Adaptive Vision

Intuitive

Powerful

Adaptable

Adaptive Vision Sp. z o. o.
ul. Bojkowska 35a, 44-100 Gliwice, Poland
Tel. +48 32 461 23 30
E-mail: info@adaptive-vision.com

Application Note

Interacting with GigEVision cameras

Created: 11 June 2019

Modified: 2 December 2019

Document version: 1.1.6

Contents

1. Introduction

The following guide provides information on how to utilize a GigE-compliant camera in Adaptive Vision Studio.

While the guide focuses on the GigE protocol most of the content can also apply with slight changes to different camera protocols, especially GenICam.

2. Basic Image Acquisition

The simplest scenario in which cameras are utilized in Adaptive Vision is continuous synchronous acquisition.

Continuous means that the camera acquires new images by itself – it does not need to be triggered to do so.

Synchronous means that the program will wait for a new image until it is acquired – the iteration will not finish without a new image.

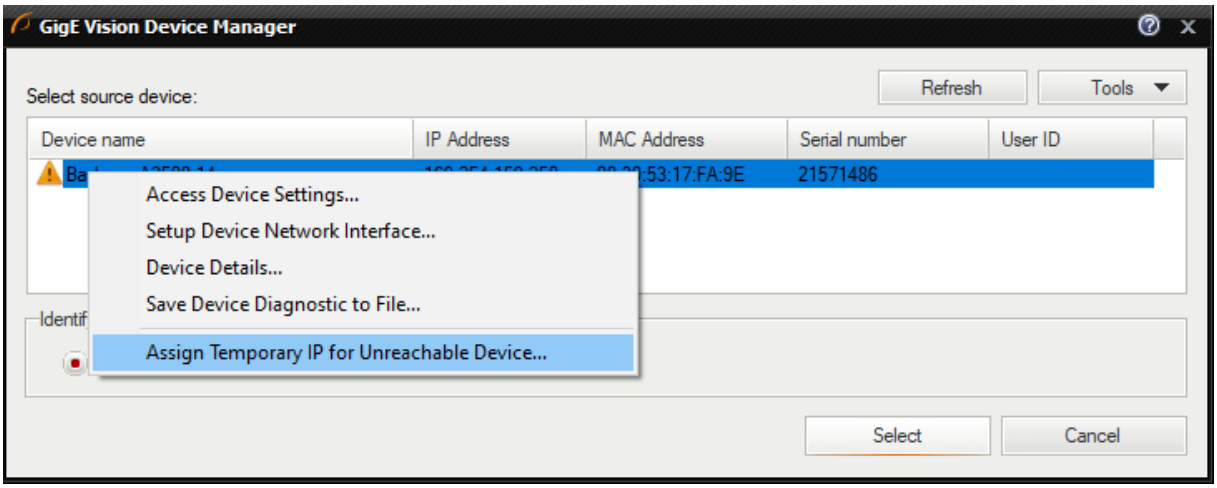
The steps to perform such acquisition are as follows:

1. Connect the camera to the computer or network switch. If the camera is to be powered through Power-Over-Ethernet (PoE) make sure that it is connect to a device capable of supply power that way.

2. In a new program in Adaptive Vision Studio add the filter GigEVision_GrabImage:Synchronous.

3. Open the editor of the inAdress input and set the camera address.

    a. If the camera is displayed but with a warning sign it means it is not in the computer's subnet. This can be solved by assigning a temporary IP address the camera



    b. When the camera is in the same subnet it is possible to change its settings (for example changing its static IP).

    c. If the camera is connected but not detected you may open the window from point a), manually type the camera MAC address and assign it another IP address. The menu can be opened from the Tools menu.

