CanonHDRCapture - Developers Documentation

Franz Daubner

May 2007

1 Changing the code

1.1 Getting the project to compile

To run the application the .net 2.0 runtime environment must be installed. If you want to make changes to the code and rebuild the application you need Microsoft Visual Studio 2005.

1. Start Visual Studio and open the solution file by clicking *File* → *Open* → *Project\Solution* and select the solution file "`CanonHDRCapture.sln`" in the project root directory.
2. Check in the Solution Explorer window if the CanonHDRCaptureGui Project is the startup project. If the project name is written in bold it is already the startup project, if not right-click the project name and select *Set as StartUp Project*. (If the solution explorer isn't visible at startup select *View* → *Solution Explorer* from the menu)
3. Select the desired build target, either *Release* or *Debug* from the tool bar. If the tool bar doesn't show the build targets you can choose a target by selecting *Build* → *Configuration Manager* from the menu.
4. Build the solution by selecting *Build* → *Build Solution* from the menu.

The output directory is either in `<ProjectRoot>\CanonHDRCaptureGui\bin\Release` or `<ProjectRoot>\CanonHDRCaptureGui\bin\Debug` (depending on the selected build target) and contains all neccesary files to run the application.

1.2 Changing the default settings

There are two types of settings, user-specific settings and application-wide settings. The user specific settings are stored in a file called "`user.config`" in the path
`C:\DocumentsandSettings\<UserName>\LocalSettings\ApplicationData\CanonHDRCaptureGui\<RandomPath>`
where `<UserName>` specifies the username of the currently logged in user and `<RandomPath>` represents a randomly generated path like
`CanonHDRCaptureGui.exe_Url_vyn0sxjec4fi3nazb3zx1ke0t21emrus\1.0.2454.21710`
(don't ask, its a Microsoft thing...)
The application wide config file is stored in the application root directory and is called "`CanonHDRCaptureGui.exe.config`".
Both config files are generated as human-readable xml and can be edited with a text editor. The application config can also be edited with the Visual Studio Designer (doubleklick the "`Settings.settings`" entry in the Solution Explorer under `CanonHDRCaptureGui/Properties`)
The user-specific config can be safely deleted and will be generated anew each time the application is started. The application config should not be deleted (although the application can run without it)
The following values are currently stored in the config files:
User specific config:

- FilePath: Specifies the currently selected path for storing captured images
- MinLuminance: Minimum pixelvalue to determine if an image is underexposed

- MaxLuminance: Maximum pixelvalue to determine if an image is overexposed
- Fisheye: Enable/disable checking of the image corners
- Settings for the special treatment of fisheye lenses

Application wide config:

- Default values for all user config properties
- Default values when creating a new Timer

2 Software components

2.1 Project Descriptions

2.1.1 ApiWrapper

Language: managed C++
The main purpose of the ApiWrapper is to make the functions of the Canon api available for a C# program. It also facilitates the use of the api by providing an interface specifically tailored to the needs of the CanonHDRCapture application.

2.1.2 CanonHDRCaptureGui

Language: C#
This Project contains the user interface and the main application logic.

2.1.3 SharedStructs

Language: C#
Contains the classes, data structures and interfaces used by both ApiWrapper and CanonHDR-CaptureGui

2.2 Interaction with the Canon Api

(see Figure 1)
All interaction with the camera from the point of view of the Gui is done via the CameraControl class, located in the CanonHDRGui Project. The CameraControl class provides properties for reading and setting parameters on the camera and publishes events, indicating if values get changed or the status of the camera changes (like the camera getting disconnected). The CameraControl class in turn communicates with the CamInterface class which is located in the ApiWrapper project. To be able to receive events from the CamInterface, the CameraControl registers itself with the IUpdateClient interface located in the SharedStructs project. Since a pointer to a managed function cannot be passed directly to an unmanaged function, the NativeHelper is used to enable the CamInterface to get notified of events from the Canon Api (see the source code documentation for further details).

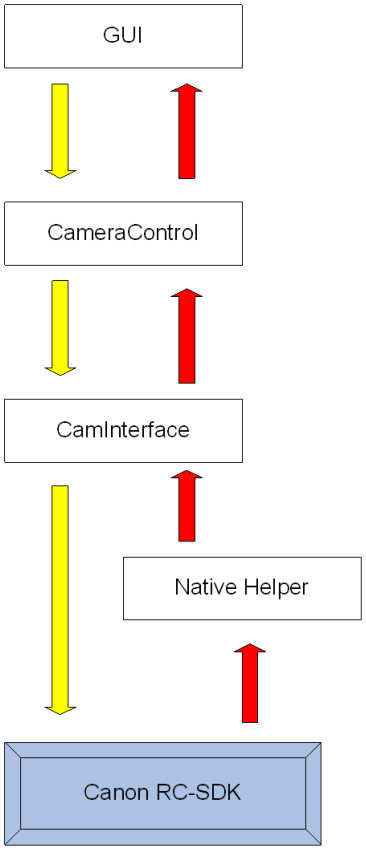

Figure 1: data flow

2.3 Timer - Execution

(see Figure 2)
The capturing of a HDR-Sequence is done by a HDRSequenceController which takes a HDRSequenceSettings class as input. The HDRSequenceController is implemented as a state machine. It tells the CameraControl to capture an image and then waits until it is triggered again by an event, typically by the ImagePipe indicating that a new image has arrived. Then it decides whether another image needs to be captured, adjusts the exposure parameters on the camera accordingly and fires another shot. As soon as all necessary images have been captured, it fires an event that the sequence has been captured successfully. The execution of a timer is done by the HDRIntervalController class, which needs a HDRIntervalSettings object as input and uses a HDRSequenceController for capturing single sequences. The architecture of the HDRIntervallController is similar to the HDRSequenceController, with the exception that it is triggered by timer events, not by camera events.

