Gidel

Boost your vision!

ProcFG™

, and Frame Grabber and ImageProcessing System

Data Book

January 2024

Gidel **products and their generated products are not designed, intended, authorized, or warranted to be suitable for use in life-support applications, devices or systems or other critical applications**

Gidel
Boost your vision!

Contents

Figures

Tables

Scope

The purpose of this Data Book is to provide architectural, hardware and installation information as well as a basic User Manual for the ***Gidel ProcFG***™ system.

The Data Book is organized into the following chapters:

1. **Introduction** – system overview and key features.
2. **System setup** – system requirements and setup procedures.
3. **Getting Started** – GUI overview and quick-start tutorial.
4. **ProcFG Application** – detailed description of GUI features, functions, and parameters.
5. **CameraConfig and GenTL** – detailed description of the CameraConfig application and Gidel's GenTL API.
6. **Image Processing**– explanation on how to integrate Gidel and user IPs.
7. **API** – description of ProcFG API methods.
8. **References** – list of reference documents.
9. **Glossary** – definitions of terms and acronyms.
10. **Revision History**– document history.

1.1. Introduction

Gidel's frame grabber families are based on FPGA technology enabling both high-performance image acquisition and custom real-time image processing. The Gidel frame grabbers interface via PCIe with standard PCs enabling co-processing and convenient system development and integration.

The **ProcFG** suite provides the infrastructure for grabbing and implementing the user's image processing blocks on Gidel's frame grabber boards. For users who want to use the Gidel frame grabbers for pure high-performance grabbing, the ProcFG is provided as a plug-and-play solution ready for use with an application GUI as well as an API suite with examples for developing a custom user application. The ProcFG currently supports grabbing from Camera Link, GigE Vision and CoaXPress cameras as well as the ability to customize the grabbing to any user-defined camera protocol.

The ProcFG infrastructure implements GenTL (GenICam's Transport Layer) API as defined by the GenICam GenTL standard, for camera configuration and image streaming. Using Gidel's GenTL the ProcFG can be used to interface with 3rd party GenTL compatible image processing software packages. **MVTec** is an official partner of Gidel and its **Halcon** machine vision software has been verified to be fully compatible with the ProcFG. For information on additional compatible 3rd party software packages, please contact the Gidel Support (support@gidel.com).

The ProcFG system has the following unique qualities:

1. An open architecture enabling customization of the image acquisition and image processing.
2. Flexibility to tailor the system to non-standard user camera via interface and protocol customization.
3. Flexibility to select on-the-fly image Regions Of Interest (ROIs) for efficient data transfer to Host computer where further processing can take place. High-performance processing and ease of use are achieved by combining on-FPGA processing with software processing.
4. Capability to extract multiple ROIs per Line camera's strip or per Area Scan camera's frame, including the ability to extract overlapping ROIs.

5. ROI frames may be extracted according to reference coordinates enabling alignment with reference image even when there is image shifting, scaling, rotation, etc.
6. Support for acquisition from both Line and Area Scan cameras. Line camera's image strip is partitioned into frames that may be overlapped. Frame overlapping enables localized filtering to process each pixel using neighboring pixels within its own frame.

The presentation in this Data Book is initially from the perspective of the ProcFG GUI enabling clarifying basic concepts and understanding the ProcFG functionality. The later part of the Data Book addresses the user FPGA code integration and software application development using the ProcFG API.

1.2. System Overview

The ***Gidel ProcFG*** suite consists of a PCI/e FPGA frame grabber board, ProcFG GUI Application, API suite for developing user application, ProcWizard Development Software for implementing image processing blocks on FPGA, application examples and FPGA templates. Based on reconfigurable FPGA technology, the acquisition and processing blocks can be designed by the user to implement a customized vision/imaging system. The following figure shows a typical system implementation.

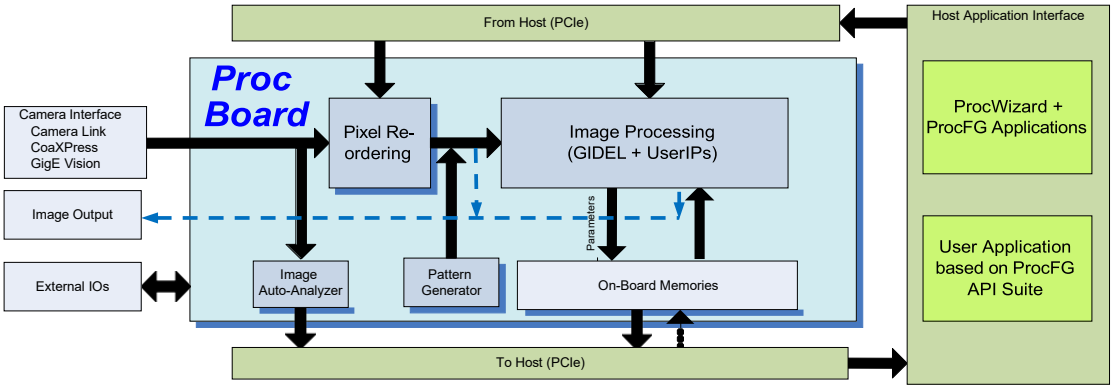


Figure 1: ProcFG Typical System Example

The ProcFG's acquisition path is from the source camera to the Proc Board and finally to the Host computer via the PCIe interface. The Proc board has the function of capturing and processing the incoming image data. The ProcFG application or user application on the Host computer has the function of controlling the Proc board's acquisition mode and processing. The Proc board can transfer from the on-board memory to the Host computer whole image data or alternatively offload image Regions Of Interest (ROIs) for further processing and according to image analysis, request on-the-fly additional ROIs.Using the ProcFG application or API-based user application, the image data can be displayed, stored and/or further processed.

The ProcFG supports Camera Link, CoaXPress, and GigE Vision image capture including all configurations as defined by the Camera Link and CoaXPress standards. Image acquisition includes automatic image parameter recognition, e.g., frame size, frame count, etc. In case of CoaXPress, the frame format (e.g., YCbCr601) and sub-format (e.g., 444) are automatically recognized. The ProcFG system automatically detects whether the image data is from a Line or Area scan cameraand operates accordingly.

The system's processing is fully configurable. In Figure 1, for example, the pixel data continues to the *Image Processing A Unit* implementing algorithms that process incoming pixel data prior to storing them in the on-board memory. The processing may include shading, LUT, etc. *Image Processing B Unit* implements algorithms that may process data at later stages and may operate on tiles (much shorter line widths), thus overcoming the FPGA's main bottleneck – the limited on-chip memory capacity.

The ProcFG system has two basic operating modes:

1. **Static Predefined Capture Configuration**

   In this capture mode, the image acquisition and processing configuration is predefined in a static manner by the software application, and thereafter, the ProcFG captures images according to this configuration. This mode includes two capture variations:

   a. **All images**: all acquired images, or parts of all the images (i.e., ROIs) are continuously transferred to the Host computer in a FIFO order.

   b. **Latest Image**: only the latest acquired image or part(s) of the latest acquired image is transferred to the Host computer.

2. **Dynamic, On-The-Fly Capture Configuration**

   In this capture mode, the image acquisition and processing is dynamically configured on-the-fly enabling analyzing incoming data and intelligently selecting the next requested image data. Image data can be transferred in any desired time-stamp order, including transfers of multiple ROIs per image, and transfers of ROIs of varying sizes and locations. In the API, the **Dynamic** capture mode is called **ROIList** mode. This mode can be entered in two ways:

   a. By a **User Application:** This is the real **Dynamic API capture** mode. ROIs are dynamically requested by the user application using the ProcFG API method QueueROI().

   b. Using the **ProcFG GUI:** The user has to select **ROI List Mode** in the Acquisition tab. ROIs are captured according to a pre-created (.ini) file. Although not truly dynamic, it is used for simulating the Dynamic API capture mode.

The ***ProcFG*** is comprised of the following components:

- ***Frame Grabber Board***: An FPGA based board that interfaces with the Host computer via a PCIe bridge. Currently the frame grabber families include Camera Link, CoaXPress, GigE Vision, and Custom grabbers.
- **Application software**: An intuitive GUI enabling to capture, process, display and save Camera Link and CoaXPress images. The processing includes capture of software-defined Region(s) Of Interest (ROIs).
- **API methods**: A set of ProcFG API methods that can be used to develop a customized user application that is fully compatible with the system design needs.
- **Camera Configuration software**: A GUI enabling to configure CoaXPress cameras connected to Gidel Proc boards.
- **GenTL API**: A set of API methods implementing GenICam's Transport Layer Interface for CoaXPress and GigE Vision, that can be used by 3rd party applications.
- **Proc Developer Kit™**: Tools for integrating Gidel and user's IPs enabling development of a custom imaging/vision system. The Developer's kit includes the ProcWizard development software, Gidel IPs and a API library.

1.3. Key Features

- Supports Camera Link Base, Medium, Full and 80-bit (single-zone only) configurations (“Deca” / “Full Plus”). Note: for multi-zone 80-bit configuration, contact Gidel.
- Support for a variety of Camera Link image formats: Mono, Bayer, RGB and RGBA (8, 10, 12, 14 and 16 bits/color)
- Supports CoaXPress cameras and camera simulators conforming to CoaXPress Standard specifications Version 2.0, including the following CoaXPress image formats: Raw, Mono, Bayer, RGB, RGBA, YUV, YCbCr601 and YCbCr709 (8, 10, 12, 14 and 16 bits/color). **Note:** in the current ProcFG version, Planar format is not supported. For Planar support, contact Gidel.
- Configuration of CoaXPress cameras.
- Flexibility to meet camera(s) and processing needs by selecting a fitting combination of Gidel Proc Board and Daughter Board(s).
- Open architecture allowing integration of:
  - ✓ User algorithm in FPGA supported by auto-generated API addition.
  - ✓ Mixed imaging protocols.
  - ✓ Combination of image acquisition and image transmission.
- Dynamic configuration of Camera Link parameters (such as number of zones, zones direction and parallel pixels) and CoaXPress parameters.
- Multi-operation modes:
  - ✓ Conventional grabbing modes (preset scheme).
  - ✓ Selective ROI grabbing using on-the-fly requests from user application.
- Automatic partitioning of line scan (and large Area scan) camera input into frames, with frame overlapping option. Frames may be grabbed in reference to global coordinates enabling easy comparison with reference images.
- Off-loading processing from the host CPU to the FPGA allowing real-time and affordable processing.
- Dynamic multiple ROI capture for both Line and Area scan cameras.
- API suite allowing customized application interface.
- Supported by Gidel Imaging Library (GIL) enabling to integrate Gidel field-tested image-processing IPs.
- Future extension to other imaging standards such as DVI, SDI, etc. (by replacing the interface daughter board).

2.1. System Requirements

The ProcFG system requirements are as follows:

- ✓ Operating system:
  - Windows 11 and Windows 10, 64-bits
  - Linux (kernel 2.6.x- 5.19.0 based on Proc 9.8.8) 64 bit. For 32-bit version,please contact the Gidel technical support.
- ✓ HawkEye-CL, HawkEye-cXp, or HawkEye-GigE boards.
- ✓ Optional 8 GB or 16 GB SODIMMs. For list of compatible Gidel certified SODIMMs, refer to the Gidel SODIMM Datasheet.
- ✓ Standard Camera Link or CoaXPress cables.

2.2. Installing the ProcFG system

For information regarding the ProcFG installation, refer to the *ProcFG packing list and installation* document located in the distribution disk, and in the following default folder: ***C:\Program Files\Common Files\Gidel\Doc\Frame Grabbers\ProcFG***

3.1. Overview

The ***ProcFG*** application provides an intuitive GUI enabling configuring the Proc Board's image acquisition and processing, transferring selectively processed image data to host computer, and displaying and saving image data.

The ProcFG application may serve as a developer's platform or as an application for initial familiarization with the ProcFG system prior to developing a customized application using the ProcFG API methods. In addition, the underlying FPGA design may be reconfigured according to the user design to incorporate customized acquisition flow and customized processing blocks.

Figure 2: ProcFG GUI

The GUI is composed of **Configuration Tabs**, **Image** and **Status Bar Display**, **Menus**, **Tool Bar**, and **Frame Information** as shown in **Figure 2** and briefly described below:

1. **Configuration Tabs** configure the ProcFG. The tabs include:
    a. **Acquisition Tab:** sets Grabbing modes, number of frames to capture, etc.
    b. **ROI Tab:** defines Image Region(s) Of Interest.
    c. **Camera Tab:**
        - **Camera Link Tab** (in case of Camera Link): sets Camera Link parameters, including Format, Sub-Format, bits per color and Taps configuration.

d. **Misc. Tab:** configures output options and memory buffers.

2. **Image** and **Status Bar Display** show the captured image frames and real-time acquisition status.
3. **Menus** and **Tool Bar** allow general File management and frame grabbing control.
4. **Frame Information** shows the acquired image size (Width and Height), the Format, and Bits Per Color. This is the image information as it is transmitted by the camera and captured in the Proc Board's memory; it is not necessarily the image size transferred to the Host computer as for example in case of an ROI. In case of line camera, Height=1. Below the Frame Information, the Zoom factor is displayed.

3.2. Quick-Start Tutorial

This section will enable you in just a few bare-essential steps to get started with the ***ProcFG*** application. The ProcFG GUI is explained in more detail in chapter 4.

Prior to starting the tutorial, you must setup your system as detailed in chapter 2.

To begin capturing image frames, do the following:

1. Run the ***ProcFG.exe*** application.
2. If multiple Proc boards are installed on the host computer, you will be prompted to select a board (Figure 3). Select your board of choice and click **OK**.

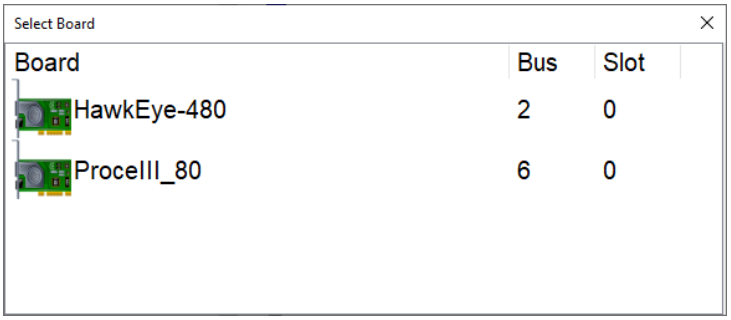


Figure 3: Board Selection

3. First, configure the ProcFG via the **Configuration Tabs** located on the right side of the GUI.
4. In the **Acquisition Tab,** select **All Images** to capture incoming camera images continuously in a FIFO order. Set the **No. of frames / lines to grab** to '0' for unlimited amount of image capture.
5. In the **ROI Tab,** select the **Enable ROI** and define the ROI size via the **Width** and **Height** parameters. If you want to offset the ROI relative to the image's first pixel, define the **OffsetX** and **OffsetY** parameters. If you are using an Area camera, you have just defined the initial ROI that will be extracted from each incoming image frame. If you are using a Line camera, then the ROI is equivalent to a frame extracted from the strip of incoming image data; frames will be extracted from the strip in a continuous manner. For Line camera, **Height** parameter defines the number of lines per frame. Clicking the **Advanced Mode...** button opens advance mode settings for capturing multiple ROIs/frames. For

now, we will only capture a single ROI/frame per incoming image, so do not click the **Advanced Mode...** button.

6. In case of **Camera Link**, in the **Camera Link Tab,** select or clear the **Ignore DVAL** and **Ignore FVAL** depending on whether your camera uses the DVAL and FVAL signals, respectively. For area cameras, FVAL must be used, i.e., the **Ignore FVAL** must be cleared. In the Camera area, select the Format, Sub-Format and the Bits Per Color. In the **Taps** area, define the number of image zones and the number of parallel pixels per zone. Finally, in the **Zones direction** area, click each zone's arrow button to set the pixels direction. The Camera Link setting should be identical to your camera's settings.
7. In the **Misc. Tab,** select **Display Images** and set the **Display every** box to 1, indicating that every incoming frame/ROI will be displayed. For now, we will not save the captured images, so clear the **Save Raw Images** check box. In addition, we will not log the ProcFG's execution process, so in the **Log** area, set **Verbosity Level** to **OFF**. Finally, set **No. of Buffers** to '3', meaning that three frame buffers are allocated in the Host computer's RAM.
8. In the **File** menu, select **Save Configuration As...** to save your current configuration in a new configuration file (.**gxfx**).
9. **IMPORTANT NOTE!**
   In case of cXp, after Power OFF/ON and after SOF loading, you must run a Camera Configuration application before ProcFG can perform grabbing (capturing images).
10. Assuming that your camera is connected properly and free running, you are now ready to begin capturing images. On the Toolbar, click  to begin image acquisition. You should now see the captured ROI/frame images displayed in the ProcFG GUI display window. If you would like to slow down the image display rate, in the **Acquisition Tab** set **Sleep Between Frames** to 100 ms. You can zoom in and out via the mouse wheel and pan by dragging the image with the Left mouse button pressed. Alternatively, you can zoom and pan via the keyboard as follows: **zoom** by "+" or "–", **pan** by the four arrow keys. Note that left clicking the mouse sets a new zoom center that is marked by a cross cursor (). The current **Zoom factor** is displayed in the lower-right side of the GUI.
11. Look at the **Status bar** at the bottom of the GUI and note number of incoming frames (**In**), frame number (**Frame #**), number of fames displayed (**Grabbed**), on-board memory usage (**Mem)**, etc.
12. In the toolbar, try clicking  (Previous Frame/Next Frame buttons) to pause the image acquisition and display the frames currently stored in the Proc board's memory. To continue image acquisition, click  (Continue).
13. Play around with different configurations. Each time you make a configuration change, you must click  to abort the image acquisition and clear the memory buffers. To apply the changes and restart the Acquisition process, click  .

   For further details regarding each of the GUI's functions/parameters, refer to chapter 4.

4. ProcFG Application

The following chapter provides a detailed description of the ProcFG application software and its advanced operation modes.

4.1. The Application Menues and Toolbar

The ProcFG is composed of three menus: **File**, **Control** and **Help.** Most of the menu items are supported by Toolbar buttons allowing convenient and quick GUI control.

4.1.1. File Menu

The File menu has the following items:

Table 1: File Menu

Submenu	Tool- bar Button	Description	Shortcuts
Open Configuration		Opens a ProcFG configuration that has an extension .**gclfg**. A **.gclfg** file contains all the Frame Grabber's settings that were previously configured via the **Configuration Tabs** and saved. The **Configuration Tabs** will be set according to the **.gclfg** file.	Alt-F-O
Save Configuration		Saves changes made to current .**gclfg** configuration file.	Alt-F-S
Save Configuration As...	None	Saves the current acquisition settings in a .**gclfg** configuration file.	Alt-F-A
Save Image		Saves an image to file when the system is in a Pause state due to Next Frame/Previous Frame mode.	Ctrl+S, Alt-F-I

4.1.2. Control Menu

The Control menu has the following items:

Table 2: Control Menu

Submenu	Tool-bar Button	Description	Shortcuts
Grab		Initiates image capture.	G or Alt-T-G
Pause Display		Pauses image display only. The transfer of image frames from the Proc board's memory to the Host computer continues operating uninterruptedly.	Space/C or Alt-T-C
Continue		Continues displaying the grabbed images when the image display has been paused. Also, continues grabbing and displaying when grabbing and image display has been paused by the **Previous Frame** or **Next Frame** operations.	Space/C or Alt-T-C
Previous Frame		Displays the previous frame in the Proc board memory. Clicking this button automatically sets the image acquisition in pause. The status bar (left side) will indicate "**Pause**". The number of total frames that can be viewed is respective to the image size and the on-board memory size. In order to save an image when in pause state, click  (Save Image).	PageUp or Alt-T-P
Next Frame		Displays the next frame in the Proc board memory. See comments in "Previous Frame" row.	PageDown or Alt-T-N

Stop		Stops the image acquisition process, but continues to transfer image frames that have already been captured on the Proc board's memory until the memory is emptied.	S or Alt-T-S
Abort		Aborts the image acquisition, including the transfer of images that have already been stored in the Proc board's memory.	A or Alt-T-A
Display Data		Opens/hides a window displaying the hexadecimal representation of a selected area of the currently displayed image. Pressing the Display Data button, opens/hides the Frame Data window. Clicking the right mouse button on a specific image pixel, displays the hexadecimal data of an area around that pixel. The window can be hidden by clicking the Display Data button, or when the Frame Data window is in focus by pressing **Esc** or the top-right **x** of the Frame Data window.	D or Alt-T-D toggles show/hide window; clicking **Esc** when the Display window is in focus will hide the window
Statistics Data		Opens/hides a window displaying statistics data on pixels in a parallelogram ROI. To draw a new ROI, in Statistics Data mode drag the mouse with the right button pressed on the ProcFG image. When the Right button is released, a **Parallelogram ROI** is displayed. See additional details in ***section*** 4.4.	T or Alt-T-T
Histogram		Opens/hides a window displaying a histogram for pixels in an ROI drawn on the displayed images. In addition, limited Statistics Data is displayed. See additional details in ***section*** 4.5.	H or Alt-T-H

Profile		Opens/hides a window displaying a graphical presentation of pixel values along a line. To draw a line, drag the mouse with the right button pressed on the ProcFG image in Profile mode. See additional details in ***section 4.3***	F or Alt-T-F
Grabber Reset		Resets the system IPs, including Gidel and user IPs (e.g., design algorithm), to default parameters.	Alt-T-R

4.1.3. Help Menu

The Help menu has the following items:

Table 3: Help Menu

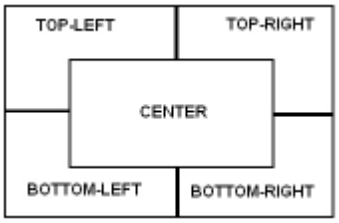
Submenu	Toolbar Button	Description	Shortcuts
ProcFG Data Book	NA	Opens the ProcFG Data Book.	Alt-H-D
ProcFG Quick Guide	NA	Opens the ProcFG Quick Guide showing a summary of the ProcFG shortcuts.	Alt-H-Q
About ProcFG	NA	Provides ProcFG version information.	Alt-H-A

4.2. Image Display Window

The Image Display Window displays the images or regions of interest (ROIs) of images that have been captured and currently reside in the Proc board's on-board memory buffer. The image displayed will be according to the grabbing mode as detailed in ***section 4.7.1 Acquisition Tab***. For further information on ROI, please refer to ***section 4.7.2***.