4. Now you can run the program. Previewing the outImage will show you the view from the camera.

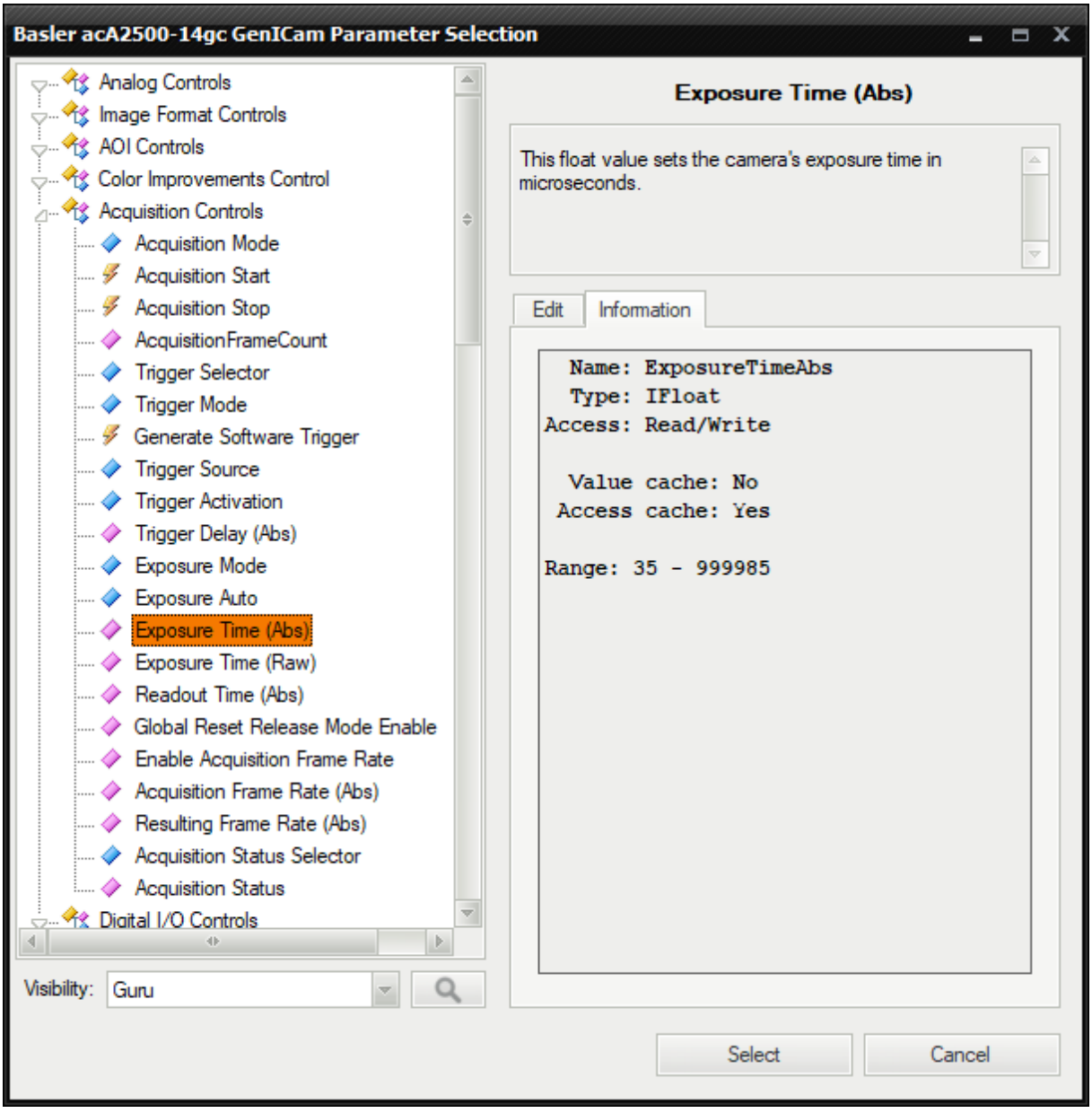
3. Changing camera parameters

You can change camera parameters programmatically using SetParameters filters. To demonstrate this, we will expand the previous program.

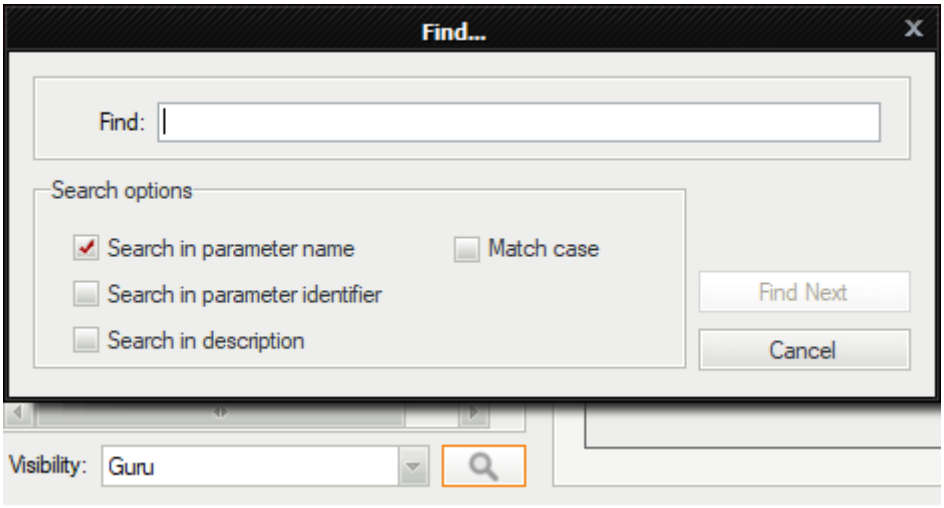
1. Add a GigeVision_SetParameter:Real filter and set its inAddress to the same address as GigEVision_GrabImage:Synchronous.

2. Specify the parameter name. if you are not sure about the name you can select through the GigEVision device tree. To do that click on the "…" button near the inParameterName.

3. Here you can see all available camera parameters with a short description. Select the parameter with name Exposure Time (Abs) (or similar if not present). As you can see it control the camera exposure time in microseconds and its type is IFloat.



a. If you cannot find the parameter try using the search function (the magnifying glass icon).



b. The type of the filter needs to match the type of the parameter. The exception is that IFloat is represented as Real in Adaptive Vision.

c. Some more advanced parameters may not be visible unless Visibility is set to the correct level.

4. Make sure that the parameter ExposureAuto is set to Off. Otherwise, it will not be possible to manually change the exposure value.

5. Before closing the window note the minimum and maximum values of exposure time. After selecting the parameter set the inValue to the lowest acceptable value.

6. Run the program. While running, steadily increase the inValue input. You will notice that the camera image gets brighter and brighter.

    a. Try not to go outside the acceptable range. It will result in an error if the input inVerify is set to True.

7. It is also possible to check a parameter's value by using GigeVision_GetParameter filters. Try adding one in the Float variant and specifying the same name as in the previous filter.

    a. You may notice that not always the read value is equal to inValue of SetParameter. It is because the camera modifies it.