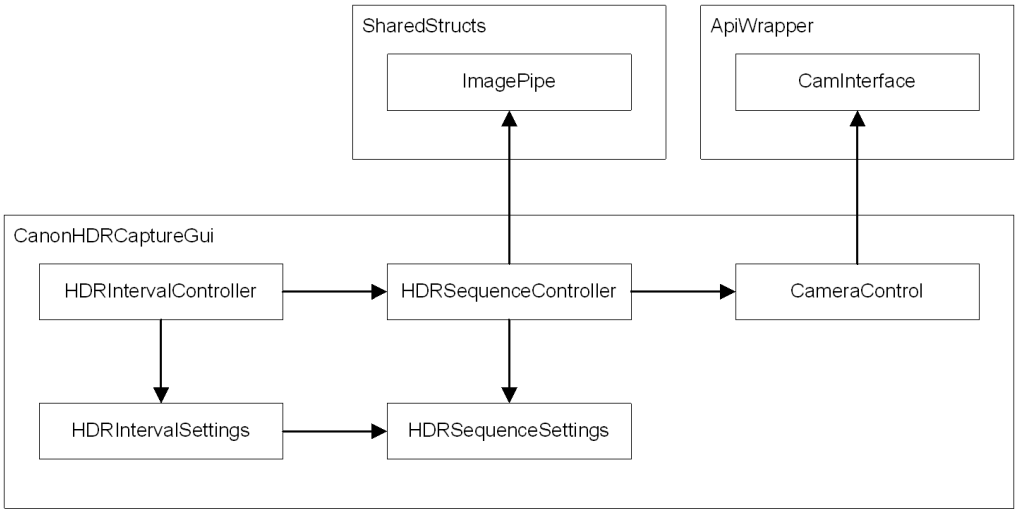

Figure 2: execution

2.4 A word about .net Events

The delegates registered with an event are always executed in the same threadcontext as the task that fired the event. That means a callback from the Canon Api triggering an event in the main window would execute code directly located in the main window. Since this is not allowed in .net 2.0 (and would hang the Api anyway) the execution of the delegates needs to be shifted to another thread. To accomplish this, some classes require a Windows.Forms.Control as a constructor - parameter, and use the BeginInvoke method of that Control to shift execution of the delegates to the Gui thread.

3 Experiences with the Canon-Toolkit

3.1 Compatibility

For this project I used the Canon RC-SDK version 8.4 which theoretically supports all Canon SLR's up to Canon 5D. Since Canon significantly changed the way pictures are retrieved from the camera starting with the Canon 20D, different function calls and callbacks are needed to support older cameras. Since I didn't have a suitable camera for testing purposes at my disposal, currently no camera older than the 20D is supported. Starting with the 30D Canon changed the communication protocol again, so newer cameras are not supported either.
Supported models include:

- EOS-1Ds Mark II
- EOS-1Ds
- EOS-1D Mark II N
- EOS-1D Mark II
- EOS 5D
- EOS 20Da
- EOS 20D
- EOS 350D (EOS Digital Rebel XT)

3.2 Changing Shooting Parameters on the camera

According to the documentation, if you want to change a shooting parameter (like aperture) on the camera, you would call a function like RCSetAvValue with the appropriate Parameters and the function would return an error code whether the operation succeeded. What the manual fails to mention is that these functions actually always return an error, the only exception being that the parameter on the camera is already set to the desired value. Sometimes the value on the camera gets changed after calling a set-function (even though the function returned an error) and sometimes not. If you try to set camera parameters in rapid succession (consecutive calls in the code actually) it never works on the first try (well, in some rare cases it does, but most of the time it doesn't). My solution to this problem was trying to set a parameter until the value is set on the camera or a threshold of "tries" is reached (just in case it's really impossible to change that value).
the "algorithm" looks like this:

```
for (int i = 0; i < 100; i++)
{
    SetParameterXY(NewParameter);
    if (GetParameterXY() == NewParameter)
        break;
}
```

It usually takes 2-5 tries until the value is set on the camera so I found a threshold of 100 tries to be sufficient.

3.3 Killing the Canon-API

There are two surefire ways to render the Canon API completely useless:

1. Using the debugger to stop the thread in which the API is running while the API is communicating with the camera
2. Taking too long to process an event fired by the API (or trying to fire the next shot in the same threadcontext as the API callback)

In both cases the only chance to get the API to work again is to reboot the computer and to turn the camera off and on.