The Display window is supported by zooming and panning. The following table summarizes the zoom and pan controls:

Operation	Keyboard	Mouse
Zoom in	+	Rotating the Wheel Forward
Zoom Out	-	Rotating the Wheel Backward
Rectangular Zoom	NA	Dragging with the Right button pressed
100% or Full Window	Ins	Double-clicking the Left button
Define Zoom Center	NA	Clicking the Left button will mark the zoom center with a cross cursor (+); the displayed image will be **shifted** so the pixel under the cursor will be displayed at the center of the window. If the pixel under the cursor is near the image border(s), it will not be displayed at the window center, until the zoom is big enough. That pixel will **remain** the center for subsequent zooming operations increasing/decreasing the zoom by the wheel, or by the keyboard buttons: clicking the "+", "-" or double clicking. The zoom center will **change** on other zooming operations (Pan, rectangular zoom, and move to corner/center), and when clicking Grab. Note that the Zoom Center cursor (+) is indicated on the image, during and shortly after zoom operations. In case of zoom operations when the image is static (when the GUI is paused or after Next/Previous frame clicks), in order to hide the Zoom Center cursor, the user has to click **Esc**, or to return to normal Grab mode (using Continue).

Operation	Keyboard	Mouse
Pan/Move Zoom Center	4 Arrows	Dragging with the Left button pressed. If the image is moved horizontally and/or vertically, the Zoom center stays at the center of the window along X and/or Y. If the image is not moved horizontally and/or vertically, the Zoom center moves along X and/or Y. In both cases, the new Zoom center is indicated by the Zoom cursor.
Move to Corner/ Move to Center	Pressing the **Home** key moves the Zoom center to the image center or to one of the image corners. Each press moves the center to the next point out of the 5 possible points (image center or corner).	Double-clicking the Left button with the Ctrl Key pressed, the displayed image is either centered or moves to one of the corners, depending on the curser's location as follows:  If the current zoom is small (e.g., if the whole image is displayed on the screen), the image will not move to the corner, until the zoom is big enough.

Note that there are situations that not all captured images will be displayed. If the images are not transferred quickly enough from the on-board memory to the Host computer memory's there will be data loss as incoming image data will overwrite existing image data in the on-board FIFO buffer. In this situation, the message "***Memory OVERFLOW, Missed [XX] frames.***" will appear in the Status bar.

Similarly, when reducing the image transfer rate by using the **Sleep Between Frames [ms]** (see ***section 4.7.1 Acquisition Tab***), some of the images that are captured by the Proc board may be lost depending on the transfer sleep duration.

4.3. Profile

The Profile tool provides a graphical presentation of pixel values along a line. To enter Profile mode, click the Profile button  or select **Profile** from the Control menu. To exit Profile mode, you can perform one of the following actions: click the Profile button, select **Profile** from the Control menu, click the "**x**" on the Profile window, or press **Esc** when the Profile window is in focus.

To draw a line on the ProcFG image in Profile mode, drag the mouse with the right button pressed. To draw a horizontal or vertical line, drag the mouse with the right button pressed and with the **Shift** key pressed. Note that while ProcFG is not in Profile mode, when you drag the mouse with the right button pressed, Rectangular Zoom is activated.

When you move the mouse (without any button pressed) along the profile line (see Figure 4), a special cursor is displayed on the profile line, and a corresponding vertical purple line is drawn on the profile window.

On the top-right corner of the profile window, information on the pixel under the cursor is displayed (see Figure 5): the point number along the line, the X, Y pixel coordinates, and the value(s) of the displayed pixel color component(s).
The second line on the top-right corner displays statistics information on each of the color components of the latest pixel values in the current cursor position: minimum, maximum and standard deviation. Up to 32 pixels in the current cursor position are taken into account.

You can display the history of the latest profiles on top of each other. Up to 8 profiles are displayed, starting with the oldest profile. The latest (current) profile is displayed last, in brighter colors (see Figure 6 and Figure 7). To display or hide the profiles history click **Shift-Y** or **Alt-Y**.

In case of color images (e.g., RGB or RGBA), you can select which color is displayed in the Profile window. Each time you click **Shift-C** or **Alt-C**, the displayed color is changed in the following repeated sequence: All colors, Red, Green, and Blue; In case of RGBA, the fourth color is also displayed; In case of YUV, YCbCr601 and YCbCr709, Y is displayed in white.

You can increase the profile scale factor around a vertical center. The default scale factor is 100%. For 8 bits per color, the default vertical axis range is 256, for 10 bits per color it is 1024, etc. The vertical axis range changes according to the scale factor. When you increase the scale factor, the vertical axis range is decreased. For example, when the scale factor is 200%, the vertical axis range is 128.

The default vertical center is half of the vertical axis range. For 8 bits per color, it is 128. For 10 bits per color, it is 512, etc. You can increase or decrease the

vertical center by 1/256 of the default vertical axis range. For 8 bits per color, the increase/decrease is 1. For 10 bits per color, it is 4, etc.

You can move the profile line using the four arrows with the **Ctrl** key pressed. Each time, the line is moved by one pixel or one line to the direction corresponding to the pressed arrow key.

You can Zoom in (decrease) and out (increase) the profile line via the mouse wheel or via the keyboard using “+” or “-”. You can Pan the profile line along itself using the 4 Arrows or by dragging with the Left button pressed.

Note that ProcFG hides the profile window each time you perform an operation that does not allow updating the profile, e.g., changing the zoom, dragging the image, etc. ProcFG stays in Profile mode, so you can draw a new profile line even though the profile window is not displayed. While resizing the ProcFG window, ProcFG hides the profile window. Click the **Profile** button to show the profile window.

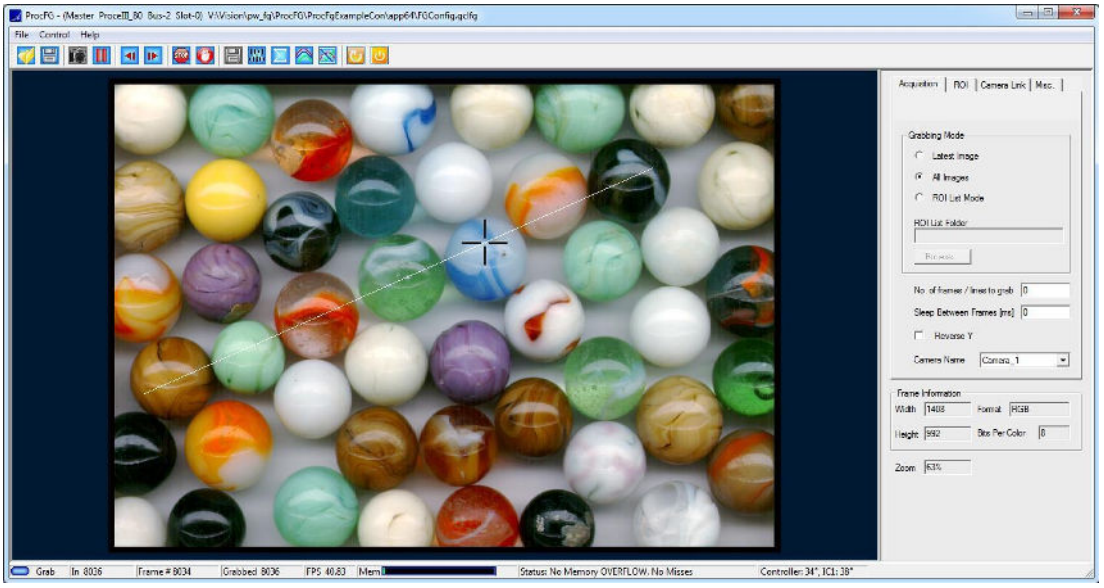


Figure 4: Profile line and Profile cursor

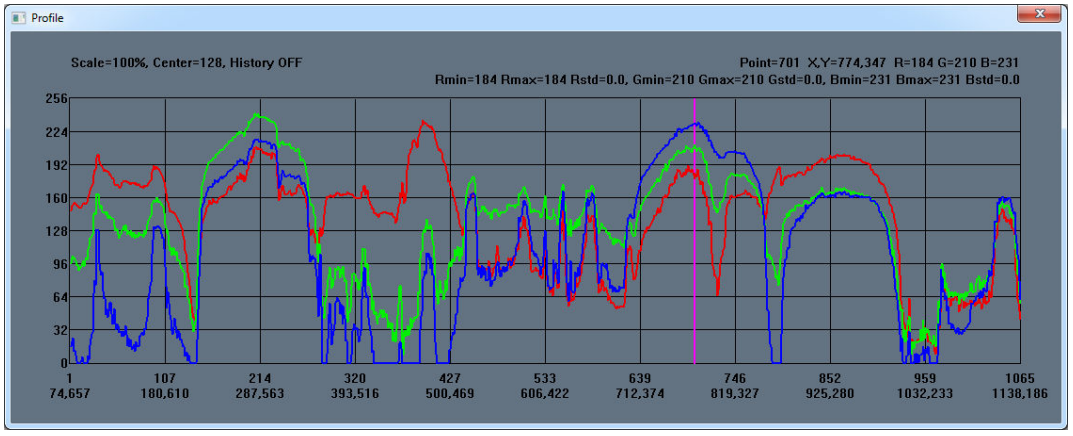


Figure 5: Profile Window

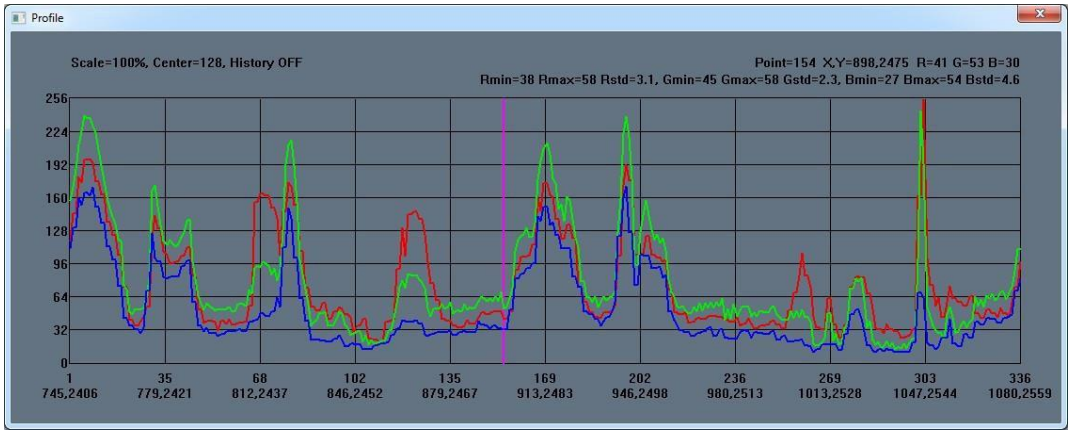


Figure 6: Profile Window without History

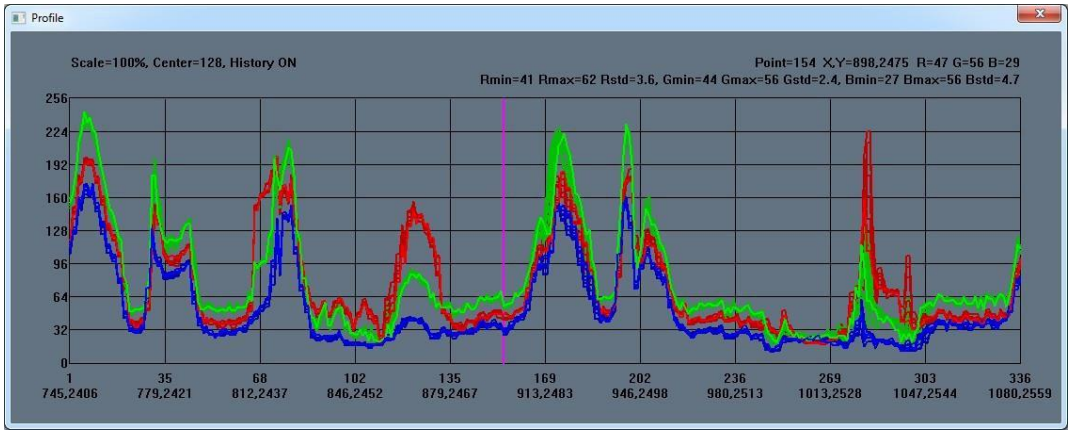


Figure 7: Profile Window with History

The following table describes the Profile controls:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Draw Line	NA	NA	NA	Drag with the Right button pressed**
Increase / Decrease Scale Factor	NA	Shift + / -	Alt-I / Alt-D	NA
Vertical Center Up / Down	NA	Shift- Up / Down Arrow	Alt-U / Alt-N	NA
Move Line parallel to itself Up / Down / Left / Right	NA	Ctrl- 4 Arrows	NA	NA
Profile Line Zoom In / Out	NA	+ / -	NA	Rotate the Wheel Forward / Backward
Pan Profile Line along itself	NA	4 Arrows	NA	Drag with the Left button pressed
Change Color	NA	Shift-C	Alt-C	NA
Show / Hide Profile History	NA	Shift-Y	Alt-Y	NA
Reset Parameters	NA	Shift-Home	Alt-Home	NA
Show Profile		F	Alt-T-F	NA
Hide Profile		F/**Esc***	Alt-T-F	Press "**x**" on the top-right corner*

* **Esc** and "**x**" function when the Profile window is in focus

** To draw a horizontal or vertical line, drag with the **Shift** key pressed

4.4. Statistics Data

The Statistics Data tool allows displaying Statistics Data for pixels in an ROI drawn on the displayed images. The ROI shape is a **parallelogram**, where the top and bottom lines are always horizontal. The Statistics Data includes, among others, Average and Standard Deviation for each color component of pixels in the ROI, as well global information on the ROI itself (e.g., the number of pixels).

In order to draw a new ROI, Drag the mouse with the Right button pressed. When the Right button is released, a **Parallelogram ROI** is displayed. You can move the ROI and change its parameters (e.g., Width and Height) using the Keyboard and the Mouse.

To enter Statistics Data mode, click the Statistics Data button  or select **Statistics Data** from the Control menu. To exit Statistics Data mode, you can perform one of the following actions: click the Statistics Data button, select

Statistics Data from the Control menu, click the "**x**" on the Statistics Data window, or press **Esc** when the Statistics Data window is in focus.

Here is an example of the displayed Statistics Data Window:

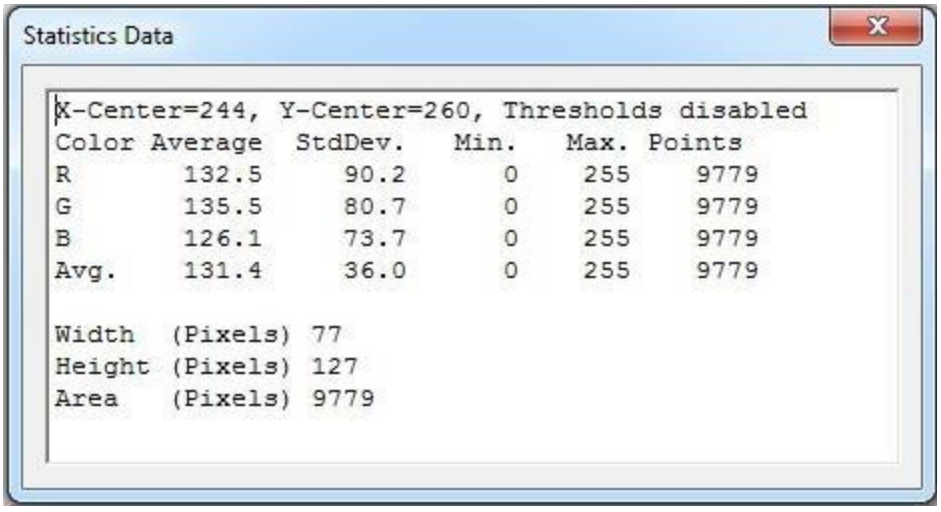


Figure 8: Statistics Data Window

Notation:

- **X-Center**, **Y-Center**: The ROI center coordinates in pixels.
- **Color**: R/G/B, Y/U/V, Y/Cb/Cr, Mono or Avg.: Color components.
  Avg. is the average of the color components R, G, B (and A, in case of RGBA) in each pixel. It is displayed only in case of RGB and RGBA.
- **Average**: Average of the color component* values for pixels where **all** color component values are within the Thresholds**.
- **StdDev.**: Standard Deviation of the color component* values for pixels where **all** color component values are within the Thresholds**.
- **Min., Max.**: Minimum and Maximum color component* values for pixels where **all** color component values are within the Thresholds**.
- **Points**: Number of points of the color component* for pixels where **all** color component values are within the Thresholds**.
- **Width (Pixels)**: The ROI horizontal Width in Pixels.
- **Height (Pixels)**: The ROI vertical Height in Pixels.
- **Area (Pixels)**: The number of Pixels within the ROI (**ignoring** the Thresholds).

* The **Avg**. row (displayed only in case of RGB and RGBA), is treated as any other color component.

** **Thresholds** are automatically calculated at 1% and 99% of the allowed range. For 8 Bits Per Color, they are 3 and 252 (255*0.01 and 255*0.99). If using Thresholds is enabled, only pixels where **all** their color component

values are within the thresholds are taken into account in the statistics calculations.

To draw a **new ROI**, **drag** with the Right button pressed. While dragging with the Right button pressed, a **line** representing the ROI **center line** is displayed on the screen. To draw a horizontal or vertical line, Drag with the **Shift** key pressed.

When the Right button is released, a corresponding **Parallelogram ROI** is displayed. The ROI **Center line** coincides with the drawn line (that is deleted). The (vertical) **Height** of the ROI is identical to the vertical component of the drawn line length (i.e., to the line length along Y). The (horizontal) **Width** of the ROI is the previously used Width. If the drawn line is close to **horizontal/vertical**, the ROI shape is **Rectangular**.

If when the Right button is released there are end point(s) of the ROI **outside** the boundaries of the displayed image, the ROI size is decreased so that the whole ROI is **inside** the boundaries. If possible, the ROI **Center line** coincides with (or is parallel to) the drawn line. if the ROI Width is decreased, it is not used when a new ROI is drawn by dragging with the Right button pressed.