With GigeVision_GetParameter filters it is not only possible to check editable parameters but also to check read-only ones. For example, if the camera was set to automatically adjust exposure, the current exposure time might be useful to know. Another example would be to read parameters holding device specific information, like maximum image size.

For some quick testing you may also want to set parameters directly in the GigEVision tree.

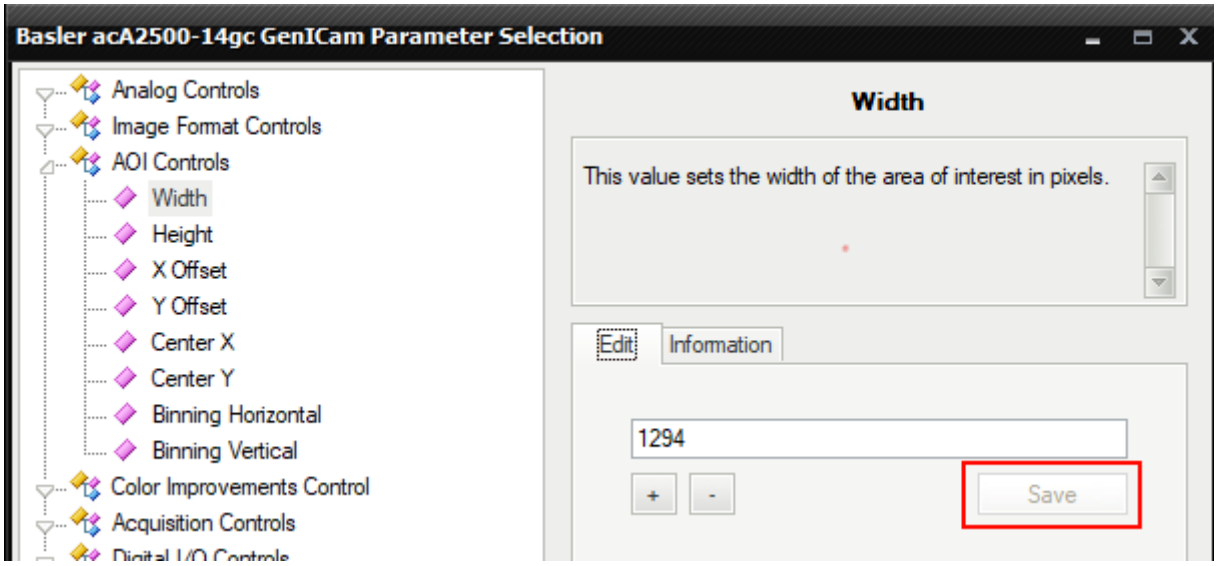
It is important to note that while most parameters are stored in volatile memory and as such will reset after unplugging the camera from power. There are some parameters that are stored in non-volatile memory, such as Device User ID, but they usually do not affect the acquisition directly.

Because of this it is recommended to design program in such a way that every parameter with value different than default is set programmatically.

4. Starting and stopping acquisition

It is important to note that not all parameters can be changed while the acquisition is running. Some require stopping the acquisition and restarting it.

1. Run the program and open the device tree (Tools -> Manage GigE Vision Devices…). Find the Width parameter. You will notice that the save button is grayed out due to running acquisition.



2. Add another SetParameter filter (Integer) and select the Width parameter. You may notice that it is greyed out.

    a. While selecting the parameter check the information tab. There may additional information about possible value, e.g. increment. Some parameters can only take value that are multiples of certain numbers, like 2 or 4.

