DLLExampleVS2010

This is a simple MFC dialog project which demonstrates all the basic functionality required to use the camera to perform a high-speed capture and then download the captured frames.

The application UI is extremely basic since the main purpose is for the code to show how to use the DLL with the camera. Error checking and handling are limited to help improve clarity.
It was designed for Visual Studio 2010, but should be compatible with future versions, and should work with older versions (up to a point) as well if the project settings are re-created.

How to use the program:

The program requires MS_CameraControl.dll to run. MS_CameraControl.dll may require you to install the Visual Studio 2010 Redistributable (See redist.txt for details). It is also necessary that you have installed the drivers that came with the camera control software.

The camera needs to be powered on and connected, but this can be done either before or after starting the application. After starting the software and powering on the camera you should now be able to find and connect to the camera using the “Connect” button.

NOTE: The process of finding and connecting to the camera can take some time, during which the example program will be unresponsive. Connecting to the camera or other slow tasks would not typically be done on the UI thread to avoid this problem, but for simplicity the example doesn’t do this.

Once connected to the camera, it is now possible to start a capture. The image size, capture speed, exposure time, etc. are hardcoded in the example; see the code if you want to change them.

The Capture button starts a continuous capture, which will keep capturing frames until the button is pressed again to stop the capture. When the Camera memory is full, new frames will overwrite old frames, in a loop.

The Pre/Post trigger button will start capturing similar to above, but the camera will be waiting for a trigger pulse input (from hardware). When the camera receives this pulse, it will continue capturing a specified number of frames (see the code) and then automatically stop.
This means you can maximize the use of the available memory on the camera without worrying about unintentionally overwriting frames before and including the trigger event.
As an example: if the camera can store 1000 frames and is configured for 500 post-trigger frames, at least 500 pre-trigger frames will not be overwritten.
If triggered after capturing only 10 frames, there will obviously only be that many pre-trigger frames available.
It is also possible to stop the camera through software before all the post trigger frames have been captured if desired.

NOTE: The example software does not poll the camera state to determine if/when the camera has stopped recording, so although the hardware maybe have stopped it is still necessary to press the stop button before proceeding to download.

The camera sends back preview frames while capturing, but since it is generating data much faster than gigabit Ethernet can transmit it, it is only sending back a small subset of all the frames.

To get all the frames, the capture must be stopped, so the camera can be put into download mode. Depending on whether a continuous or pre/post capture was done previously, the software will either download frames to PC RAM (they will be displayed but not stored, unless you store them) or saved to a RAW AVI file by the DLL. This is not a limitation of the camera or DLL; it was done this way simply to show two possible alternatives for downloading.

The download start and end frame are specified by software, in the case of pre/post trigger mode you can retrieve the frame number where the trigger occurred (if it was triggered) to help determine what range of frames should be downloaded. Additionally the download speed can be slowed down if there are problems with dropped frames, etc.

NOTE: The path in the AVI file case is the current directory (typically the application location, unless it has been changed), so if the software doesn't have write permissions to that location it may fail to save.

About the code:
DisplayWnd.cpp – A window to display images. Nothing specific to the camera in here and won't be discussed further.
DLLExampleVS2010Dlg.cpp – Contains all the relevant code, along with some boilerplate MFC code.

The beginning of the file contains some global variables and function definitions, along with the MFC boilerplate.
Next are the button click handler functions, which contain some application and UI logic, but more importantly the calls to the functions that make everything go. From this point it's probably easiest to just look at the code to follow the execution path for each main step: Connect, Capture and Download.