Operations on the ROI:

- To **Move** the ROI **parallel** to itself Up / Down / Left / Right, use **Ctrl-4 Arrows**, or **Drag** with the **Left button pressed**.
- To **Pan** the ROI along its center line, use the **4 Arrows**.
- To Increase / Decrease the **ROI** (horizontal) **Width**, use **Shift Right / Left Arrow**, or **Alt-W / Alt-D**. The ROI center line and the ROI Height are not changed.
- To Increase / Decrease the **ROI** (vertical) **Height**, use **Shift Up / Down Arrow**, or **Alt-E / Alt-G**. The ROI center line is extended/decreased but not moved. The ROI Width is not changed.
- To **Rotate** the ROI center line clockwise / counterclockwise, keeping the top and bottom lines horizontal, without changing the ROI Height or the ROI Width, use **Shift + / -**, **Shift-R / Shift-L**, or Rotate the **Wheel Backward / Forward** with the **Control key pressed**.
- To **Zoom** the ROI **In /Out**, use **+ / -**, or Rotate the **Wheel Forward / Backward**. The ROI Width and Height change in the same percentage, keeping the aspect ratio.
- To Enable / Disable using the **Thresholds**, use **Shift-T**.
- To **Reset** the ROI parameters (Width, Height, Angle), use **Shift-Home** or **Alt-Home**.
- To **Show** Statistics Data, Click the "**Statistics Data**" button, or press T.

- To easily draw an ROI covering the whole image, you can first draw a horizontal ROI covering the whole image width, and then draw a vertical ROI covering the whole image height.
- To **Hide** Statistics Data, Click the "**Statistics Data**" button, press T/Esc*, or Press "**x**" on the top-right corner*.
  * **Esc** and "**x**" function when the "**Statistics Data**" window is in focus.

Note: If as a consequence of changing the ROI one or more end point of the ROI gets **outside** the boundaries of the displayed image, the ROI is **not** changed so that the whole ROI is **inside** the boundaries.

The following table describes the Statistics Data controls:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Draw Line	NA	NA	NA	Drag with the Right button pressed**
Move ROI parallel to itself Up / Down / Left / Right	NA	Ctrl-4 Arrows	NA	Drag with the Left button pressed
Pan ROI along its center line	NA	4 Arrows	NA	NA
Increase / Decrease ROI Width	NA	Shift Right / Left Arrow	Alt-W / Alt-D	NA
Increase / Decrease ROI Height	NA	Shift Up / Down Arrow	Alt-E / Alt-G	NA
Rotate ROI center line clockwise / counterclockwise	NA	Shift + / - Shift R / L	NA	Rotate the Wheel Backward / Forward with the Control Key pressed
Zoom ROI In / Out	NA	+ / -	NA	Rotate the Wheel Forward / Backward
Enable / Disable Thresholds	NA	Shift-T	NA	NA
Reset ROI Parameters	NA	Shift-Home	Alt-Home	NA
Show Statistics Data		T	Alt-T-T	NA
Hide Statistics Data		T/**Esc***	Alt-T-T	Press "**x**" on the top-right corner*

* **Esc** and "**x**" function when the Profile window is in focus

** To draw a horizontal or vertical line, drag with the **Shift** key pressed

4.5. Histogram

The Histogram tool allows displaying a histogram for pixels in an ROI drawn on the displayed images. In addition to the histogram, limited Statistics Data is displayed. Like in case of the Statistics Data tool, the ROI shape is a **parallelogram**, where the top and bottom lines are always horizontal.

The number of histogram bins (intervals) depends on the number of bits per color. In case of 8 bits per color, there are 256 bins. In all other cases, there are 1024 bins. The number of points (color values) represented by each bin varies. For example, in case of 12 bits per color there are 4 (4096/1024) points per bin.

To enter histogram mode, click the Histogram button  or select **Histogram** from the Control menu. To exit histogram mode, you can perform one of the following actions: click the Histogram button, select **Histogram** from the Control menu, click the "**x**" on the histogram window, or press **Esc** when the histogram window is in focus.

In order to draw, modify or move an ROI, use the Statistics Data GUI operations (see ***section 4.4***).

The displayed histogram window is similar to the Profile window, but in this case, the graphs and color values indicate bin counts. Here is an example of the displayed histogram window of a Bayer image and the corresponding limited Statistics Data window:

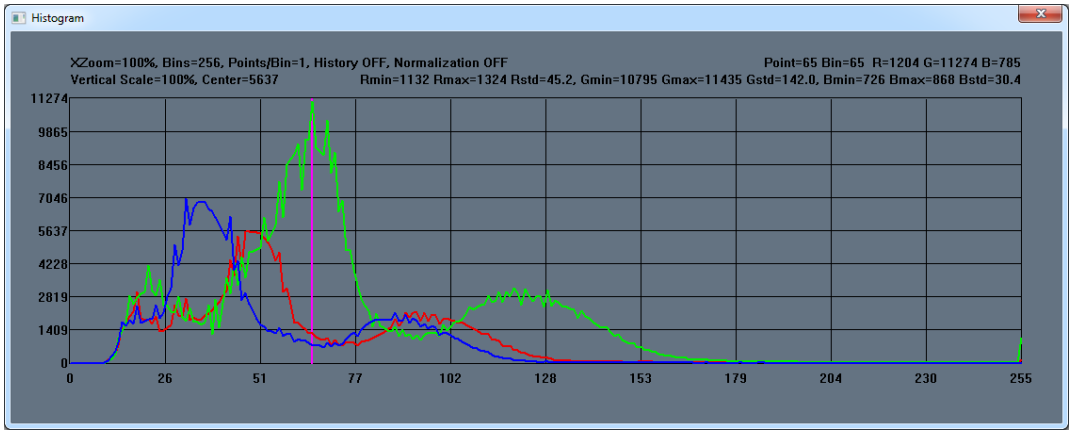


Figure 9: Histogram Window

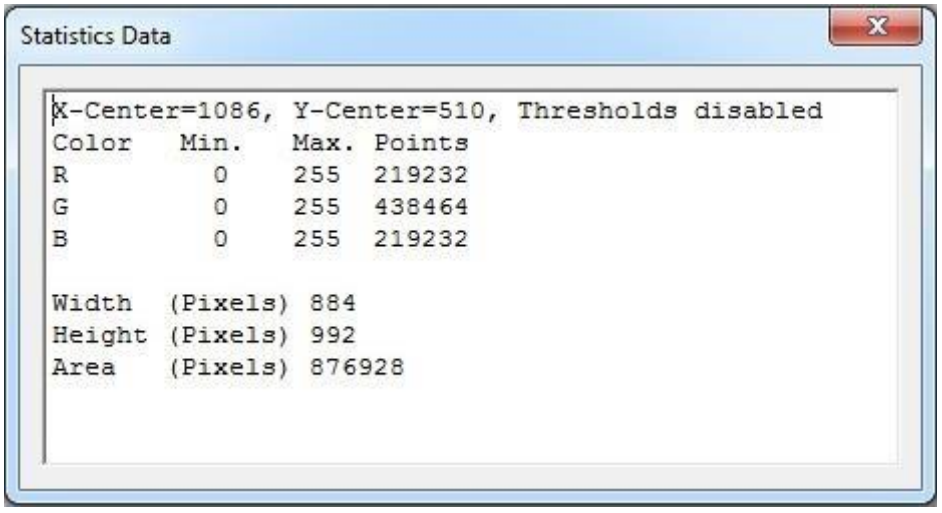


Figure 10: Corresponding Statistics Data Window

When the mouse is moved (without any button pressed) on the **displayed image**, a vertical purple line is drawn on the histogram window. The location of the purple line corresponds to the location of the mouse pointer on the displayed image. When the mouse pointer is located on the left side of the displayed image, the purple line is drawn on the left side of the histogram window, corresponding to the first bin. When the mouse pointer is located on the right side of the displayed image, the purple line is drawn on the right side of the histogram window, corresponding to the last bin.

The top value of the vertical scale of the histogram window depends on the maximal count in all bins. It is possible to increase / decrease the histogram vertical scale factor around a vertical center. This affects the top and bottom values of the vertical scale. The default scale factor is 100%. The histogram vertical center can be moved Up / Down. The default vertical center is half of the top value of the vertical scale.

It is also possible to increase / decrease the histogram Zoom along X and also move the histogram Left / Right. These features enable to focus on a subset of the bins.

In the case of Bayer format and 411/422 sub-formats, histogram normalization can be enabled / disabled. In these cases, certain colors appear more than others. In the case of Bayer format, the Green color appears two times more than the other colors. In case of 411/422 sub-formats, the Y color appears 4/2 times more than the other colors. When histogram normalization is enabled, the graph values of the indicated colors (Green or Y) are divided by 2/4, and the displayed values in the histogram text lines are almost not affected (they can be smaller by up to 1/3).

Figure 11 is an example of the displayed histogram window of a Bayer image with normalization enabled. Compare it to the corresponding histogram window with normalization disabled shown at Figure 9.

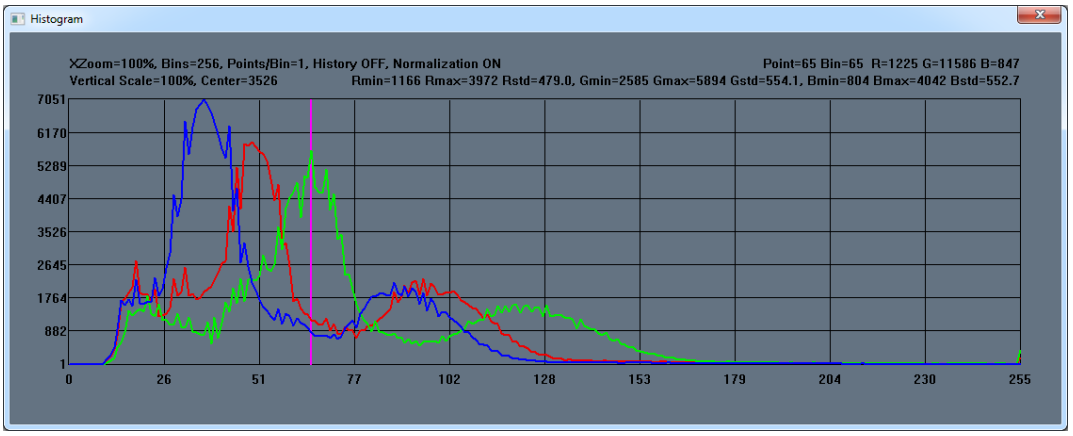


Figure 11: Histogram Window with normalization enabled

It is possible to reset either only the histogram parameters or the ROI parameters (without changing the histogram parameters).

Like in the Profile tool (see section 4.3), the Histogram color can be changed to display all colors or only a single color. In addition, the histogram history can be shown / hidden.

The following table describes the Histogram controls:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Statistics Data operations*	NA	*	*	*
Increase / Decrease Histogram Vertical Scale Factor	NA	Ctrl + / Ctrl -	Alt-W / Alt-D	NA
Histogram Vertical Center Up / Down	NA	Ctrl] / Ctrl [	Alt-U / Alt-N	NA
Increase / Decrease Histogram X Zoom	NA	NA	Alt Up / Down Arrow	NA
Histogram X Left / Right	NA	NA	Alt Left / Right Arrow	NA
Enable / Disable Histogram Normalization**	NA	Shift-N	NA	NA
Reset Histogram Parameters	NA	Ctrl-Home	NA	NA
Change Histogram Color***	NA	Shift-C	Alt-C	NA
Show / Hide Histogram History***	NA	Shift-Y	Alt-Y	NA

Show Histogram		H	Alt-T-H	NA
Hide Histogram		H/**Esc******	Alt-T-H	Press “**x**” on the top-right corner****

* All Statistics Data operations (except Alt-W / Alt-D) are available

** In case of Bayer format and 411/422 sub-formats

*** This is a Profile operation

**** **Esc** and “**x**” function when the Histogram window is in focus

4.6. ProcFG Quick Guide

The following tables summarize all the ProcFG operation shortcuts.

Grab operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Grab		G	Alt-T-G	NA
Pause Display		Space/C	Alt-T-C	NA
Continue		Space/C	Alt-T-C	NA
Stop		S	Alt-T-S	NA
Abort		A	Alt-T-A	NA

Zoom and Pan operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Zoom In	NA	+	NA	Rotate the Wheel Forward
Zoom Out	NA	-	NA	Rotate the Wheel Backward
Pan / Move Zoom Center	NA	4 Arrows	NA	Drag with the Left button pressed
Rectangular Zoom	NA	NA	NA	Drag with the Right button pressed
Define Zoom Center	NA	NA	NA	Click the Left button
Move to Corner / Center	NA	Home	NA	Double-click the Left button with the Ctrl Key pressed
100% / Full Window	NA	Ins	NA	Double-click the Left button

Next Frame / Previous Frame:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Next Frame		PageDown	Alt-T-N	NA
Previous Frame		PageUp	Alt-T-P	NA
Continue		Space/C	Alt-T-C	NA

Profile operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Draw Line	NA	NA	NA	Drag with the Right button pressed**
Increase / Decrease Scale Factor	NA	Shift + / -	Alt-I / Alt-D	NA
Vertical Center Up / Down	NA	Shift- Up / Down Arrow	Alt-U / Alt-N	NA
Move Line parallel to itself Up / Down / Left / Right	NA	Ctrl- 4 Arrows	NA	NA
Profile Line Zoom In / Out	NA	+ / -	NA	Rotate the Wheel Forward / Backward
Pan Profile Line along itself	NA	4 Arrows	NA	Drag with the Left button pressed
Change Color	NA	Shift-C	Alt-C	NA
Show / Hide Profile History	NA	Shift-Y	Alt-Y	NA
Reset Parameters	NA	Shift-Home	Alt-Home	NA
Show Profile		F	Alt-T-F	NA
Hide Profile		F/**Esc***	Alt-T-F	Press "**x**" on the top-right corner*

* **Esc** and "**x**" function when the Profile window is in focus

** To draw a horizontal or vertical line, drag with the Shift key pressed

Display (Hex) data operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Define Data Center	NA	NA	NA	Right click defines an image pixel. This pixel is used as the Data Center even after zoom operations, until it is redefined.*
Show Display Data		D	Alt-T-D	NA
Hide Display Data		D/Esc**	Alt-T-D	Press "x" on the top-right corner*

*After Move to Corner / Center (either via the mouse or via the Home key) and after Rectangular Zoom, the Data Center is changed to the new zoom center (image corner/center or rectangle center).

** **Esc** and "**x**" function when the Data Display window is in focus.

Statistics Data operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Draw Line	NA	NA	NA	Drag with the Right button pressed**
Move ROI parallel to itself Up / Down / Left / Right	NA	Ctrl-4 Arrows	NA	Drag with the Left button pressed
Pan ROI along its center line	NA	4 Arrows	NA	NA
Increase / Decrease ROI Width	NA	Shift Right / Left Arrow	Alt-W / Alt-D	NA
Increase / Decrease ROI Height	NA	Shift Up / Down Arrow	Alt-E / Alt-G	NA
Rotate ROI center line clockwise / counterclockwise	NA	Shift + / - Shift R / L	NA	Rotate the Wheel Backward / Forward with the Control Key pressed
Zoom ROI In / Out	NA	+ / -	NA	Rotate the Wheel Forward / Backward
Enable / Disable Thresholds	NA	Shift-T	NA	NA
Reset ROI Parameters	NA	Shift-Home	Alt-Home	NA
Show Statistics Data		T	Alt-T-T	NA
Hide Statistics Data		T/**Esc***	Alt-T-T	Press "**x**" on the top-right corner*

* **Esc** and "**x**" function when the Profile window is in focus

** To draw a horizontal or vertical line, drag with the **Shift** key pressed

Histogram operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Statistics Data operations*	NA	*	*	*
Increase / Decrease Histogram Vertical Scale Factor	NA	Ctrl + / Ctrl -	Alt-W / Alt-D	NA
Histogram Vertical Center Up / Down	NA	Ctrl] / Ctrl [	Alt-U / Alt-N	NA
Increase / Decrease Histogram X Zoom	NA	NA	Alt Up / Down Arrow	NA
Histogram X Left / Right	NA	NA	Alt Left / Right Arrow	NA

Enable / Disable Histogram Normalization**	NA	Shift-N	NA	NA
Reset Histogram Parameters	NA	Ctrl-Home	NA	NA
Change Histogram Color***	NA	Shift-C	Alt-C	NA
Show / Hide Histogram History***	NA	Shift-Y	Alt-Y	NA
Show Histogram		H	Alt-T-H	NA
Hide Histogram		H/**Esc****	Alt-T-H	Press "**x**" on the top-right corner****

* All Statistics Data operations (except Alt-W / Alt-D) are available

** In case of Bayer format and 411/422 sub-formats

*** This is a Profile operation

**** **Esc** and "**x**" function when the Histogram window is in focus

Open and Save operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Open Configuration…		NA	Alt-F-O	NA
Save Configuration		NA	Alt-F-S	NA
Save Configuration As…	NA	NA	Alt-F-A	NA
Save Image		Ctrl-S	Alt-F-I	NA
Open Recent Configuration	NA	NA	NA	NA
Exit	NA	Ctrl-X	Alt-F-X	NA

Reset Operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
Grabber Reset		NA	Alt-T-R	NA

Help Operations:

Operation	Toolbar	Keyboard	Alt keys	Mouse
ProcFG Data Book	NA	NA	Alt-H-D	NA
ProcFG Quick Guide	NA	NA	Alt-H-Q	NA
About ProcFG	NA	NA	Alt-H-A	NA

4.7. Configuration Tabs

The ***Configuration Tabs*** configure the frame acquisition.

Figure 12: Configuration Tabs

The ***Configuration Tabs*** are comprised of the following four tabs:

1. **Acquisition Tab** for setting acquisition mode.
2. **ROI Tab** for defining the captured images Region(s) of Interest.
3. **Camera Tab:**
   - **Camera Link Tab** (in case of Camera Link) for defining the Camera Link parameters.
4. **Misc. Tab** for configuration of output options: Image Display, Data Save and Log.

The following subsections describe in detail each of the Tabs and its respective parameters/controls.

4.7.1. Acquisition Tab

The **Acquisition Tab** defines the ProcFG's operation mode.

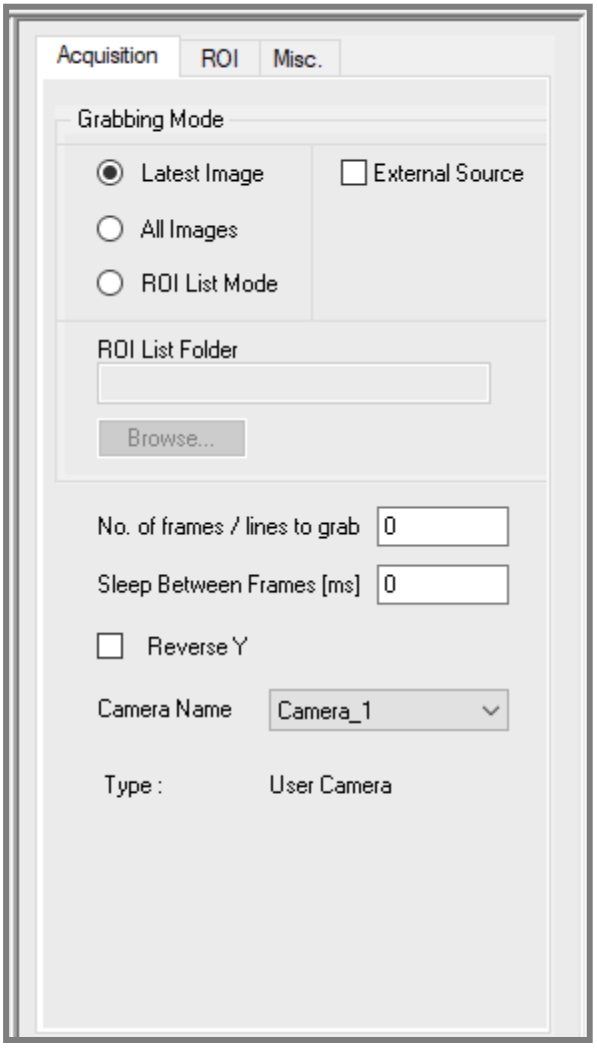


Figure 13: Acquisition Tab

The ProcFG application can capture images in three modes:

1. Selecting **Latest Image** sets the system in an acquisition mode that transfers Frame(s)/ROI(s) to the host computer only for the latest image stored in the Proc board's memory.

2. Selecting **All Images** sets the system in an acquisition mode that transfers frame(s)/ROI(s) to the host computer in FIFO order for all images stored in the Proc board's memory. In case of memory overrun, missed images are skipped.

3. Selecting **ROI List Mode** sets the system in an acquisition mode that transfers ROIs to the host computer according to the sequence defined in and ROI List file (.ini file). This mode is for simulating dynamic ROI

request based on the ProcFG API. The **ROI List Folder** box enables to browse to desired .ini file.

For further information regarding ROIs and ROI list, refer to ***sections 4.7.2-4.7.3***.

Selecting the **External Source** option will indicate that acquisition start/stop will be controlled from an external trigger source. For additional information on external triggering refer to section 7.7.3.