3. Enter 240 as inValue and run the program. You will encounter an error saying that the parameter is not writeable.

4. Now add a GigeVision_StopAcquisition before the previous Set filter and rerun the program.

5. The parameter is now set without errors and the output image is smaller.

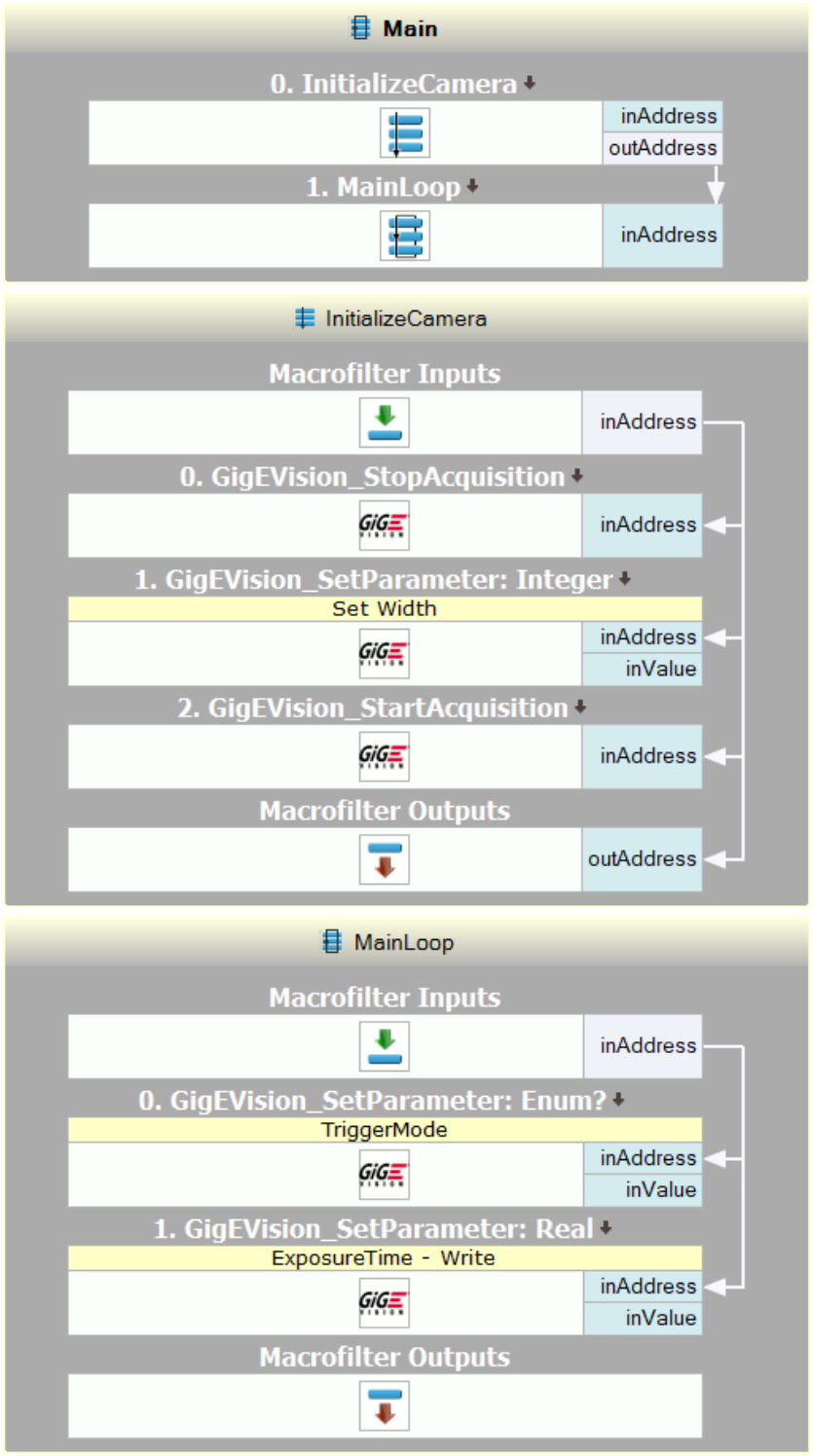
As you can see the Width parameter was not writable while the acquisition was running. Stopping the acquisition before changing the parameter made it writable.

You may notice that we have a filter to explicitly stop the acquisition but there is not filter to explicitly start it. GrabImage will attempt use an ongoing acquisition but if none is present it will start a new one.

While it is acceptable for simple programs, it should be avoided when the status of the acquisition is program-controlled.

Now we will make a proper program out of this.

1. Start with removing the filters added in this chapter.
2. Now select the rest and extract a task macrofilter (called MainLoop) from them. This will be the main acquisition loop.
3. Create a new step macrofilter called InitilializeCamera and add it before MainLoop.
4. In InitilializeCamera add the three following filters GigeVision_StopAcquisition, GigeVision_SetParameter:Integer (select Width parameter), and GigeVision_StartAcquisition.
    a. Set the width to the maximum possible value.
5. Drag inAddress of any of the filter to the top bar to create a new input. Then add an output of the same type and connect it to the input
6. Go up one level and drag the outAdress output do the MainLoop filter creating a new input.
7. Inside that filter connect the newly created input to all the GigE filters.
    a. Now every GigE filter in the whole program shares the same camera address.
    b. The final program should look like this:



Now the program is divided into two parts. The first part is executed once, after that the program enters MainLoop which run continuously until the program is stopped.

The parameters that can be changed during acquisition are set in the InitializeCamera filter where the acquisition is guaranteed to be stopped.

All the other parameters are set in MainLoop.

In real applications not all parameters need to be changed during runtime. Even then they probably will only be changed from time to time, not in every iteration.

For example, specifying if the camera will run continuously or in trigger mode will likely be done only once (even though it can be set during acquisition). Exposure time might be changed multiple times.

It is a good practice to have all the parameters that are set only once in one macrofilter (like InitilializeCamera) regardless of if they required stopped acquisition.

The parameters changed from time to time should be set in a variant macrofilter.

The reason for that is time-saving. Setting the parameter to the same value does not take much time (Adaptive Vision caches previous values and avoids resending the same ones), but if we have a lot of parameters it adds up. It is also more intuitive if one-off parameters are in a dedicated filter.

It is important to note how the cameras work in Adaptive Vision. When the acquisition is started Adaptive Vision creates a background thread which buffers incoming frames in memory. This thread persists until the acquisition is stopped or the application exits the task that started the acquisition.

For example, if InitializeCamera was a task and not a step macrofilter the acquisition would stop when exiting it and it would have to be restarted in MainLoop.

Also, instead of passing the camera address as a parameter it is possible to put it into a global parameter.

5. Changing trigger mode – basic

While the default mode of most cameras is continuous acquisition very often it is more desirable to only acquire new images after a certain signal, i.e. trigger.

Not only you have more control over the acquisition time of each frame, less frequent acquisition requires smaller bandwidth.

To demonstrate triggered acquisition, we'll use the current program.

Run the program, ensure that you displayed the camera image in the preview.

