. Normally this error means that some USB command returned some unexpected (and therefore unhandled) error.

4.3 ABC_RETVAL_BUSY

The camera returned a BUSY response, meaning that the camera menu is displayed on the screen. No further operations can be done until the menu is exited (by the user). See page 18.

4.4 ABC_RETVAL_BUFFER_TOO_SMALL

The buffer passed to some function (like `ABCanonGetAllSettings`() or `ABCanonGetWritableSettings`()) is too small.

4.5 ABC_RETVAL_INVALID_HANDLE

The camera handle passed to the function is not valid (it's `NULL`).

4.6 ABC_RETVAL_INVALID_PARM

A parameter passed to the function is not valid.

4.7 ABC_RETVAL_OUT_OF_MEMORY

A memory allocation within the library failed.

4.8 ABC_RETVAL_NOT_FOUND

The searched camera was not found.

4.9 ABC_RETVAL_SEM_ERROR

An error occurred while accessing the event queue synchronization semaphore.

4.10 ABC_RETVAL_INVALID_PATH

The path passed to `ABCanonSetSaveFilenameA / ABCanonSetSaveFilenameW()` is not valid.

4.11 ABC_RETVAL_TIMEOUT

No events occurred within the timeout period passed to `ABCanonGetEvent()`

4.12 ABC_RETVAL_DENIED

Shooting is not possible.

4.13 ABC_RETVAL_NOT_CONNECTED

The camera is no longer connected. Call `ABCanonDisconnectFromCamera()` to close the camera handle and to release all the internal resources.

4.14 ABC_RETVAL_THREAD_BUSY

This return value is possible only with the option `ABC_OPTIONS_HANDLE_THREAD_BUSY` set. It means that the function call has been aborted due to a possible deadlock. See Threading on page 20 for more information.

5 Library revision history

5.1 v1.00

Initial release

5.2 v1.01

- Handled notification for changes in the custom options
- Added `ABCanonGetCustomOption()` function

5.3 v1.02

- Added `ABC_OPTIONS_HANDLE_THREAD_BUSY` option
- Fixed a crash if neither `ABCanonSetSaveFilenameW()` nor `ABCanonSetSaveFilenameA()` were ever called
- Added `ABC_EVENT_JPG_DISCARDED` and `ABC_EVENT_CR2_DISCARDED` events
- Changed return values for the ABC_EVENT_JPG_READY and ABC_EVENT_CR2_READY events. Now these events are more consistent with the return values of all the other events. ****WARNING** this change is **NOT** backward compatible!!!**

5.4 v2.00

- Added Canon 5D support
- Added `ABCanonGetCustomFunction()` to replace the now deprecated function `ABCanonGetCustomOption()`.
- Added `ABCanonGetCameraType()` function.
- Added `ABC_OPTIONS_STANDARD_BULB` option for the Canon 5D

5.5 v2.01

- WINUSB.DLL is no longer implicitly linked but it is explicitly loaded during ABCanon7 initialization. This allows a third party program to implicitly link ABCanon7.DLL and start execution even if no WINUSB.DLL is installed in the system.

5.6 v2.02

- Fixed an issue that could cause excessive CPU usage.
- Added `ABCanonGetPQImageCount()` function.

- Added `ABC_IMGDL_OVERRIDE_OR_DISCARD` download mode and `ABC_OPTIONS_ALLOW_OVERWRITE` and `ABC_OPTIONS_PERMANENT_OVERRIDE` options.
- Fixed a problem that prevented the library to work with Windows 10

5.7 v2.03

- Some notifications are done by a new helper thread to avoid deadlocks.
- Added an option to make the library poll the camera when a BUSY condition occurs, instead of having to do this poll from the user code
- Added ABC_EVENT_BUSY event to notify the begin/end of camera busy condition
- Added ABC_EVENT_ALL_DONE event to notify the completion of all shoot related operations.