The **No. of frames / lines to grab** box defines the number of frames, in case of an Area Scan camera, or the number of lines, in case of a Line camera, that will be transferred to the ProcFG application for displaying and/or saving. For unlimited frames/lines, set the **No. of frames / lines to grab** to '0'.

The **Sleep Between Frames [ms]** box defines how much time to pause (in milliseconds) after grabbing of each successive image or ROI. In this way, it is possible to inspect the image progression in "slow motion", but in case of memory overrun, missed images are skipped.

Selecting **Reverse Y** reverses the image around the x-axis such that what was the upper row is now the lower row and vice versa, etc.

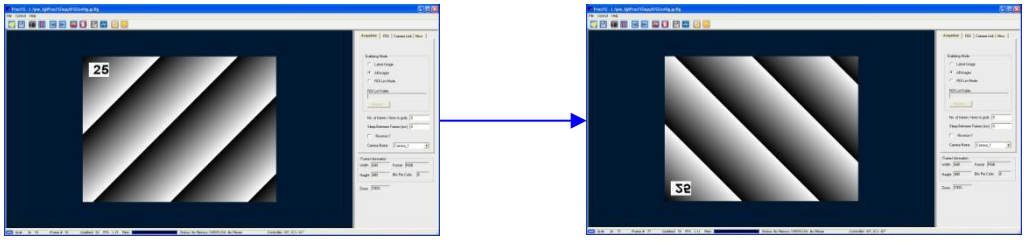


Figure 14: Reverse Y Example

In case of capturing images from several cameras, several instances of the ProcFG application have to be opened, one per camera. The **Camera Name** box selects a camera per specific ProcFG instance. The first instance operates as Master, and the other instances operate as Slaves. For further information on capturing images from multiple cameras, please refer to ***section 4.9***.

The **Frame Information** area displays the image frame information:

1. **Width** and **Height** are the acquired images size in pixels. **Note:** for Camera Link cameras, the frame width (in pixels) must divisible by 32.
2. **Format** can be Raw, Mono, Bayer, RGB, RGBA, YUV, YCbCr601 or YCbCr709. In case there is a Sub-Format, it is appended to the Format, yielding Format/Sub-Format. In case of Bayer, the sub-formats are: GR, RG, GB and BG. In case of YUV, YCbCr601 and YCbCr709, the sub-formats are: 411, 422 and 444. **Note:** in the current ProcFG version, Planar format is not supported. For Planar support, contact Gidel.
3. **Bits Per Color** is the number of bits per color.

4.7.2. ROI Tab

An important aspect of a Frame Grabber is the ability to discern ROIs in real-time, and to request intelligently and dynamically progressive ROIs from acquired images. The Proc board leverages the FPGA's high performance to process acquired images and to transfer efficiently to the Host computer only bare essential ROIs.

ROIs can be used to join adjacent areas by using overlapping ROIs without losing valuable neighboring pixel information.

The ProcFG provides flexibility to extract image data in a variety of ways:

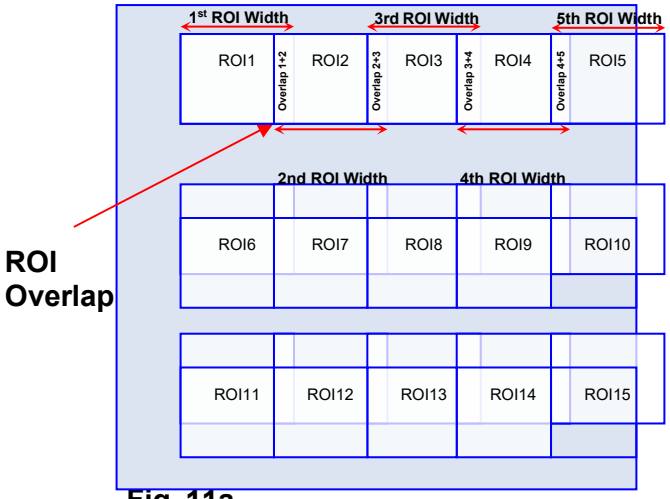
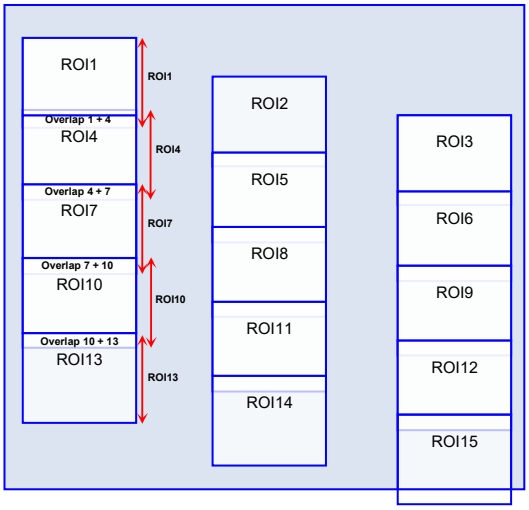
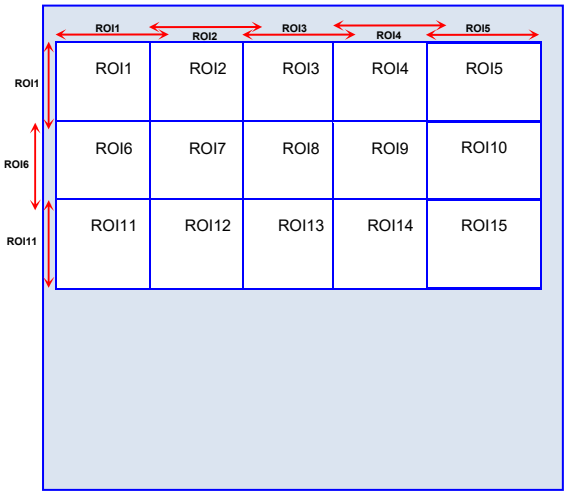


Fig. 11a

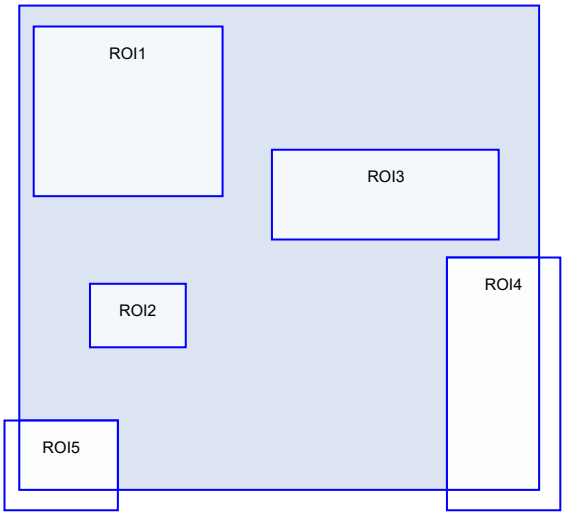
Rows of Overlapping ROIs



Columns of Overlapping ROIs



Matrix of Overlapping ROIs



ROIs with Random Sizes and Locations

Within the framework of the ProcFG application, the act of extracting relevant data specifically refers to the operation of transferring ROI data from the Proc

Board's memory to the Host Computer's memory; subsequently, data can be displayed and/or saved in the computer's hard disk.

For clarification purposes, in case of an Area Scan camera, the extracted image data is referred to as the image frame's **ROI**. In case of a Line camera, the input image data received from the camera is a continuous strip of lines. The image data extracted from the strip is referred to as a **frame**.

From the perspective of an Area Scan camera, the ProcFG can extract multiple ROIs per each acquired frame. From the perspective of a Line camera, the ProcFG can extract multiple frames from the "infinite" image strip. In ROI List mode, for Line cameras it is also possible to extract ROIs from the frames.

The ProcFG has three ROI operation modes:

1. **Basic Mode**: in this mode, only a single ROI/frame is extracted per each acquired image frame/sub-strip. The basic settings include the ROI/frame's size and the ROI/frame's location offsets.

2. **Advanced Mode**: in this mode, multiple ROIs/frames are extracted per each acquired image frame/sub-strip. This mode is especially valuable for creating overlapping Frames for Line camera images. Moreover, Advanced mode allows capturing frames according to reference coordinates and in this manner aligning captured frames with reference images.

3. **ROI List Mode**: in this mode, ROIs can be transferred to the Host computer in any random timestamp sequence. If a Line Camera is used, then the acquired strip is broken into a virtual grid/matrix of frames defined by the Advanced Mode, from which ROIs can be extracted according to the ROI List file. The ROI List mode is particularly intended for simulating and debugging Dynamic ROI requests based on ProcFG API.

4.7.2.1. ROI Tab in Basic Mode

Basic mode is limited to setting the size and location offsets of a single ROI/frame per acquired image. Advanced mode, detailed in the next section, allows defining the location of multiple ROIs/frames per image.

In Basic mode, the ROI Tab is as follows:

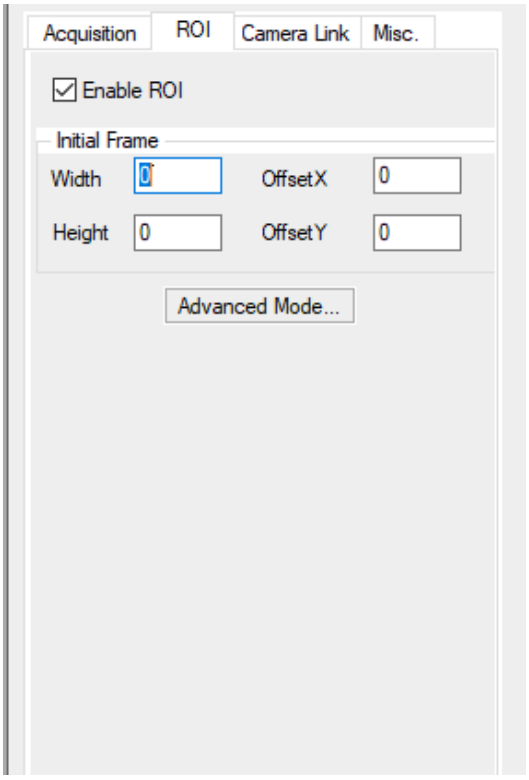


Figure 15: ROI Tab in Basic Mode

Clearing the **Enable ROI,** sets the system in a full-image capture mode; in other words, the entire image data will be transferred to the Host rather than a specific Region Of Interest. To enable selective ROI capture, select the **Enable ROI.**

The ROI setup is slightly different for Line and Area Scan Camera as explained in the following sections:

Line Camera ROI

The image data that arrives from a Line camera is an ongoing strip composed of image lines. Basic mode configuration enables to break the “infinite” strip into frames of a specified number of lines.

We use the **Width** and **Height** boxes to define the frame size. **Width** specifies the frame width in pixels and can be any fraction of the camera line width. For Camera Link, the Width must be divisible by 32. The **Height** specifies the frame’s number of lines. **OffsetX** and **OffsetY** define the starting location of the ROI as illustrated in the following figure:

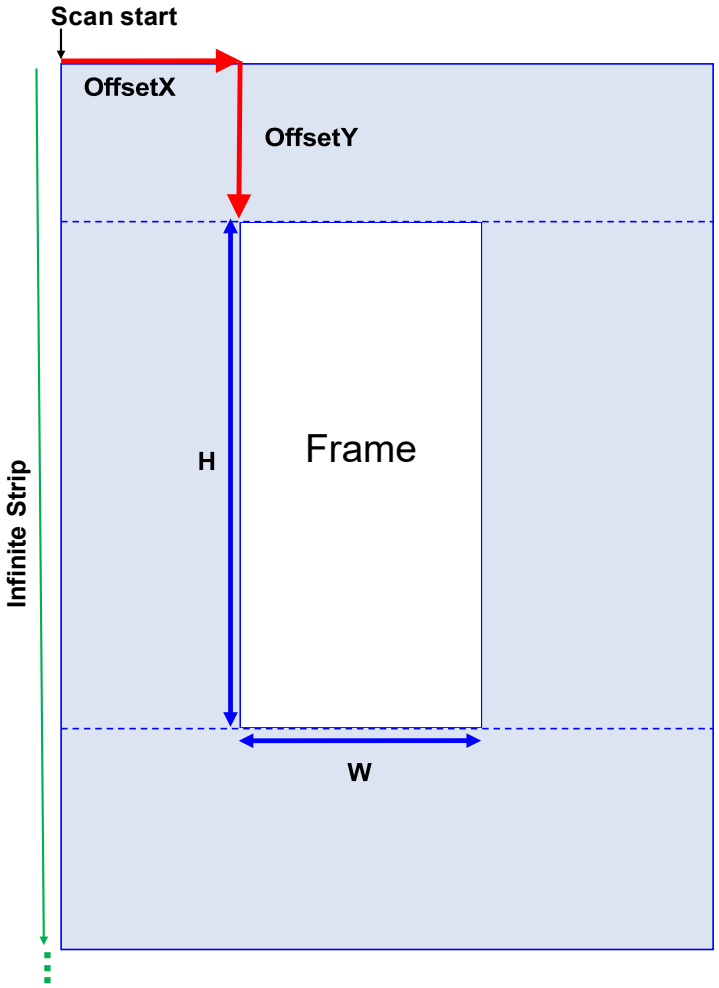
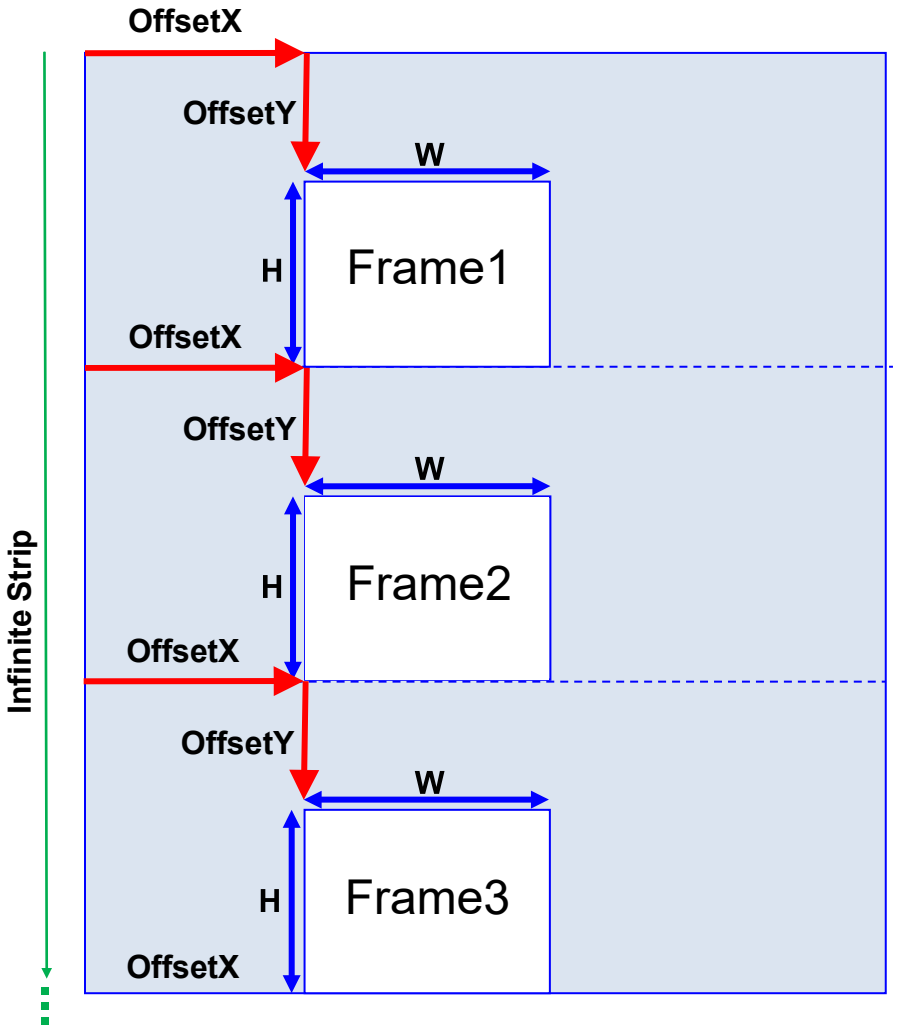


Figure 16: Basic Mode - Line Camera Frame (ROI)

NOTE:	If **Height** is set to ‘0’, then the frame height will be set to a default value of 1024 lines.

In Basic mode, for a Line camera once **Height** is defined, each successive frame will have the same height, i.e., same number of lines.

Figure 17: Basic ROI Mode - Successive Line Camera Frames



Area Camera ROI

For Area Scan camera, Basic mode defines a single ROI's location and size. Use **Width** and **Height** to define the ROI's size and **OffSetX** and **OffsetY** to define the ROI's initial location.

The following figure is an example of an acquired area camera image frame whose size is 640×480 pixels from which an ROI of 160×120 with an OffsetX=320 and an OffsetY =240 is extracted:

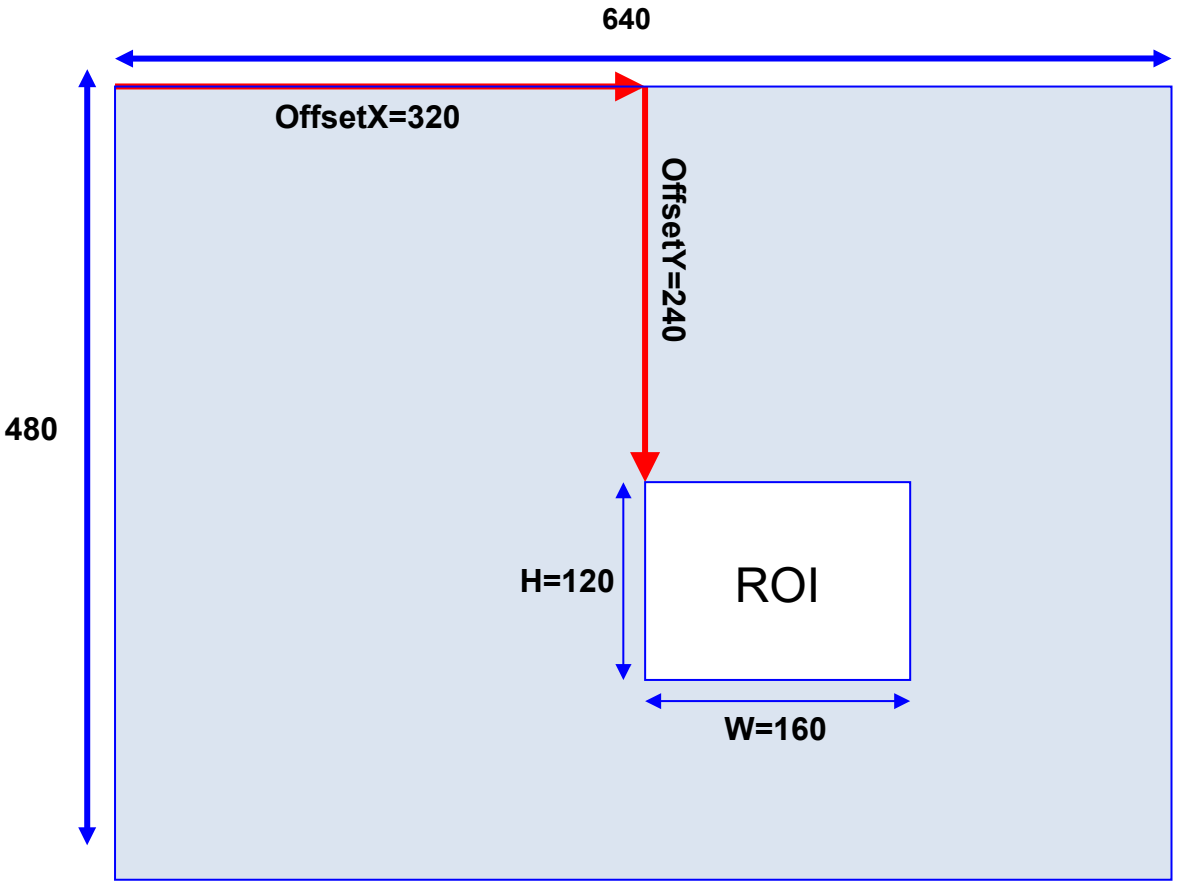


Figure 18: Basic Mode - Area Scan Camera ROI

The ROI shown in Figure 18 will be extracted from each successive image frame stored in the Proc board's memory and transferred to the Host's memory for displaying and/or saving.

4.7.2.2. ROI Tab in Advanced Mode

The ROI Advanced mode allows defining the location of multiple ROIs per image frame for an Area Scan Camera or multiple frames per scanned sub-strip for a Line Camera. Frames and ROIs can be overlapped enabling localized filtering on a single frame/ROI that includes neighboring pixel data.