1. Now open the GigeVision device tree and find the parameter Trigger Mode. Set it to On. Also make sure that Trigger Selector is set to Frame Start.
2. You will notice that no new images are grabbed by the camera.
3. Find the Trigger Source parameter and set it to Software.
4. Find the Generate Software Trigger command (commands feature the lightning bolt icon) and execute it.
5. A new image will be grabbed by the camera.

You may also want to have an external trigger (for example, an output from a PLC). Then you must select the appropriate Trigger Source, e.g. Line1.

You may also have to ensure that the signal matches what the camera expects. The Trigger Activation parameter specifies what value will the camera look for.

6. Changing trigger mode – advanced

Very often we want the trigger properties to be controlled directly by the program. For example, the camera may be triggered in one part of the program but also acquire images continuously in another part. Or maybe the camera is to be triggered by different sources depending on the program state.

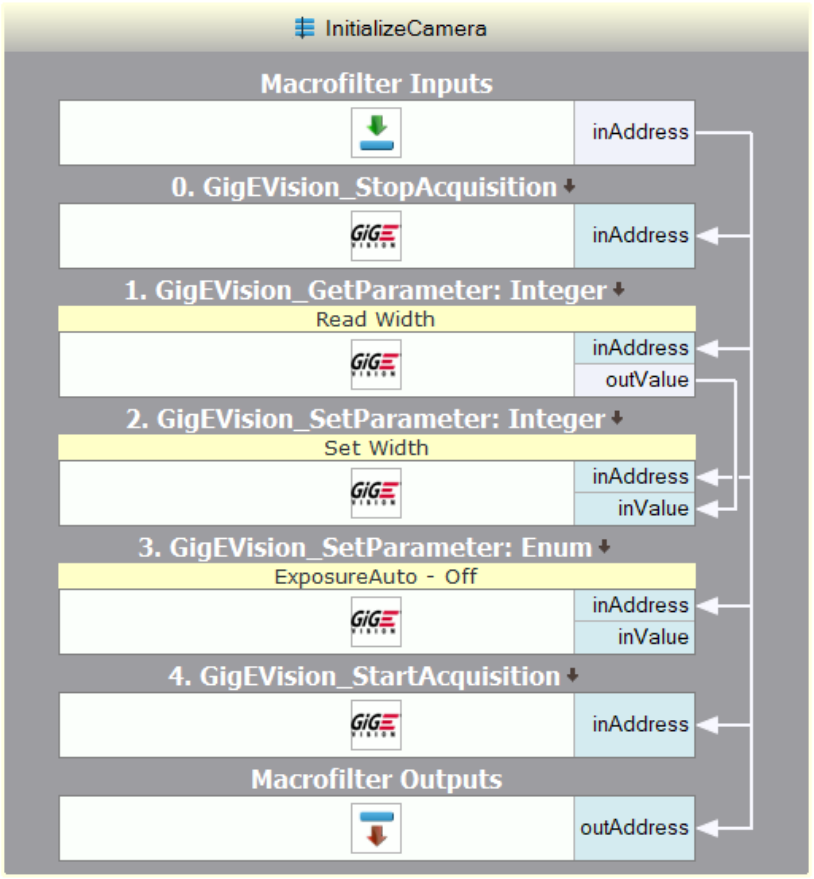
Trigger parameters can be changed during acquisition. Therefore, it is possible to change them on-the-fly.

Now we will prepare a more advanced program which that initializes camera and allows the user to change how the camera is triggered and the exposure time from the HMI while the program is running.

1. To ensure that the camera will be set to use Software as the source of trigger, add a SetParameter:Enum filter in the InitiliazeCamera (before StopAcquisition). Through the Device Manager choose the TriggerSource parameter and set inValue to Software.

2. Now repeat the previous step, changing the parameter to ExposureAuto and set its value to Off.

3. Since different cameras may have different acceptable maximum width you may want to add a GigeVision_GetParameter:Integer filter that would read the parameter WidthMax in device information). Connect its output to the filter that changes the Width.



View of modified InitializeCamera. Comments added for clarity

4. Now we will modify the contents of MainLoop macrofilter. We want to be able to change trigger mode and to execute a software trigger.

    a. Add an instance of GigEVision_SetParameter:Enum. Connect it to the camera address and select TriggerMode as its parameter.

    b. Add a GigEVision_ExecuteCommand filter, connect them to the camera address. Through Device Manager select parameter TriggerSoftware;

    c. Move all those filters before GrabImage.

5. Let's design an HMI. In this case it will feature:

    a. View2DBox – which will display the camera image;

    b. ComboBox – for selecting trigger mode;

    c. NumericUpDown – to control exposure time;

    d. ImpulseButton – to generate software trigger;

    e. Labels – to label other controls.

    Add them and distribute on the HMI canvas.