In addition, Advanced mode enables capturing ROIs/Frames that are aligned to a reference image even when there is image shifting, scaling or rotation.

Clicking the **Advanced** button will expand the ROI tab as follows:

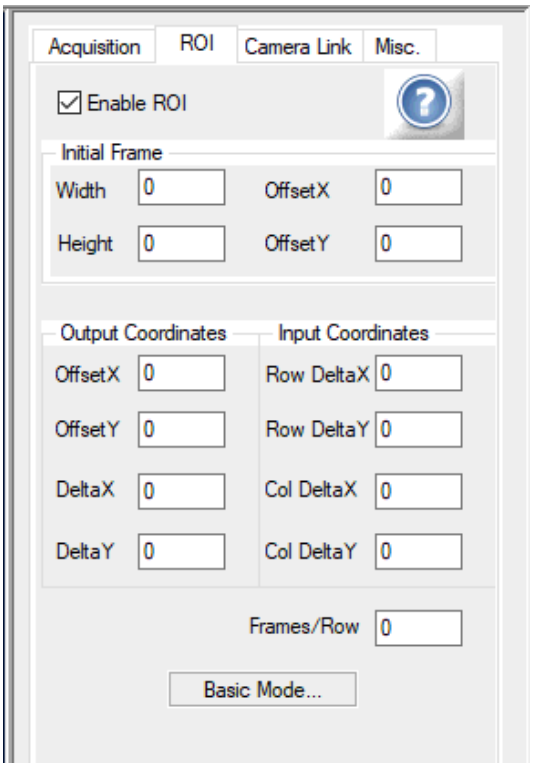


Figure 19: ROI Tab in Advanced Mode

4.7.2.2.1. Input Coordinates

To define multiple ROIs/frames per acquired image frame/sub-strip, in addition to the **Height**/**Width** and **OffsetX**/**OffsetY** parameters, use the **Row DeltaX/Row DeltaY** and **Col DeltaX/Col DeltaY** as follows:

1. **Row DeltaX** = the starting X coordinate of the following ROI/frame relative to the previous ROI/frame's starting X coordinate, where both (adjacent) ROIs/frames are within the same row.

2. **Row DeltaY** = the stating Y coordinate of the following ROI/frame relative to the previous ROI/frame's starting Y coordinate, where both (adjacent) ROIs/frames are within the same row.

Once the Row DeltaX/Row DeltaY parameters have been defined, a row of ROIs/frames will be generated and transferred to the Host computer as illustrated in the following figure:

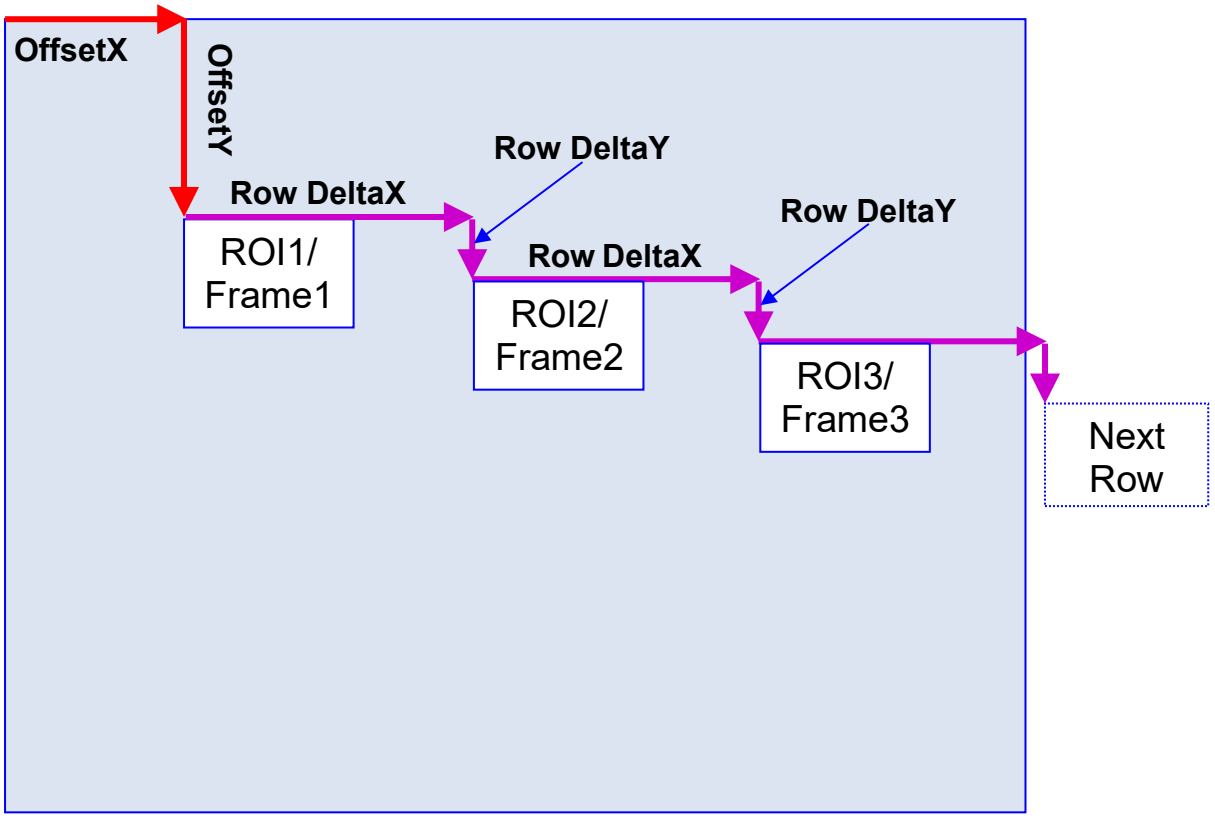


Figure 20: Multiple ROIs/Frames in a Single Row

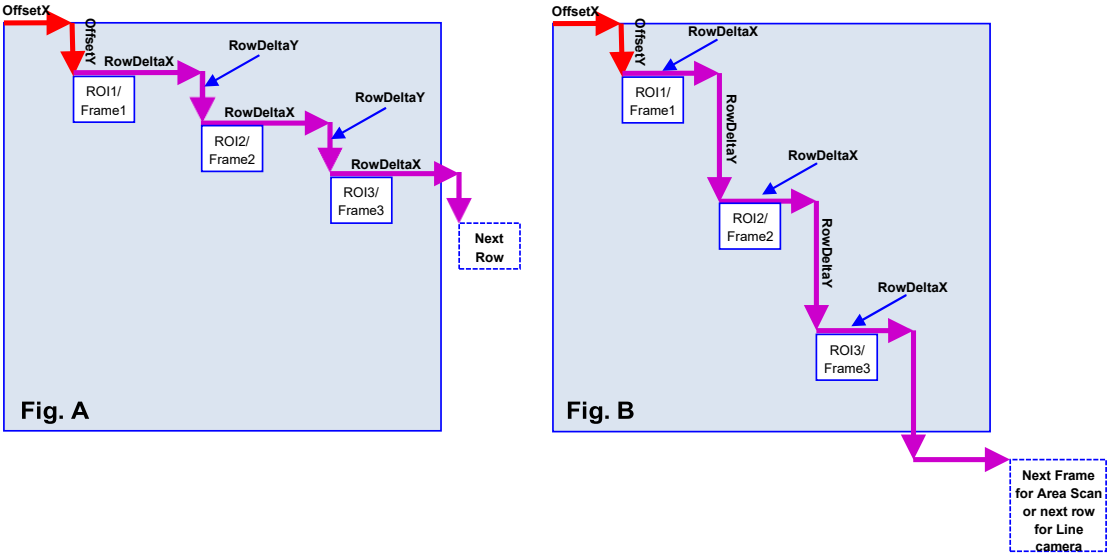
Such a sequence of ROIs/frames that progress according to the Row DeltaX and Row DeltaY parameters is referred to a **"Row of ROIs/frames"**.

One of the following ways will determine the number of ROIs generated per row:

1. The **Frames/Row** box value defines the number of ROIs or number of frames per row (see Figure 19).

2. If the **Frames/Row** box is set to **0**, then successive ROIs/frames will be grabbed until an ROI/frame is detected to extend fully beyond the area of the acquired image. This can happen in two situations as illustrated in the following figure:

Figure 21: End of ROI/Frame Row Example



For both Line and Area Scan cameras, if an ROI image extends beyond the acquired images right boarder as shown in ***Figure 21A***, this indicates that the end of an ROI/Frame row has been reached and a new ROI/Frame row needs to begin. To determine the starting location of the next ROI/Frame row, we use the **Col DeltaX** and **Col DeltaY** fields as follows:

1. **Col DeltaX** = the ROI/frame row's starting X coordinate relative to the previous ROI/Frame row's starting X coordinate.

2. **Col DeltaY** = the ROI/frame row's starting Y coordinate relative to the previous ROI/Frame row's starting Y coordinate.

Once the Col DeltaX/Col DeltaY parameters have been defined, the next ROI/frame row sequence will be generated as illustrated in the following figure:

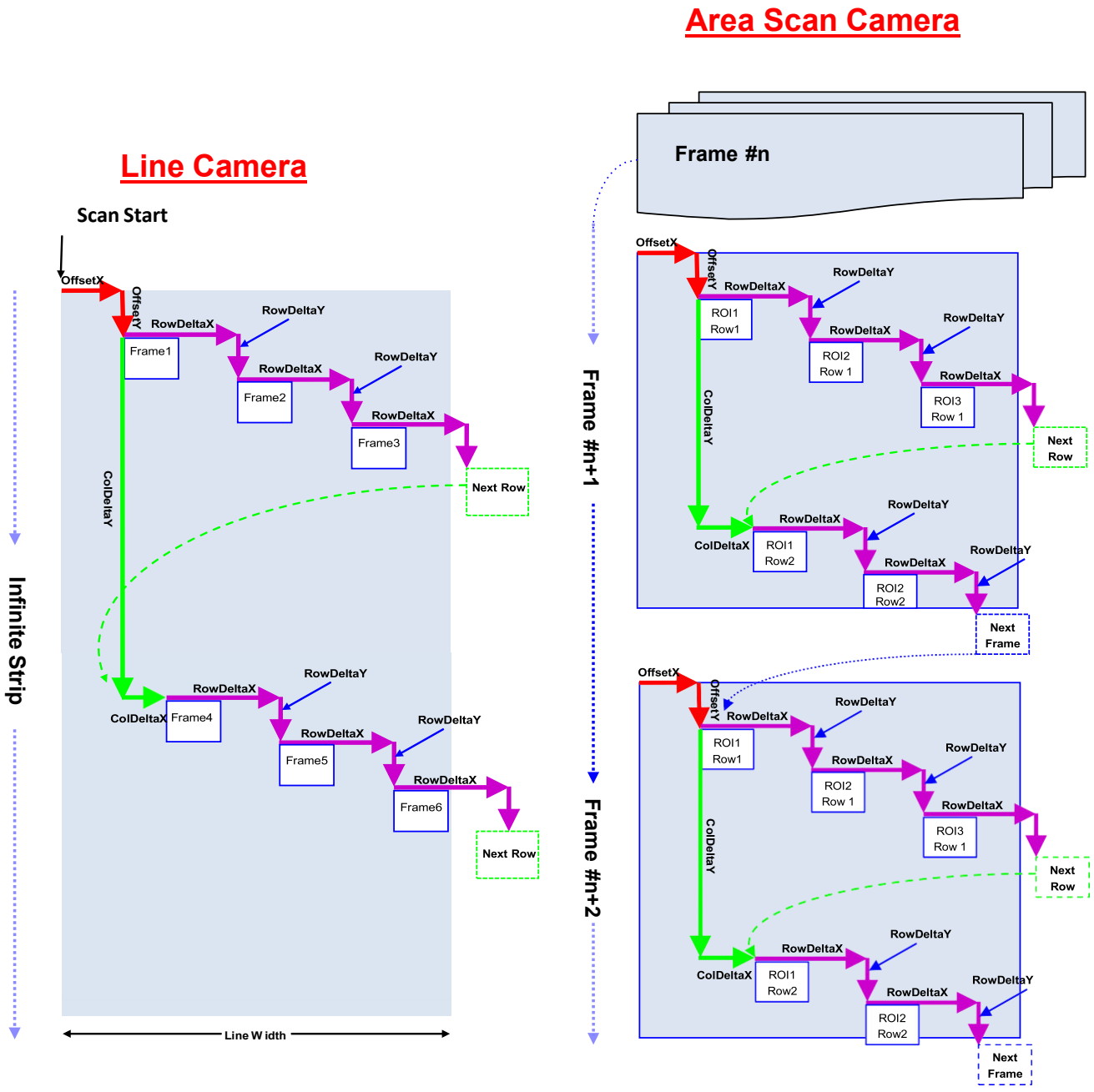


Figure 22: Successive ROI Rows for Line and Area Scan Cameras

For Area Scan Camera, successive ROI rows will be generated until an ROI has been located beyond the lower boarder of the acquired image as shown in ***Figure 21A***; at this point, the ProcFG will continue to the next image stored in the Proc Board's memory and begin transferring to the host computer a new sequence of ROIs/Frames.

For Line Camera, at the end of each row of frames, the next row of frames will begin according the **Col DeltaX/Y** parameters.

4.7.2.2.2. Output Coordinates

Images captured by the camera often represent only a partial image from a larger view. Therefore, the images transferred to the frame grabber represent only relative coordinates in relation to the first frame whose first pixel represent the (0,0) origin. As a result, there is a need to translate the relative coordinates of captured images back into what is referred to as Output or World coordinates.

The ProcFG allows determining the World Coordinates per each captured ROI/frame based on predefined user parameters as follows:

1. **OffsetX** = the x-coordinate offset for the first ROI/frame.
2. **OffsetY**= the y-coordinate offset for the first ROI/frame.
3. **DeltaX** = the x-coordinate offset of the next ROI/frame relative to the x-coordinate of the current ROI/frame.
4. **DeltaY** = the y-coordinate offset of the next ROI/frame relative to the y-coordinate of the current ROI/frame.

In a matrix of **m** by **n** ROIs/frames, each individual ROI/frame will have the following starting coordinates:

Starting coordinates (x, y)= (OffsetX + m×DeltaX, OffsetY + n×DeltaY)

The ProcFG API methods enable to read the World coordinates and process each ROI/frame accordingly.

4.7.3. ROI List Mode

ROI List mode offers flexibility to extract ROIs/Frames in any desired sequence from image frames. As in the normal acquisition modes (All Images or Latest Image), the image frames are continuously acquired and stored in the Proc board's memory. Based on an ROI List file (**.ini**), the ProcFG will extract ROIs in a specified sequence.

ROI List mode is specifically intended for simulating **Dynamic ROI** mode. Dynamic ROI enables to request ROIs in the same manner as the ROI list requests, except that the requests are executed on-the-fly using the ProcFG API.

ROI list operates slightly different for Area Scan and Line cameras. The ProcFG receives information from the Camera Link camera indicating whether the camera is Area Scan or Line and then operates accordingly.

For Area Scan cameras, the ROIs are extracted exactly as prescribed in the ROI List ini file. In other words, all the data defined in the ROI tab, including the Advanced parameters, are ignored. Each frame stored in the Proc board's memory is numbered in a sequential order. These numbers later serve as references used to extract ROIs in any random sequence from frames stored in the Proc board's memory.

For Line Cameras, the frame sizes and locations are defined according to the ROI Tab's Advanced mode parameters. These frames form a two-dimensional matrix where each frame is assigned matrix indices (x, y). The ROI List uses these indices to specify which ROIs to extract from the matrix. In this way, ROI List mode enables to extract ROIs from Line camera strips in any random time stamp order for any image data stored in the Proc board's memory.

If a requested ROI is not yet stored in the Proc board's memory, the ProcFG waits till it is available. If the requested ROI has been overwritten by newer ROIs/Frames, an indication that the requested ROI is not available is returned as part of the image data together with an empty image, and the returned (empty) image is neither displayed nor saved.

The following is an example of an ROI List:

```
[Global]                    // Global Section

NumIterations = 0           //Number of Iterations. 0 means infinite
DeltaFrameNumX = 0          //Delta Frame Number along X for Line Camera
DeltaFrameNumY = 60         //Delta Frame Number along Y for Line Camera. Delta Frame Number for Area Scan Camera
DeltaRoiXoffset = 0         //Delta Roi X Offset
DeltaRoiYoffset = 0         //Delta Roi Y Offset
DeltaWidth = 0              //Delta Roi Width
DeltaHeight = 0             //Delta Roi Height
InitialSleep = 1000         //Initial Sleep [ms] before sending (queuing) the first ROI
//First ROI with LastRoi=1. 0 means no such ROI (but ROIs based on the next parameter can be used)
LastRoiEvery = 0            //Last ROI every n ROIs. 0 means only the First ROI (according to the previous parameter)

[ROIs]                      //ROI Section

FrameNumX = 0               //Line Camera: Frame number along X. Area Scan Camera: Not used
FrameNumY = 0               //Line Camera: Frame number along Y. Area Scan Camera: Frame Number
RoiXoffset = 0              //Offset of the ROI from the Frame origin along X
RoiYoffset = 0              //Offset of the ROI from the Frame origin along Y
RoiWidth = 640              //ROI Width
RoiHeight = 480             //ROI Height
Sleep = 0                   //Sleep time [ms] after the ROI is sent (queued)

FrameNumX = 0
FrameNumY = 11
RoiXoffset = 0
RoiYoffset = 0
RoiWidth = 640
RoiHeight = 480
Sleep = 0

Etc., etc.
```

Figure 23: ROI List Example

The ROI section shown in Figure 23 uses the following parameters:

Table 4: ROI List Parameters

Parameter	Description
FrameNumX	For Line cameras, in an **X** by **Y** matrix of frames, FrameNumX represents the **X** index. For Area Scan Camera this parameter is not used.
FrameNumY	For Line cameras, in an **X** by **Y** matrix of frames, FrameNumY represents the **Y** index. For Area Scan Camera this parameter represents the number of a specific image frame.
RoiXoffset	For Area Camera, the RoiXoffset defines the ROI's x offset with respect to the origin of the frame stored in the Proc board's memory. For Line Scan Camera, the RoiXoffset defines the ROI's x offset with respect to the origin of the ROI as defined by the matrix indices.
RoiYoffset	For Area Camera, the RoiYoffset defines the ROI's y offset with respect to the origin of the frame stored in the Proc board's memory. For Line Scan Camera, the RoiYoffset defines the ROI's y offset with respect to the origin of the ROI as defined by the matrix indices.
RoiWidth	The width of the ROI
RoiHeight	The height of the ROI
Sleep	The pause time (in milliseconds) between the time the current ROI is requested from the ProcFG API and the time the next ROI is requested.

In addition, the ROI List enables to perform loop iterations using global parameters detailed in Table 5.

Note that the Global parameters of the form Delta... (e.g., DeltaWidth) are changes (deltas) added to the corresponding parameter values in each ROI in the ROIs section. Each time through the loop, the Delta... parameters are added to the previous parameter values.

Table 5 : ROI List Global Parameters

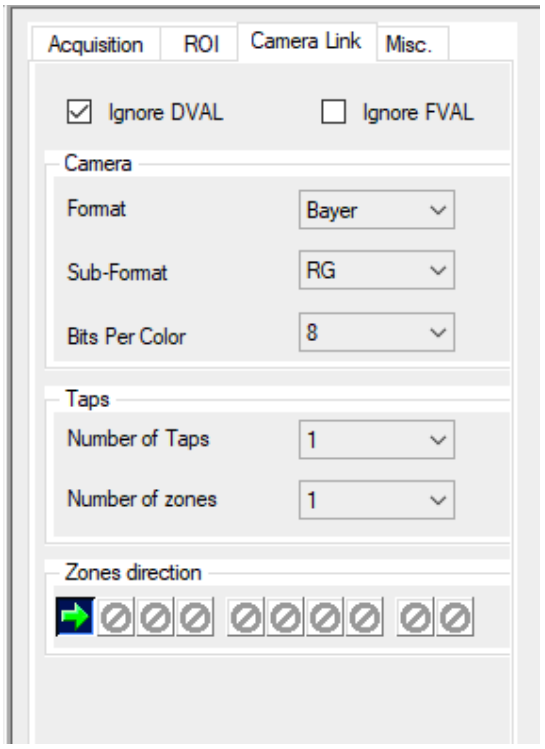
Global Parameter	Description
NumIterations	The Number of Iterations to execute. '0' means infinite.
DeltaFrameNumX	For Line Camera only: specifies within the frames matrix (X by Y) the current iteration's X index relative to the previous iteration's X index.
DeltaFrameNumY	For Line Camera, specifies within the frames matrix (X by Y) the current iteration's Y index relative to the previous iteration's Y index. For Area Scan camera, specifies the current iteration's frame number relative to the previous iteration's frame number.
DeltaRoiXoffset	The current iteration's ROI X offset relative to the previous iteration's ROI X offset.
DeltaRoiYoffset	The current iteration's ROI Y offset relative to the previous iteration's ROI Y offset.
DeltaWidth	The current iteration's ROI width relative to the previous iteration's ROI width.
DeltaHeight	The current iteration's ROI height relative to the previous iteration's ROI height.
InitialSleep	Initial pause [ms] before sending the first ROI request to the ProcFG API.
FirstLastRoi(*)	Indicates which ROI in the ROI list is tagged as the first "Last ROI". 0 means there is no first "Last ROI".
LastRoiEvery(*)	Indicates the frequency of "Last ROI" tagging in the ROI list. For example, LastROIEvery=100, means that every 100th ROI will be tagged "Last ROI". If the FirstLastRoi=5, the second "Last ROI" will be the 105th ROI. 0 means that only the first "Last ROI" is tagged.

(*) It is possible to break down the ROIs into subgroups for display/save. An ROI that has been tagged "Last ROI" indicates that this is the last ROI to display/save in this group. In turn, the ProcFG stops the transfer of ROIs from the Proc board to the Host's computer's memory. Once all ROIs have been displayed, the Proc board resumes transferring ROIs to the Host computer until the next "Last ROI" is encountered.

4.7.4. Camera Link Tab

The **Camera Link Tab** is displayed only in case of Camera Link. It configures the Camera Link parameters according to the transmitting source camera's parameters.

Figure 24: Camera Link Tab



Some Cameras use the DVAL signal and others do not. Selecting the **Ignore DVAL** check box will ignore the DVAL signal, while clearing the **Ignore DVAL** check box will incorporate the DVAL signal. If you are using a Line camera that does not use FVAL, select **Ignore FVAL**. For Area cameras, the **Ignore FVAL** must be cleared.

The **Camera** area defines the transmitting camera's configuration. The **Format** can be one of the following image formats:

1. Mono
2. Bayer
3. RGB
4. RGBA

If the image format has further sub-format options, in the **Sub-Format** box select an image sub-format. For example, for the Bayer images, you may select **GR**, **RG**, **GB** or **BG**.

Bits Per Color can be 8, 10, 12, 14 or 16 for Mono, Bayer and RGBA formats, and 8, 10, 12 for RGB format.

The source camera's total number of taps is defined by the following formula:

Total number of Taps = Number of parallel pixels × Number of zones

In the **Taps** area, configure the taps as follows:

1. **Number of parallel pixels**: defines the number of parallel pixels transmitted per zone.

2. **Number of zones**: defines the number of zones per acquired image.

The ProcFG supports all Camera Link configurations (Base, Medium, Full and 80-bit single zone), and according to the selected **Bits Per Pixel** format, automatically lists the available **number of zones** and **parallel pixels**. For 80-bit ("Deca" / "Full Plus") multi-zone support, contact Gidel.

Finally, the **Zone direction** area defines the source camera's pixel ordering direction within each zone. Each button in the **Zone direction** area signifies a zone and the arrow indicates the pixel ordering direction. The number of enabled buttons will be according to the number of zones defined. Clicking a button toggles the arrow's direction.

NOTE:	In case of Camera Link, Gidel provides a number of RBF files to support different configurations as follows: 1. RBF per specific type of Proc board. 2. RBF for multi-zones supporting all configurations except for 80-bit configurations. 3. RBF for single-zone supporting only single-zone configurations, including 80-bit single zone configurations. For further information on the RBFs supplied, please refer to ***section 4.7***.

4.7.5. Misc. Tab

The ProcFG's acquired image data can be displayed and/or saved. In addition, it is possible to generate a Log file that provides time-stamped information regarding the acquisition process useful for developing and debugging.

Note that the Save Folder has changed from **Logs** to **..\..\FgImages**.

The **Misc Tab** configures the ProcFG's displaying, saving and logging settings.

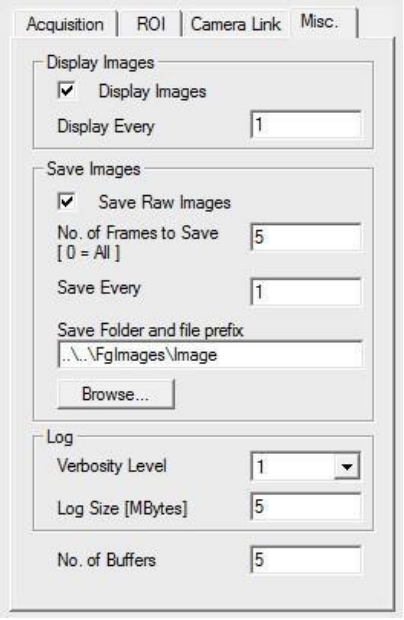


Figure 25: Misc. Tab

In the **Display Images** area shown in Figure 25, selecting **Display Images** will display the captured images according to the **Display Every** box, where 0 or 1 mean that every image will be displayed, 2 means that every second frame will be displayed and so forth. Clearing the **Display Images** check box means that the captured images will not be displayed.

In the **Save Images** area, selecting the **Save Raw Images** will save the image files in the computer's disk according to the number of frames requested as defined in the **No. of Frames to Save**. The saved files contain Raw images as obtained from the FPGA. In order to be displayed, they have to be converted to the display format. This is done by the ProcFG GUI. The frequency of saved files will be determined by the **Save Every** field, where 0 or 1 mean save all images, 2 means save every second image, and so forth. The **Save Folder and file prefix** specifies where to save the files as well as the prefix given to the group of image files that will be captured. Clicking **Browse...** opens the following dialog box:

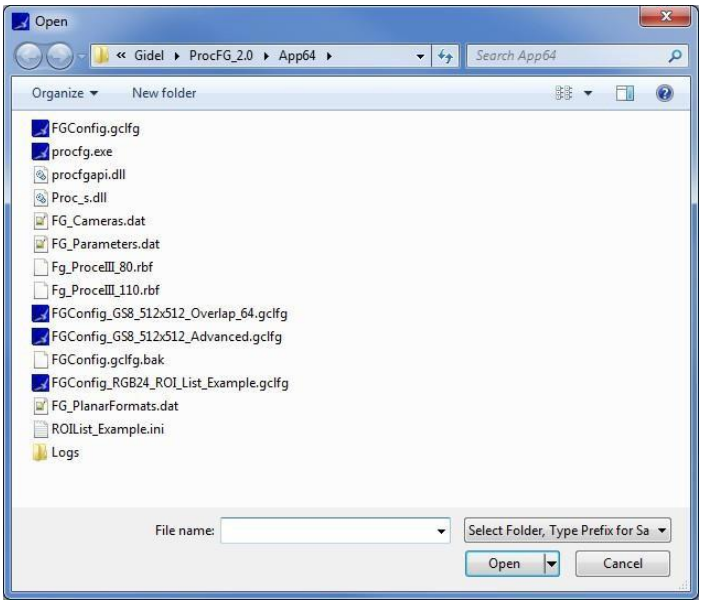


Figure 26: Image Save Dialog Box

After browsing to a desired file, in the **File name** box, enter a desired prefix and click the **Open** button to apply the settings.

The saved file names will be according to the following syntax:

[Prefix]_[Sequential Frame #]_[Absolute Frame #]_[Y]_[X].raw
(for example: ***Image_32_32_4_8.raw***)

Where,

Prefix = Specified files' prefix.

Sequential Frame # = a sequentially incremented number of the ROI/frame. For example, if the **Save Every** is set to 20, the sequence number will be 0, 20, 40, etc. meaning that only every 20th frame received by the Host computer is saved.

Absolute Frame # = the absolute number of the frame.

[X] and **[Y]** = the ROI/frame's indices in a matrix of multiple ROIs/Frames.

For example, in case of a 2x2 matrix, the file names may be:

Image_0_0_0_0.raw, Image_1_0_0_1.raw, Image_2_0_1_0.raw, Image_3_0_1_1.raw, Image_4_1_0_0.raw, Image_5_1_0_1.raw, Image_6_1_1_0.raw, Image_7_1_1_0.raw

Note that in case there is only one ROI per frame, the file names are simpler (not containing X and Y), according to the following syntax:

[Prefix]_[Sequential Frame #]_[Absolute Frame #].raw

For example:

Image_0_0.raw, Image_1_1.raw, Image_2_2.raw, Image_3_5.raw, Image_4_6.raw

In the above example, two images were skipped.

In case the user clicks Save Image (after clicking Next frame or Previous frame), the file name does not include the **Sequential Frame #**.

In case that more than one camera is used, the camera name is appended to the name of the folder where images should be saved.

For example, if the **Misc.** tab contains "Logs\Image", images of the first camera will be saved in "Logs_CL1" and images of the second camera will be saved in "Logs_CL2".

Clearing the **Save Raw Images** means that captured images will not be saved.

NOTE:	The line width in the saved files may be bigger than the expected line width. For explanation on how to calculate actual line width, please refer to ***section 4.7.6***.

The **No. of Buffers** specifies how many frame/ROI buffers to allocate in the Host computer's RAM memory. Each buffer holds a single frame's or ROI's data. For optimal system performance, it is recommended to work with 3 to 4 buffers. Larger

amount of buffers may strain the RAM resources. As a rule, you should not use more than 50% of the board's memory allocated for the ProcFG. For example, if you are using a 2 GB SODIMM, you should use a maximum of 1 GB.

The ProcFG can generate log files that can be useful for debugging and verification. Generated log files are saved in the **Logs** folder located in the **application** folder.

In the **Log** area, if **Verbosity Level** is set to **OFF,** no log will be generated. **Verbosity Level** of 1 means that a log will be generated, containing minimal information. Verbosity Level 2 generates a more detailed log reserved for Gidel technical support purposes. The **Log Size...** box specifies the size, in MB, of the generated log file.

The following is an example of a log file:

```
INF   16:55:01.112    5460   Log created.
INF   16:55:33.565    5984   Grab Started...
INF   16:55:33.565    5984   Grabber: Memory Size = 512 MBytes
INF   16:55:33.565    5984   CameraLink Reorder IP parameters:
INF   16:55:33.565    5984           Input <24> bits per pixel
INF   16:55:33.565    5984           Number of parallel pixels <1>
INF   16:55:33.565    5984           Number of zones <1>
INF   16:55:33.565    5984           Zones Direction <0>
INF   16:55:33.565    5984           Format <4>
INF   16:55:33.565    5984           Sub-Format <0>
INF   16:55:33.565    5984           Acquisition Frame Count Not set (Continuous Mode)
INF   16:55:46.674    5984   CameraLink IP parameters:
INF   16:55:46.674    5984           Format <RGB>
INF   16:55:46.674    5984           Sub-Format <0>
INF   16:55:46.674    5984           Bits Per Color <8>
INF   16:55:46.674    5984   ======================GrabParameters======================
INF   16:55:46.674    5984   Grab FPGA parameters: XPadding=0, BitsPerColor=8, ImageType=1
INF   16:55:46.674    5984   Grab FPGA parameters: Width=640, Height=480
INF   16:55:46.674    5984   Grab Parameters: Width(Bytes)=2560, BytesPerPixel=4.000
INF   16:55:46.674    5984   Grab Parameters:LineCamera=0,StripWidth=640,StripHeight=480
INF   16:55:46.674    5984   Grab Parameters: StripWidthInBytes=2560, StripRowXPadding=0
INF   16:55:46.674    5984   Grab Parameters: CameraNumber=1, RoiListMode=False
INF   16:55:46.674    5984   Grab Parameters: CameraNumber=1, RoiListMode=False
INF   16:55:46.674    5984    ..........................................................
INF   16:55:46.674    5984   Frames Count (after DMA): In<1> Frame #<0> Grabbed<1>
INF   16:55:46.674    5984   Memory use< 0.24%>
INF   16:55:46.674    5984   Frame data Start<1>:  00 00 00 00 01 02 04 00 02 04 08, etc.
INF   16:55:46.674    5984   Frame data End  <1>:  4F BF 9F 00 50 C1 A3 00 51 C3 A7, etc.
INF   16:55:46.674    5984    ..........................................................
INF   16:55:46.674    5984   Frames Count (after DMA): In<1> Frame #<1> Grabbed<2>
INF   16:55:46.674    5984   Memory use< 0.27%>
INF   16:55:46.674    5984   Frame data Start<2>:  00 00 00 00 01 02 04 00 02 04 08, etc.
INF   16:55:46.674    5984   Frame data End  <2>:  4F BF 9F 00 50 C1 A3 00 51 C3 A7, etc.
INF   16:55:46.674    5984   ..........................................................
INF   16:55:46.690    4956   WaitForBuffer Thread. Number of Displayed Images = 1
INF   16:55:46.690    5984   Frames Count (after DMA): In<2> Frame #<2> Grabbed<3>
INF   16:55:46.690    5984   Memory use< 0.27%>
INF   16:55:46.690    5984   Frame data Start<3>:  00 00 00 00 01 02 04 00 02 04 08, etc.
INF   16:55:46.690    5984   Frame data End  <3>:  4F BF 9F 00 50 C1 A3 00 51 C3 A7, etc.
INF   16:55:46.690    5984   ..........................................................
```

Figure 27: Log File Example

Table 6 details the log file's parameters.

Table 6: Log Parameters

Log Parameter	Description
Log created.	Indication that the Log file has been created.
Grab Started...	Indication that acquisition has started (whenever the Grab button is clicked).
Memory Size	The size of the on-board memory (in Mbytes) used for image acquisition.
CameraLink Reorder IP parameters:	
Input <X> bits per pixel	The number of bits per pixel: Grayscale 8/10/12/14/16 bpp or RGB 24/30/36 bpp.
Number of parallel pixels <X>	The number of parallel pixels per zone sent on each clock.
Number of zones <X>	The number of image zones. Note: For 80-bit ("Deca" / "Full Plus") multi-zone support, contact Gidel.
Zones Direction <X>	Zone fill direction: one bit is allocated to each zone. Bit 0 for the leftmost zone, bit 1 for the 2nd zone from the left, etc. **0** = left to right, **1** = right to left.
Acquisition Frame Count	**Not set (Continuous Mode):** image acquisition is continuous not limited to a number of frames. **Acquisition Frame Count <X>:** acquisition is limited to a number of X frames.
Acquisition parameters obtained from the FPGA:	
XPadding	The number of bytes added between lines.
BitsPerColor	The number of bits per color.
ImageType	Image type: **0** = grayscale **1** = color image (RGB)
Width	The line width in pixels.
Height	The number of lines per frame. For Line camera, this parameter is 1.
Parameters calculated from the Acquisition parameters: (some of the parameters are identical to previous parameters)	
Width(Bytes)	The line width in bytes.
BytesPerPixel	The number of bytes per pixel: **GS8** = 1 **GS10/12/14/16** = 2 **RGB24** = 4 **RGB30/36** = 8

LineCamera	Camera type: **Area scan camera** = 0 **LineCamera** = 1
StripWidth	The Strip width in pixels.
StripHeight	For Area scan camera: the Strip height in lines.
StripWidthInBytes	The Strip width in bytes.
StripRowXPadding	The number of bytes added between lines.
RoiListMode	ROI List mode: **False** = ROI list not used **True** = ROI list is used
Frames Count (after DMA):	
In<X>	Frames/lines transferred to the on-board memory.
Frame #<X>	Frame/ROI number (absolute number).
Grabbed<X>	Number of frames/ROIs transferred from the on-board memory to the host.
Memory use <X%>	Usage percent of the Proc board's memory.
Acquisition Data	
Frame data Start<X>	The first bytes of the first line of the ROI of frame <X>.
Frame data End <X>	The last bytes of the last line of the ROI of frame <X>.

4.7.6. Calculating Saved File's Line Width

The line width in the saved files of the captured images may be bigger than the expected line width.

In order to find out the line width in a saved file, do the following:

1. Right click on the .raw saved file and select **Properties**.
2. Find out the File Size in bytes (**FileSizeBytes).** Do not use the "Size on disk" value.
3. Find out the Number of Lines in the saved file (**NumberOfLines)**. This is the ROI Height, or the Frame Height.

Divide the File Size in Bytes by the number of lines in the saved file. This is the Line Width in Bytes, **LineWidthBytes**.

LineWidthBytes = FileSizeBytes / NumberOfLines.

To find out the Line Width in Pixels, **LineWidthPixels**, divide the Line Width in Bytes by the number of Bytes Per Pixel, **BytesPerPixel**:

LineWidthPixels = LineWidthBytes / BytesPerPixel.

In case of YUV, YCbCr601, YCbCr709 with sub-format of 411 or 422, multiply the obtained value by 2 to get the original width in pixels before the compression used in

those sub-formats.
The value of **BytesPerPixel** is:
1 8 bits per color with Raw, Mono, Bayer format.
2 10/12/14/16 bits per color with Raw, Mono, Bayer format.
4 8 bits per color with RGB, RGBA, YUV, YCbCr601, YCbCr709.
8 10/12/14/16 bits per color with RGB, RGBA, YUV, YCbCr601, YCbCr709 format.

4. In order to display the image, using a graphical application, use **LineWidthPixels** as the **Image Width**.

Notes:

1. In some cases, there is an Offset at the start of each line of the saved image. In **Basic ROI mode**, the Offset in Bytes, **OffsetXBytes**, is calculated from OffsetX in the ROI tab as:

OffsetXBytes = (**OffsetX * BytesPerPixel) & 0x1F**

In Advanced ROI mode, the equation is:
OffsetXBytes = ((OffsetX + nRow*ColDeltaX + nCol*RowDeltaX) * BytesPerPixel) & 0x1F,

where **ColDeltaX** and **RowDeltaX** are defined in the ROI tab, and nRow and nCol are the matrix indices of the frame, and counting starts from zero.

2. The Offset in Pixels, **OffsetXPixels**, can be calculated as:
OffsetXPixels = OffsetXBytes / BytesPerPixel.

3. In some cases, there are bytes padding each line, **XPadding**.
To find out XPadding in bytes, **XPaddingBytes**, subtract the Offset in Bytes from The Line Width in Bytes.
XPaddingBytes = LineWidthBytes – OffsetXBytes.

4. XPadding in Pixels, **XPaddingPixels**, can be calculated as:
XPaddingPixels = XPaddingBytes / BytesPerPixel.

4.8. The Status Bar

The Status Bar provides status on the ongoing acquisition process.

Figure 28: Status Bar

The status bar includes the following parameters:

Table 7: Status Bar Fields

	Parameter	Description
1.	**State**	Operation mode: **Ready:** ready for grabbing **Grab:** currently grabbing **Pause:** acquisition has paused
2.	**In**	Frames/lines transferred to the on-board memory.
3.	**Frame #**	Frame/ROI number (absolute number) currently being displayed.
4.	**Grabbed**	Number of frames/ROIs transferred from the on-board memory to the host.
5.	**FPS**	The ROI/frame transfer rate to Host computer, in frames per sec.
6.	**Mem**	Status of the Proc Board's memory usage.

(Table continued on next page…)

	Parameter	Description
7.	**Status**	Current operation status, for example: • Ready • Grab Started. Waiting … • Grabbing … • No Memory OVERFLOW. No misses • Memory OVERFLOW. Missed XX Frames • Grabbing Next Frame … • Last Frame is displayed. • Grabbing Previous Frame … • First Frame is displayed. • Grab Stopped by Abort command. • Grab Stopped by Abort/Reset/Acq Counter.
8.	**Controller, IC1**	Indicates the temperature of the Proc board's controller and the FPGA device.

Table continued from previous page.

4.9. Multi ProcFGs for Multi Camera Support

Several instances of ProcFG can be opened using the same Proc board and the same FPGA as follows:

1. Each ProcFG instance must grab images from a different camera.

2. Only one of the ProcFG instances may be in master mode. The others are in slave mode. If the RBF is loaded using ProcWizard, the ProcWizard application is in master mode and all instances of ProcFG are in slave mode. If ProcWizard is not used for loading the RBF, the first instance of ProcFG is in master mode (and it loads the RBF), and all other instances are in slave mode.

3. In case of multiple cameras, images of each camera will be saved in a unique folder in order to prevent mixing images from different cameras. The folder name of each camera will include the camera's name according to the settings in the **Misc. tab**. For example, if the **Save Folder and file prefix** in the Misc. tab is "Logs\Image", and the camera name is "Camera_2", images will be saved in "Logs_Camera_2" folder instead of in "Logs" folder. Accordingly, the image files' prefix will be "Image", for example, ***Image_0_0.raw*** and ***Image_1_1.raw***.