6. Now we will configure the controls:

    a. View2DBox – change its InitialSizeMode to FitToWindow and connect to GrabImage's outImage;

    b. ComboBox (for trigger mode)

        i. Connect outText to inValue of SetParameter for TriggerMode;

        ii. Expand List in Data category in Properties and add the following item: On, Off

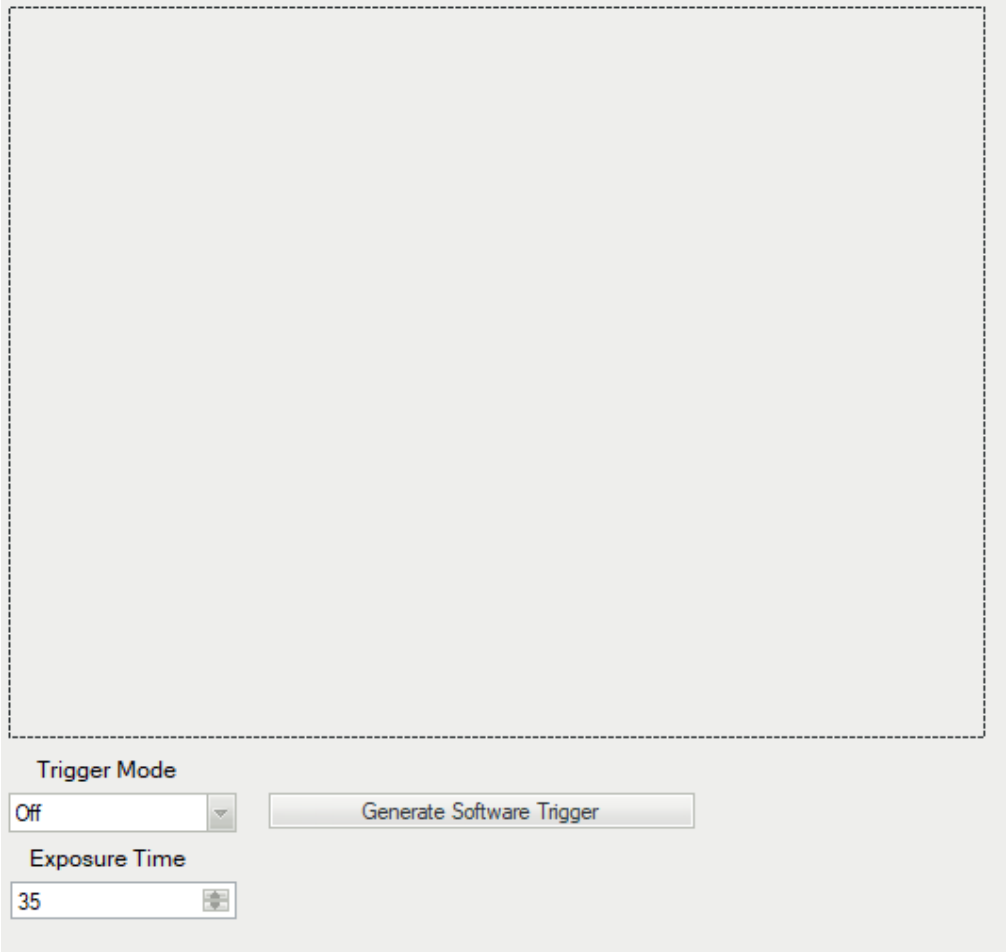
        iii. Set Selection to 0

    c. ImpulseButton – connect its outValue to ExecuteCommand's inValue;

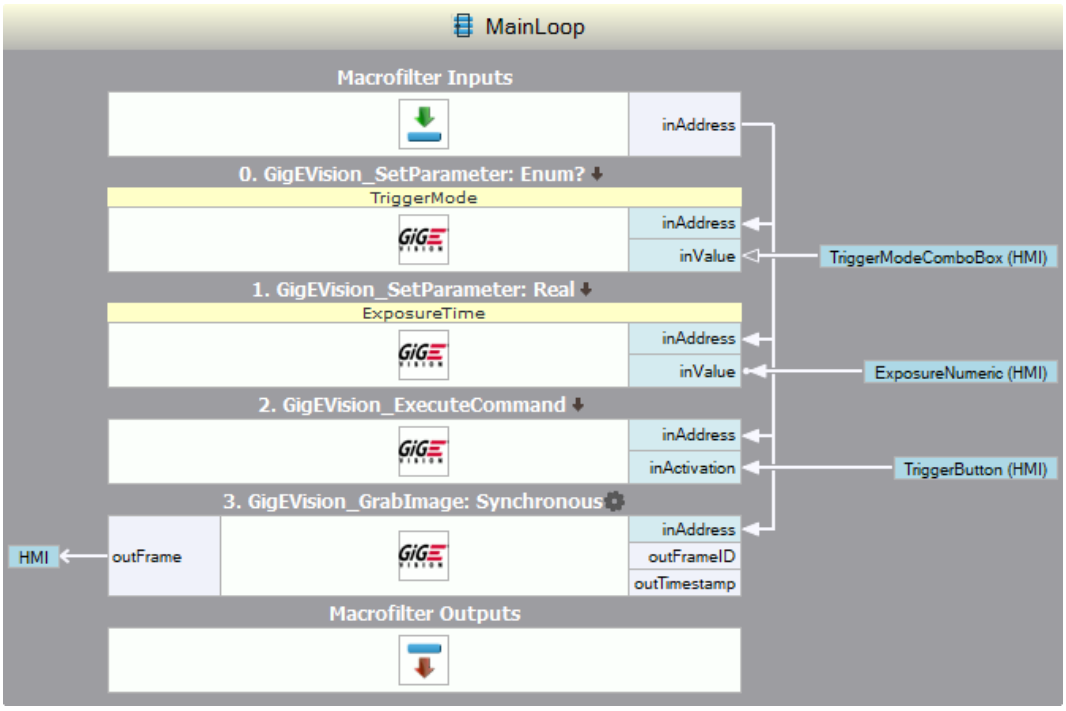
    d. NumericUpDown:

        i. Connect outValue to inValue of SetParameter for ExposureTime;

        ii. Set Minimum and Maximum to the values specified in the Device Tree for that parameter (for the camera used while writing this note the values were 35 and 999985).