4.10. ProcFG Configuration File (.gxfg)

ProcFG uses a configuration file (.**gxfg**) that contains various parameters defining ProcFG behavior. The default file is **FGConfig.gxfg** located in the application folder. Using ProcFG GUI, the user can Open and Save configuration files. These actions are performed by the ProcFG library.
Most users who do not use ProcFG API do not have to understand the structure of the configuration file, since it is handled by the ProcFG library. A configuration file includes several sections - not all sections are relevant to all camera types.

4.10.1. Configuration File Example

Below is the content of a configuration file with comments describing the parameters. A detailed description of the parameters is included in other parts of the GUI description.

```
<?xml version="1.0"?>
<FG>
      <ROI> <Feature Name="Width">0</Feature>
            <Feature Name="Height">0</Feature>
            <Feature Name="OffsetX">0</Feature>
            <Feature Name="OffsetY">0</Feature>
            <Feature Name="DeltaX">0</Feature>
            <Feature Name="DeltaY">0</Feature>
            <Feature Name="StripOffsetX">0.000000</Feature>
            <Feature Name="StripOffsetY">0.000000</Feature>
            <Feature Name="RowDeltaX">0.000000</Feature>
            <Feature Name="RowDeltaY">0.000000</Feature>
            <Feature Name="ColDeltaX">0.000000</Feature>
            <Feature Name="ColDeltaY">0.000000</Feature>
            <Feature Name="FramesPerRow">0</Feature>
      </ROI>
      <CameraLink>
            <Feature Name="NumParallelPixels">1</Feature>
            <Feature Name="NumZones">1</Feature>
            <Feature Name="ZonesDirection">0</Feature>
            <Feature Name="IgnoreFVALMode">false</Feature>
            <Feature Name="IgnoreDVALMode">true</Feature>
            <Feature Name="BitsPerColor">8</Feature>
            <!--Available formats : Mono, Bayer, RGB, RGBA-->
            <Feature Name="Format">Bayer</Feature>
            <!--Available subformats for Bayer: 0-GR, 1-RG, 2-GB,
            3-BG-->
            <Feature Name="SubFormat">1</Feature>
      </CameraLink>
      <Acquisition>
            <Feature Name="SleepMs">0</Feature>
            <Feature Name="ReverseYMode">false</Feature>
            <Feature Name="RoiListMode">false</Feature>
            <Feature Name="RoiListPath"></Feature>
            <!--Available modes : NextFrame, LatestFrame-->
            <Feature Name="GrabMode">NextFrame</Feature>
            <Feature Name="FG_ID">0</Feature>

       </Acquisition>
       <Log><Feature Name="LogVerbosity">0</Feature>
            <Feature Name="LogSizeMB">5</Feature>
       </Log>

</FG>
```

4.10.2. Starting ProcFG using a Configuration File

ProcFG can be started by double-clicking a configuration (.**gxfg**) file; ProcFG is started using the configuration file. This requires file association between .gclfg andProcFG. This is automatically done as part of the installation of ProcFG.

5. CameraConfig and GenTL

In addition to the ProcFG GUI and ProcFG API, the ProcFG package includes the **CameraConfig** application and Gidel's **GenTL** (GenICam's Transport Layer) **API** (GdlFgGenTL.cti).

The Gidel GenTL API, based on GenICam's GenTL standard, enables users to develop or use 3rd party Camera Configuration applications. In case of CoaXPress, it also enables users to develop or use 3rd party streaming applications.

The purpose of the CameraConfig application is to allow configuration of CoaXPress and GenCP cameras connected to Gidel frame grabber boards programmed with the ProcFG firmware.

For additional information, please refer to the ***CameraConfig Data Book***.

The examples provided with the ProcFG package also refer to the CameraConfig application and to Gidel's GenTL API (see ***section 1.1***).

The ProcFG infrastructure includes support for GenTL (GenICam's Transport Layer) enabling seamlessly interface with 3rd party GenTL compatible image processing software packages. **MVTec** is an official partner of Gidel and its **Halcon** machine vision software has been verified to be fully compatible with the ProcFG. For information on additional compatible 3rd party software packages, please contact the Gidel Support (support@gidel.com).

6. Image Processing

6.1. Overview

The Gidel frame grabbing boards enable adding user image processing blocks in the FPGA acquisition pipeline and to customize the pipeline flow.

This option is only available if the ProcVision suite has been purchased.

For information on adding user image processing in the ProcFG FPGA flow, refer to the ***ProcVision Getting Started User Manual.pdf***.

7.1. APIs Overview

The ProcFG system has two independent APIs (Application Programming Interfaces):

1. ProcFG API for controlling the ProcFG image acquisition, processing path, and host interface (see ***section 7.2***). The ProcFG GUI and examples are based on the ProcFG API.

2. Serial communication API, allowing a user application to control the camera via serial communication. The serial communication API is implemented in Gidel's clsergdl.dll as defined by the Camera Link Specifications.

The following chapter provides information on developing the user application based on the ProcFG API (***section 7.2***).

For further explanation on using the serial communication interface, please refer to ***AN234 Using the Camera Link Serial Communication DLL with the ProcFG.***

7.2. ProcFG Library (and API)

ProcFG API is the Application Programming Interface to the ProcFG Library. The ProcFG application accesses the library via API methods.

This section describes the main features of the ProcFG library.

- ***Section 7.3***, ProcFG API Workflow, illustrates the workflow for operating ProcFG.

- ***Section 7.4***, ProcFG API Methods, shortly describes the methods of ProcFG API.

7.2.1. ProcFG Initialization

Initialization of ProcFG is done by first calling **Init()** method.

7.2.2. ProcFG Configuration

The ProcFG configuration is determined by the configuration file. The configuration file is loaded by calling **LoadConfig()** that loads the configuration parameters (Config) from a configuration (.gxfg**)** file. After the configuration file is loaded, the ProcFG application can modify some of the parameters by:

- Calling **GetConfig()** that gets the configuration parameters (Config) into the **Config Structure** `CONFIG_INFO`.
- Modifying the needed parameters in the **Config Structure** `CONFIG_INFO`
- Calling **SetConfig()** that sets the configuration parameters (Config) from the **Config Structure** `CONFIG_INFO` Then, the ProcFG application can optionally save the modified configuration by calling **SaveConfig()** that saves the configuration parameters (Config) to a configuration (.gxfg**)** file.

7.2.3. ProcFG Grab Modes

There are several Grab (acquisition) modes:

- **GRAB_NEXT**, where **All images** are acquired. In this mode, images are skipped if the images arrival rate is too high.
- **GRAB_LATEST**, where at any point, the Latest image is grabbed, possibly ignoring earlier images that have not been grabbed.
- **RoiListMode**, a special mode handling ROI requests using **QueueROI().** This is the real Dynamic API capture mode.

7.2.4. Starting/Stopping Grab

There are several methods for controlling Grab (acquisition):

- **Grab():** Initializes the Frame Grabber HW & SW and starts grabbing.
- **StopAcquisition():** Stops acquisition (grabbing).
- **AbortAcquisition():** Aborts acquisition (grabbing).

7.2.5. NextFrame & PreviousFrame modes

In the normal grabbing (acquisition) mode, frames are grabbed continually as they are received by the HW. The ProcFG application can grab one frame at a time, using **NextFrame()** and **PreviousFrame().** Going back to the normal mode is done using **Continue().** The relevant methods are:

- **Grab():** Normally, starts grabbing in normal mode. If **Grab()** is preceded by **NextFrame(),** grabbing starts in Next/Previous mode.
- **NextFrame():** Acquires (grabs) next frame, entering (or remaining in) Next/Previous mode.
- **PreviousFrame():** Acquires (grabs) previous frame, entering (or remaining in) Next/Previous mode.
- **Continue():** Continues normal acquisition (grabbing), leaving Next/Previous mode.

7.2.6. Camera Format

The configuration parameters include three parameters that are used to initialize the FPGA in case of Camera Link:

- **BitsPerColor**: The number of bits per color.
- `CONFIG_FORMAT`: The Camera format that can be: cMono, cBayer, cRGB, cRGBA.
- **SubFormat**: Relevant only in case of Bayer camera.

7.2.7. ProcFG Buffers and Queues

Grabbed (acquired) images are stored in buffers that are delivered to the ProcFG application. Buffers are maintained in two queues:

- **Input Buffer Pool**: Containing free buffers where new images can be stored.
- **Output Buffer Queue**: Containing buffers with stored images.

There are several methods dealing with buffers:

- **AnnounceBuffer():** Defines a buffer to be used by the Input Buffer Pool and the Output Buffer Queue.
- **QueueBuffer():** Queues (inserts) a buffer that was created using AnnounceBuffer() into the Input Buffer Pool.
- **ReturnBuffer():** Returns (removes) a buffer that was created using **AnnounceBuffer().**
- **FlushBuffers():** Releases all buffer resources.

7.2.8. Obtaining ProcFG Buffers (Images)

There are two schemes the ProcFG application can get buffers (images).

- Using **GetNextBufferInfo():** This method should be called in a thread that waits for buffers in the Output Buffer Queue. This method suspends execution of the calling program/thread until there is at least one buffer in the Output Buffer Queue, or a timeout occurs.

- Using **SetImageCallBack():** This method defines a user callback method to be called by a monitoring thread providing new images. This method starts a monitoring thread that calls back the callback method provided as a parameter when it gets a new image.

In both schemes, the user's program is responsible for queuing (inserting) received buffers to the Input Buffer Pool using **QueueBuffer(),** after processing of the buffer is completed. The user may want to queue the previous buffer while processing the current buffer.

A real application should use one of the above schemes. The examples delivered with ProcFG show 2 threads each using **GetNextBufferInfo(),** and also define a user callback method using **SetImageCallBack().** This is done only for demonstration purpose.

7.3. ProcFG API Workflow

The figure below illustrates the workflow for operating ProcFG.

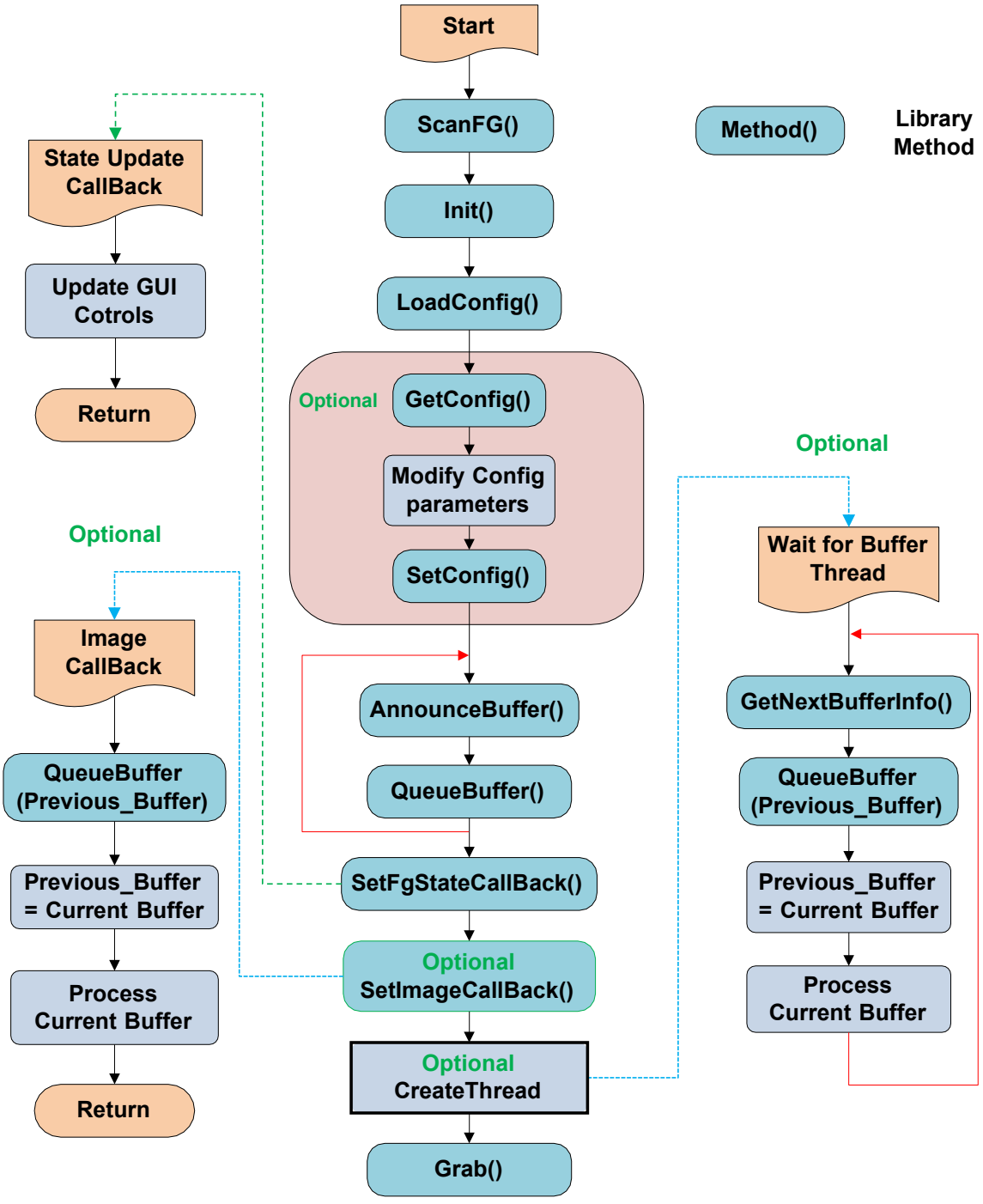


Figure 31: API Workflow

7.4. ProcFG API Methods

This section shortly describes the methods of ProcFG API.

- **Scan** Returns a list of available Proc boards
- **Init** Initializes the selected grabber board
- **LoadConfig** Loads the configuration parameters (Config) from a configuration (.gxfg**)** file
- **SaveConfig** Saves the configuration parameters (Config) to a configuration (.gxfg**)** file
- **GetConfig** Gets the configuration parameters (Config) into the Config Structure `CONFIG_INFO`
- **SetConfig** Sets the configuration parameters (Config) from the Config Structure `CONFIG_INFO`
- **GetFrameParameters** Gets Frame Parameters from HW (after getting the first image)
- **GetCameraType** Gets Camera Type based on the configuration parameters (Config)
- **GetCameraID** Gets Camera FG_ID from the array of Camera parameters built by Init()
- **GetCameraName** Gets Camera Name from the array of Camera parameters built by Init()
- **SetCamera** Sets Camera in the configuration parameters (Config) based on FG_ID or on Camera Name
- **GetMemorySize** Gets memory size
- **Grab** Initializes the Frame Grabber HW & SW and starts grabbing
- **StopAcquisition** Stops acquisition (grabbing)
- **AbortAcquisition** Aborts acquisition (grabbing)

• **GrabberReset**	Resets the Frame Grabber HW & SW
• **NextFrame**	Acquires (grabs) next frame
• **PreviousFrame**	Acquires (grabs) previous frame
• **Continue**	Continues normal acquisition (grabbing), leaving Next/Previous mode
• **GetLastError**	Returns a message describing the last error
• **SetFgStateCallBack**	Defines a callback method to be called by a monitoring thread providing ProcFG state
• **SetImageCallBack**	Defines a callback method to be called by a monitoring thread providing new images
• **AnnounceBuffer**	Defines a buffer to be used by the Input Buffer Pool and the Output Buffer Queue
• **ReturnBuffer**	Returns (removes) a buffer that was created using AnnounceBuffer()
• **QueueBuffer**	Queues (inserts) a buffer that was created using AnnounceBuffer() into the Input Buffer Pool
• **QueueROI**	Queues (inserts) an ROI request into the ROIs queue
• **GetNextBufferInfo**	Waits for a buffer in the Output Buffer Queue
• **FlushBuffers**	Releases all buffer resources
• **GetBufferInfo**	Get Buffer Information. This method is not documented since it is only for Gidel use.
• **GetNextErrorInfo**	Get Next Error Information. This method is not documented since it is only for Gidel use.
• **GetRunTimeInfo**	Returns current ProcFG state information
• **GetGrabberState**	Returns current ProcFG state
• **LogInfFormatString**	Logs Formatted Information to the Log file
• **GetProcRef**	Gets Proc Reference
• **GetFgVersion**	Gets ProcFG version and RBF information
• **GetSysInfo**	Gets ProcFG firmware information

7.5. CProcFgApi Class

This section details the methods of ProcFG API.

7.5.1. CProcFgApi::CProcFgApi

Function:

Default constructor of CProcFgApi class, providing interface for ProcFgApi.lib library. The following subsections detail the API methods belonging to the CProcFgApi class.

7.5.2. Scan

Function:

Scans the local computer PCI/PCIe bus and determines available Gidel Proc boards.

Declaration:

```
void ScanFG(SCAN_BOARD & scan);
```

Parameters:

SCAN_BOARD	Returned parameter structure including information about each available ProcFG board.

Return value:

None

Example:

```
fg::SCAN_BOARD scan;
Scan(scan)
```

7.5.3. CProcFgApi::Init (Prototype 1)

Function:

Initializes the ProcFG and configures camera information structure according to found data.

Declaration:

```
bool      Init(CAMERA_INFO & cameras);
```

Parameters:

cameras	Returned parameter. The structure contains information about cameras' status,type, name and ID.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::CAMERA_INFO    cameras;
bool result = pFg->Init(cameras);
```

7.5.4. CProcFgApi::Init (Prototype 2)

Function:

Initializes the specific ProcFG board and configures the camera information structure according to data found. Bus and slot information can be retrieved by calling the Scan function. It is possible to initialize operation of multiple ProcFG boards using the same application instance and, accordingly, to access the cameras of each board.

Declaration:

```
bool      Init(uint32_t bus, uint32_t slot,  CAMERA_INFO & cameras);
```

Parameters:

bus — Bus of destination Proc board

slot — Slot of destination Proc board

cameras — The structure contains information about cameras' status,type, name and ID.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::CAMERA_INFO    cameras;
bool result = pFg->Init(0, 2, cameras);
```

7.5.3. CProcFgApi::Init System (Prototype 3)

Function:

Initializes all ProcFG boards in the sytem, all cameras could be accessed by relevan ID and API.

Declaration:

```
bool      InitSystem(CAMERA_INFO & cameras);
```

Parameters:

cameras	Returned parameter. The structure contains information about cameras' status,type, name and ID.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::CAMERA_INFO    cameras;
bool result = pFg->InitSystem(cameras);
```

7.5.5. CProcFgApi::LoadConfig

Function:

Loads the configuration parameters from a configuration (.gxfg**)** file. The default configuration file is FGConfig.gxfg.

Declaration:

```
bool       LoadConfig(const char* ConFile);
```

Parameters:

ConFile	Relative path of ProcFG (.gxfg) configuration file.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool result = pFg->LoadConfig("my_configuration_file.gxfg");
```

7.5.6. CProcFgApi::SaveConfig

Function:

Saves the configuration parameters to a configuration (.gxfg**)** file.

Declaration:

```
bool       SaveConfig(const char* ConFile);
```

Parameters:

ConFile	Relative path of ProcFG (.gxfg) configuration file.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool result = pFg->SaveConfig("my_configuration_file.gxfg");
```

7.5.7. CProcFgApi::GetConfig

Function:

Configures Config Structure, `CONFIG_INFO`, according to the retrieved configuration parameters.

Declaration:

```
bool          GetConfig(CONFIG_INFO & Config);
```

Parameters:

`Config`	Returned parameter. A structure with the configuration parameters.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::CONFIG_INFO config;
pFg->GetConfig(config);
```

7.5.8. CProcFgApi::SetConfig

Function:

Sets the configuration parameters according to the Config Structure `CONFIG_INFO`.

Declaration:

```
bool       SetConfig(CONFIG_INFO & Config);
```

Parameters:

`Config` A structure with the configuration parameters.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::CONFIG_INFO config;
pFg->GetConfig(config);
config.Acq.GrabMode = fg::GRAB_NEXT;
pFg->SetConfig(config);
```

7.5.9. CProcFgApi::GetFrameParameters

Function:

Gets Frame Parameters from frame grabber (after capturing the first image).

Declaration:

```
bool       GetFrameParameters(FRAME_PARAM & FramePars);
```

Parameters:

`FramePars` Returned parameter: a structure with the frame parameters.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::FRAME_PARAM  frame_info;
bool result = pFg->GetFrameParameters(frame_info);
```

7.5.10. CProcFgApi::GetCameraType

Function:

Gets the Camera Type based on the configuration parameters (Config).