View of HMI



View of modified MainLoop. Data ports labelled for clarity

7. The program should display the camera image on the HMI. You should be able to change the exposure time. However, when you change the ComboBox's value to On, the program will freeze. Clicking the trigger button will not do anything.

    a. This is caused by the fact that the GrabImage in its Synchronous variant. The cannot get past GrabImage and execute a trigger, so it waits for an image indefinitely.

    b. To solve that we can change GrabImage's variant to WithTimeout. Now the program will only attempt grabbing for a specified amount of time. Let's set inTimeout to 100ms and rerun the program.

8. Now the program does not freeze when setting triggered mode on. If no image is grabbed in 100ms GrabImage returns Nil and the program proceeds to the next iteration, where the camera may be triggered.

    a. Generally if the camera is meant to acquire images only from time to time it is a good idea to use WithTimeout variant of GrabImage. It prevents program hang-ups in case of some camera problems and allows to inform the user that the camera may not be working correctly. However, the rest of the program must be designed in a way that handles Nil images properly.

We can now expand the program to enable the user to change other parameters, including those which require acquisition to be off.

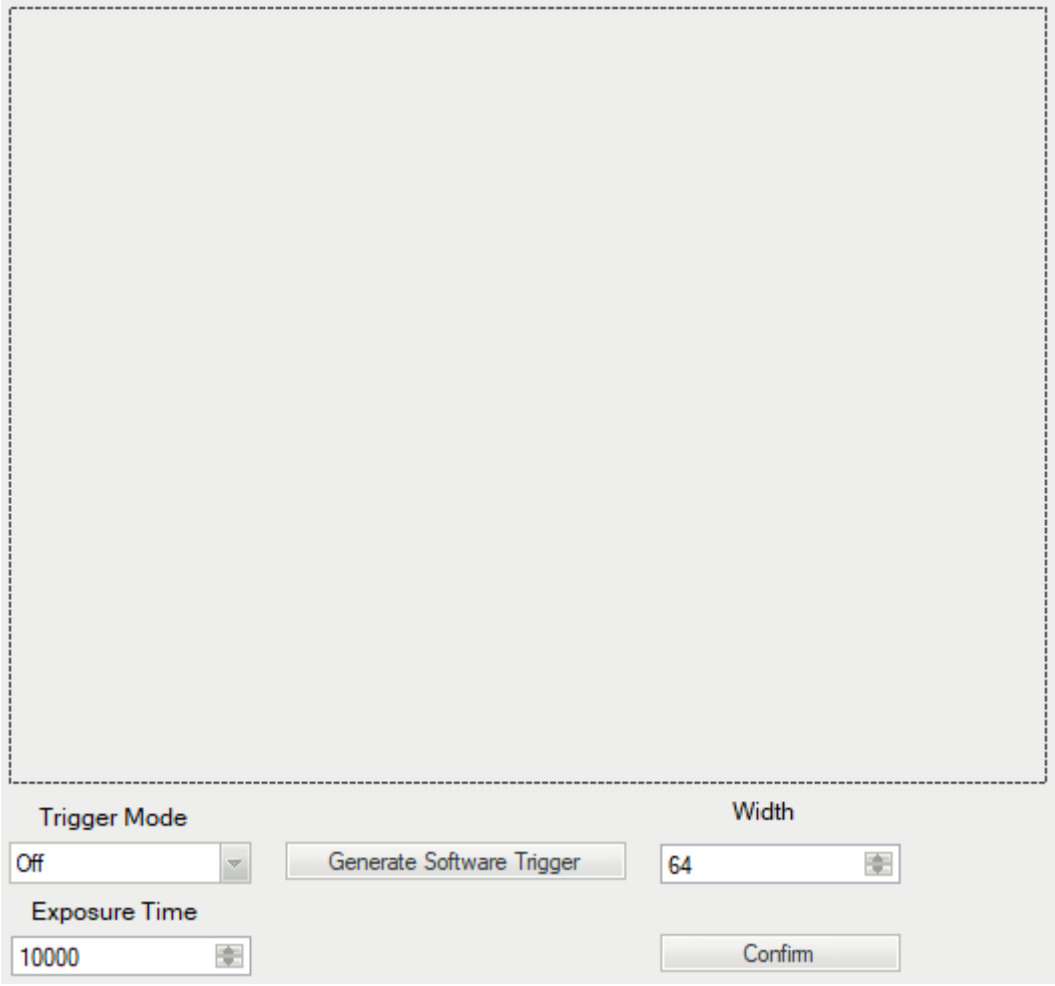
1. Create an empty variant macrofilter inside MainLoop and choose Bool as the fork type. Drag the camera address to the filter to create an input

2. Enter the filter and choose the variant True. Add a StopAcquisition filter, followed by a SetParameter filter in variant Integer and select Width parameter. After them add a StartAcquisition filter. Connect all filters to the camera address.

3. In the HMI add two new controls:

    a. NumericUpDown – to control width's value; set its Minimum and Maximum to the respective limits of the Width parameter in the camera (for the camera used while writing this note the values were 35 and 999985); set Increment to the respective value as well.

    b. ImpulseButton – to confirm new value.



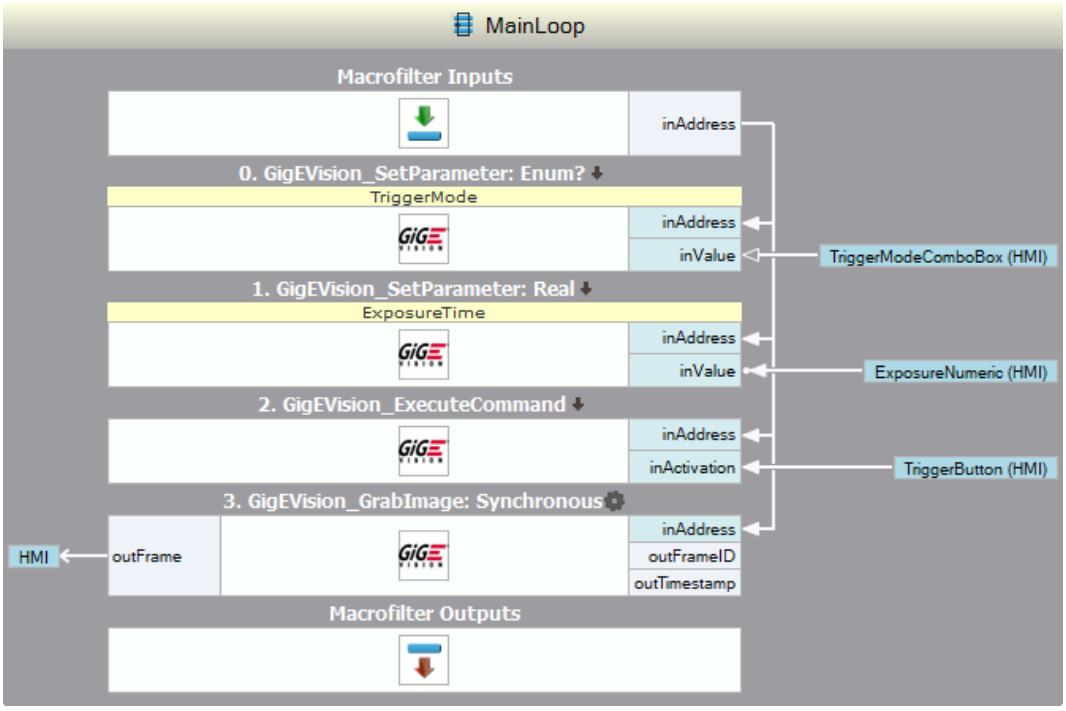
Final view of the HMI

4. Connect NumericUpDown's outValue to the new SetParameter's inValue.

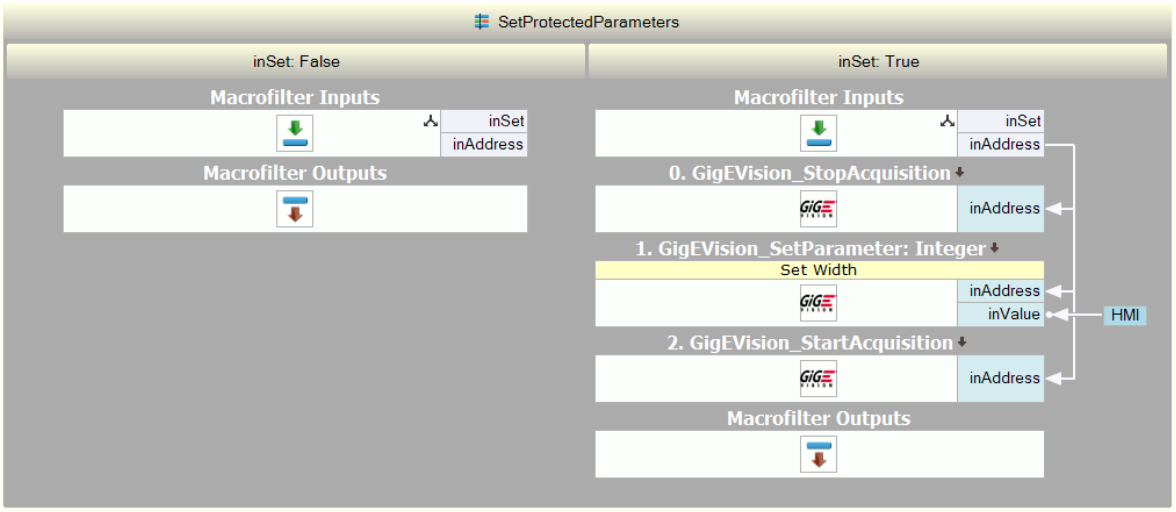
5. Go back to MainLoop and connect the new ImpulseButton's outValue to the condition port of the variant macrofilter.

Now when the user clicks the confirm button the program will enter the variant macro, stop the acquisition, change the width and restart the acquisition. If the user changes the value by using NumericUpDown buttons the value will be properly aligned.

However, if the user enters a value by typing it may be it will not be guaranteed to work with the camera. It is possible to design a macrofilter which will make sure the value accepted by the camera, but it is not the topic of this application note.



Final view of MainLoop



View of the variant macrofilter

7. Additional remarks

Camera filters cannot be executed in array mode (for example when an array of camera addresses is connected).

Since a camera is an external hardware, they may be causing errors (like connection errors) unrelated to the Adaptive Vision application. To properly handle those errors the application should feature error handling