Declaration:

```
CAMERA_TYPE GetCameraType();
```

Parameters:

None.

Return value:

CameraType	The Camera Type based on the configuration parameters (Config).

Example:

```
fg::CAMERA_TYPE type = pFg->GetCameraType();
```

7.5.11. CProcFgApi::GetCameraID

Function:

Gets the Camera FG_ID (Frame Grabber ID).

Declaration:

```
int       GetCameraID();
```

Parameters:

None.

Return value:

The Camera FG_ID from the array of Camera parameters built by Init().

Example:

```
int ID =  pFg->GetCameraID();
```

7.5.12. CProcFgApi::GetCameraName

Function:

Gets the Camera Name.

Declaration:

```
char * GetCameraName();
```

Parameters:

None.

Return value:

CameraName	The Camera Name from the array of Camera parameters built by Init().

Example:

```
char *  name = pFg->GetCameraName();
```

7.5.13. CProcFgApi::SetCamera

Function:

Sets Camera in the configuration parameters based on FG_ID or on Camera Name.

Declaration:

There are two prototypes:

```
int SetCamera(int FG_ID);
int SetCamera(char * CameraName);
```

Parameters:

`FG_ID`	The Camera FG_ID to be searched in the array of Camera parameters built by Init().
`CameraName`	The Camera Name to be searched in the array of Camera parameters built by Init().

Return value:

In case of the first prototype, int SetCamera(int FG_ID):

`CameraNumber`	If the Camera FG_ID is found in the array of Camera parameters built by Init(), the Camera Number. If the Camera FG_ID is not found, -1.

In case of the second prototype, int SetCamera(char * CameraName):

`FG_ID`	If the Camera Name is found in the array of Camera parameters built by Init(), the Camera FG_ID. If the Camera Name is not found, -1.

Example:

```
int camera_id =  pFg->SetCamera(0);
```

7.5.14. CProcFgApi::GetMemorySize

Function:

Gets the Frame Grabber memory size in MB (after registers have been created as part of the Init() method).

Declaration:

```
bool      GetMemorySize(int& SizeMB);
```

Parameters:

SizeMB	Returned parameter. Frame Grabber memory size in MB.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
int size;
bool result = pFg->GetMemorySize(size);
```

7.5.15. CProcFgApi::Grab

Function:

Initializes the Frame Grabber HW & SW according to the current configuration parameters (Config), and starts grabbing. ProcFG enters the normal acquisition (grabbing) mode. This is an asynchronous (non-blocking) method; control returns to the calling program after an event indicating grabbing should start is set.

Declaration:

```
bool      Grab(int num = 0);
```

Parameters:

num indicates the number of frames to grab.
If ***num*** is zero, acquisition is performed in ACQUISITIONMODE_CONTINOUS mode, i.e., till abort/stop command.
If ***num*** is not zero, acquisition is performed in ACQUISITIONMODE_MULTIFRAME mode, until ***num*** frames are grabbed.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool result =  pFg->Grab();
```

7.5.16. CProcFgApi::StopAcquisition

Function:

Stops acquisition (grabbing), and waits until the Output Buffer Queue is empty. This is an asynchronous (non-blocking) method; control returns to the calling program after an appropriate value is written to the FPGA AcquisitionCommand register (without waiting until the Output Buffer Queue is empty).

Declaration:

```
bool      StopAcquisition();
```

Parameters:

None.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool result =  pFg->StopAcquisition();
```

7.5.17. CProcFgApi::AbortAcquisition

Function:

Aborts acquisition (grabbing). This is an asynchronous (non-blocking) method; control returns to the calling program after an event indicating acquisition should be aborted is set.

Declaration:

```
bool      AbortAcquisition();
```

Parameters:

None.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool result =  pFg->AbortAcquisition();
```

7.5.18. CProcFgApi::GrabberReset

Function:

Terminates grabbing and resets the Frame Grabber HW & SW.

Declaration:

```
bool      GrabberReset();
```

Parameters:

None.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError..

Example:

```
bool result =  pFg->GrabberReset();
```

7.5.19. CProcFgApi::NextFrame

Function:

Acquires (grabs) the next frame. If ProcFG is in the normal acquisition (grabbing) mode, it leaves this mode and enters the Next/Previous mode. If ProcFG is in the Next/Previous mode, it stays in this mode. This is an asynchronous (non-blocking) method; control returns to the calling program after an event indicating that the next frame should be acquired is set.

Declaration:

```
bool      NextFrame();
```

Parameters:

None.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool result =  pFg->NextFrame();
```

7.5.20. CProcFgApi::PreviousFrame

Function:

Acquires (grabs) the previous frame. If ProcFG is in the normal acquisition (grabbing) mode, it leaves this mode and enters the Next/Previous mode. If ProcFG is in the Next/Previous mode, it stays in this mode. This is an asynchronous (non-blocking) method; control returns to the calling program after an event indicating that the previous frame should be acquired is set.

Declaration:

```
bool      PreviousFrame();
```

Parameters:

None.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool result =  pFg->PreviousFrame();
```

7.5.21. CProcFgApi::Continue

Function:

Continues normal acquisition (grabbing), leaving the Next/Previous mode. This is an asynchronous (non-blocking) method; control returns to the calling program after an event indicating leaving the Next/Previous mode is set.

Declaration:

```
bool      Continue();
```

Parameters:

None.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool result =  pFg->Continue();
```

7.5.22. CProcFgApi::GetLastError

Function:

Returns a message describing the last error, and clears the message.

Declaration:

ERROR_ID GetLastError(char* LastError);

Parameters:

LastError	Returned parameter. Error message string.

Return value:

The error ID.	The error IDs are defined in the enum ERROR_ID. EID_NO_ERROR indicates no error.

Example:

```
char error_txt[1024];
fg::ERROR_ID  error =  pFg->GetLastError(error_txt);
```

7.5.23. CProcFgApi::SetFgStateCallBack

Function:

Defines a callback method to be called by a monitoring thread providing ProcFG state. This method starts a monitoring thread that calls back the callback method provided as a parameter, at a period provided as another parameter. The callback method is called with the current Frame Grabber state as a parameter.

Declaration:

```
bool      SetFgStateCallBack(STATE_CALLBACK pStateCallBackFunction, long
          UpdatePeriod);
```

Parameters:

`pStateCallBackFunction` The callback method to be called.

`UpdatePeriod` The period in milliseconds the callback method is called.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
//Callback Displaying global information
void status_callback(fg::CURRENT_STATE state)
{
std::cout << "Stdtus:   Received= " << state.Received <<" Extracted="
<<state.Extracted << std::setprecision(3) << " MemUse=" <<state.MemUse*100 <<"
Fps=" <<  state.Fps <<   std::endl;
}
bool result = pFg->SetFgStateCallBack(status_callback, 100);
```

7.5.24. CProcFgApi::SetImageCallBack

Function:

Defines a callback method to be called by a monitoring thread providing new images. This method starts a monitoring thread that calls back the callback method provided as a parameter when it gets a new image.

The callback method is called with image information structure as a parameter. The image information structure includes a pointer to the image buffer. This buffer should be returned to the Input Buffer Pool using QueueBuffer() after its information is no longer needed.

The monitoring thread waits for the return from the callback method before looking for newer images, so execution of the callback method should be as short as possible.

Declaration:

```
Bool       SetImageCallBack(IMAGE_CALLBACK pImageCallBackFunction);
```

Parameters:

pImageCallBackFunction The callback method to be called.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
void grabber_callback(fg::BUFFER_DATA buffer_info)
{
    static int  frame_count = 0; //Frame Counter
    static bool is_prev_buffer = false; //Flag indicating we have a previous
    buffer
    static fg::BUFFER_HANDLE prev_buffer_handle; //Buffer Handle of previous
    buffer
    frame_count++;       //Increment Frame Counter
    if (is_prev_buffer)     //if we have a Previous Buffer
           (reinterpret_cast<fg::CProcFgApi*> (buffer_info.pUser))->
           QueueBuffer(prev_buffer_handle);  //Queue the previous buffer

    prev_buffer_handle = buffer_info.hBuffer; //Save current handle for later
    Queuing
    is_prev_buffer = true; //Indicate we have a Previous Buffer for Queuing
    next time
    std::cout<< "CallBack: Number of Received Frames: " << frame_count <<
    std::endl;
}
bool result = pFg->SetImageCallBack(grabber_callback);
```

7.5.25. CProcFgApi::AnnounceBuffer

Function:

Defines a buffer to be used by the Input Buffer Pool and the Output Buffer Queue. It calls the Proc library method CreateDMABuffer() (that Creates a buffer, locks the memory for this buffer and performs all the necessary initializations for the DMA engine) using the first 2 parameters. In case of success, the method returns a Buffer handle – a handle to a structure that includes the DMA Buffer Handle (returned by CreateDMABuffer()) and the User Pointer. In case of error, it returns NULL.

Declaration:

```
BUFFER_HANDLE AnnounceBuffer(uint32_t dwSize, void* pBuffer, void* pUser);
```

Parameters:

`dwSize`	Buffer size in bytes. It must be a multiple of 32 bytes.
`pBuffer`	Pointer to the data buffer in the PC memory. Set to NULL to automatically allocate a buffer dwSize bytes long.
`pUser`	Pointer to User Data that is kept with the buffer.

Return value:

`Buffer handle`	A handle to a structure that includes the DMA Buffer Handle (returned by CreateDMABuffer()) and the User Pointer.
NULL	AnnounceBuffer() failed. For error description, use GetLastError().

Example:

```
fg::BUFFER_HANDLE handle = pFg->AnnounceBuffer(BUFFER_SIZE, nullptr,
nullptr);
```

IMPORTANT	1. dwSize Must be a multiple of **32 bytes**.

7.5.26. CProcFgApi::ReturnBuffer

Function:

Returns (removes) a buffer that was created using AnnounceBuffer(). The buffer is returned using the Proc library RemoveDMABuffer() method.

Declaration:

There are two prototypes:

```
bool       ReturnBuffer(BUFFER_HANDLE handle);
```

Parameters:

`handle`	A handle to a structure that was returned by AnnounceBuffer().

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::BUFFER_HANDLE handle;
bool result = pFg->ReturnBuffer(handle);
```

7.5.27. CProcFgApi::QueueBuffer

Function:

Queues (inserts) a buffer that was created using AnnounceBuffer() into the Input Buffer Pool. The ProcFG application should call this method once for each buffer after it is Announced (via AnnounceBuffer()); and in addition, each time the ProcFG application completes processing of a buffer that was obtained via GetNextBufferInfo() or by the Image CallBack method.

The method can be called before or after grabbing starts. However, the ProcFG application should make sure there are enough buffers to handle the input image stream. Note that the ProcFG library delivers buffers to the ProcFG application in the order they are filled using the DMA. However, the ProcFG application can return the buffers using QueueBuffer() in any order.

Declaration:

```
bool      QueueBuffer(BUFFER_HANDLE handle);
```

Parameters:

`handle`	A handle to a structure that was returned by AnnounceBuffer().

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::BUFFER_HANDLE handle;
bool result = pFg->QueueBuffer(handle);
```

7.5.28. CProcFgApi::QueueROI

Function:

Queues (inserts) an ROI request into the ROIs queue. This method should be used when ProcFG is in a special mode handling ROI requests, RoiListMode (or Dynamic API capture mode). This mode can be set in the Config file or by SetConfig() using the RoiListMode field of `CONFIG_INFO`. This method queues (inserts) an ROI request into the ROIs queue. The ProcFG application should call this method once for each ROI that has to be grabbed, providing the ROI parameters.

An ROI request is defined by the fields of a structure (of type `ROI_INFO`) containing information on the requested ROI, see ***section 0***, `ROI_INFO`. When an ROI request with the field LastRoi not 0 is processed by the library, the library waits till the Output Buffer Queue becomes empty. The library then continues to process ROI requests.

The ProcFG library handles the ROI requests in the order they appear in the ROIs queue. ROI requests do not have to be ordered in increasing frame/line order. ROIs are grabbed as long as they still exist in the Proc board memory. If a requested ROI is not yet available in the Proc board memory, the ProcFG library will wait until it is available. If a requested ROI does not exist any longer in the Proc board, or if it is out of bounds, the buffer is returned with a completion code (`BufferCompletionCode`) indicating this. See the declaration of `BufferCompletionCode` and the list of completion codes in ***section 1.1.1***, `BUFFER_DATA`.

ROI requests can be queued only after grabbing starts, using Grab() (since the ROIs queue is initialized at that time). Grabbed ROIs are returned in image buffers to the ProcFG application as in the regular grab mode, using the Output Buffer Queue.

In case of error, the method returns a negative number. Otherwise, the method returns 0.

Declaration:

```
bool      QueueROI(ROI_INFO& RoiInfo);
```

Parameters:

RoiInfo	A structure containing information on the requested ROI.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::ROI_INFO roi;
    roi.FrameNumX  = 0;
    roi.FrameNumY  = 10;
    roi.RoiXoffset = 0;
    roi.RoiYoffset = 0;
    roi.RoiWidth   = 320;
    roi.RoiHeight  = 240;
    roi.LastRoi    = 0;

if ( false == pFg->QueueROI(roi))
    {
           return error_handler(pFg);
    }
```

7.5.29. CProcFgApi::GetNextBufferInfo

Function:

Waits for a buffer in the Output Buffer Queue. This method suspends execution of the calling program/thread until there is at least one buffer in the Output Buffer Queue, or a timeout occurs. If a buffer is available, the buffer at the top of the queue (the oldest buffer) is removed from the queue. The method returns, in the parameter `BUFFER_DATA`, information on the buffer and returns status value

Declaration:

```
status        GetNextBufferInfo(BUFFER_DATA& FGBufferInfo, uint32_t dwTimeOut);
```

Parameters:

`FGBufferInfo`	Returned parameter. Structure containing information on the buffer at the top of the Output Buffer Queue.
`dwTimeOut`	Timeout period in milliseconds. 0xFFFFFFFF, indicates no timeout. 0 checks the state of the Output Buffer Queue and returns immediately, either with a timeout or with information on the buffer at the top of the Output Buffer Queue.

Return value:

`status_ok`	GetNextBufferInfo() completed successfully, without timeout. FGBufferInfo contains information on the returned buffer.
`status_ta`	GetNextBufferInfo() completed with timeout.
`status_failed`	GetNextBufferInfo() failed. For error description, use GetLastError().

Example:

```
fg::BUFFER_DATA info;
fg::status  status =  pFg->GetNextBufferInfo(info, 100);
```

7.5.30. CProcFgApi::FlushBuffers

Function:

This method releases and clears all buffer resources. It calls the Proc method RemoveDMABuffer(). In case of application provided buffers, the application is responsible for releasing the buffer memories after calling FlushBuffers(). The assumption is that ProcFG is not transferring data, i.e., it was stopped/aborted before calling this method. In case of error, the method returns false. Otherwise, it returns true.

Declaration:

There are two prototypes:

```
bool      FlushBuffers();
```

Parameters:

None.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
bool status  =  pFg->FlushBuffers();
```

7.5.31. CProcFgApi::GetRunTimeInfo

Function:

Returns current ProcFG run-time information.

Declaration:

```
bool        GetRunTimeInfo(RUNTIME_INFO& RunTimeinfo);
```

Parameters:

RunTimeinfo	Returned parameter. Structure containing the current ProcFG Run Time information.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::RUNTIME_INFO info;
bool status  =  pFg->GetRunTimeInfo(info);
```

7.5.32. CProcFgApi::GetGrabberState

Function:

Returns current ProcFG state.

Declaration:

```
bool      GetGrabberState(STATE& state);
```

Parameters:

STATE	Returned parameter. Structure containing the current ProcFG state.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::STATE state;
bool status = pFg->GetGrabberState(state);
```

7.5.33. CProcFgApi::LogInfFormatString

Function:

Logs formatted information to the log file. The log file is created when ProcFG is started. The file name has the form Grabber_{Date}_{Time}, e.g., Grabber_2014_Dec_29_14_44_11.log.
The maximal Log Size in Mbytes is defined in the Config file, and can be changed by SetConfig() using the LogSizeMB field of `CONFIG_INFO` structure.

Declaration:

```
bool        LogInfFormatString(char* pFormat,...);
```

Parameters:

`pFormat,...`	A group of one or more parameters defining a string to be appended as a new line to the Log file. The format of the parameters is like in sprintf(). The string is preceded by three fields: “INF”, time and thread_id.

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
pFg->LogInfFormatString("print to log value %d", 5);
```

7.5.34. CProcFgApi::GetProcRef

Function:

Gets a reference to Proc enabling access to the Proc API methods.

Declaration:

```
Proc* GetProcRef();
```

Parameters:

None.

Return value:

Proc	Reference to Proc.

Example:

```
Proc * proc = pFg->GetProcRef();
```

7.5.35. GetVersion

Function:

Gets ProcFG version and RBF information.

Declaration:

```
uint32_t   GetVersion();
```

Parameters:

None

Return value:

Returns ProcFG library version.

Example:

```
uint32_t fg_ver = fg::GetVersion();
```

7.5.36. CProcFgApi:: GetSysInfo

Function:

Gets the firmware information.

Declaration:

```
bool      GetSysInfo(SYS_INFO & info)
```

Parameters:

`info` Gets firmware information about Proc Board and IPs

Return value:

If the function succeeds, the return value is true.

If the function fails, the return value is false. To get extended error information, call method GetLastError.

Example:

```
fg::SYS_INFO  sys_info;
bool status = pFg->GetSysInfo(sys_info);
```

7.5.36. CProcFgApi:: FgFinalize()

Function:

Release the resources allocated to the ProcFG. This should be the last method called when concluding the operation with the CProcFgApi.

Declaration:

```
Void      FgFinalize();
```

Parameters:

```
No parameters
```

Return value:

None

Example:

```
FgFinalize();
```

7.6. CProcFgApi Structures & enums

This section describes the main structures and enums of the ProcFG API. These structures and enums are declared in ProcFgType.h.

7.6.1. SCAN_BOARD

Description:

The following structure declares the Board Info. It is used as a parameter in the Scan function.

Declaration:

```
enum {MAX_CBOARDS=10};

typedef struct {
    uint32_t count;     // Count of found Proc Boards
        struct
         {
          uint32_t   bus; // The Bus number where the Proc board is installed
          uint32_t   slot; // The Slot number where the Proc board is installed
          char           name[64]; //  Board type name
         }SlotInfo[MAX_CBOARDS];
} SCAN_BOARD;
```

7.6.2. STATE

Description:

The following declares the various states of ProcFG. It is used as part of thCURRENT_STATE structure that is used as a parameter in the State CallBack.

Declaration:

```
enum STATE
{
    STATE_READY,                    //Ready
    STATE_GRAB,                     //Busy (Grabbing) in regular Grab mode
    STATE_NEXT_PREVIOUS,            //Grabbing in Next/Previous mode
    STATE_ERROR                     //Error
};
```

7.6.3. RUNTIME_INFO

Description:

The following declares the Runtime info structure of ProcFG.

Declaration:

```
struct RUNTIME_INFO
{
uint32_t     Received;    //Number of Frames received from the Camera
             interface. In case of Line Camera it counts lines
uint32_t     Grabbed;    //Contents of the HW Frames Counter. In case of Line
             Camera it counts lines
uint32_t     Extracted;    //Number of Frames read from Proc board memory by
             ProcFG SW
uint32_t     MemorySize;    //Grabber Memory in MB
double       MemUsed;    //The fraction of memory used by images
uint32_t      Announced;   //Number of Announced buffers
uint32_t     OutBuffer;   //Number of buffers in the Output Buffer Queue
uint32_t     InputPool;   //Number of buffers in the Input Buffer Pool
uint32_t     NotQueued;   //Number of buffers Neither in the Input Buffer Pool
             nor in the Output Buffer Queue
uint32_t     OutPoolTotal;   //Total Number of buffers inserted in the Output
             Buffer Queue
uint32_t     OutErrorQueue;   //Number Errors in the Output Error Queue
uint64_t      OutErrorQueueTotal;   //Total Number of errors inserted in the
             Output Error Queue
uint32_t     FilledBuffer;        //1 if we have a buffer being filled, else 0.
};
```

7.6.4. CURRENT_STATE

Description:

The following declares the parameters of the current ProcFG state. This structure is delivered as a parameter in the State CallBack.

Declaration:

```
struct CURRENT_STATE
{
STATE         State;                        //ProcFG state
uint32_t      Received;                     //Number of Frames received from
              the Camera interface. In case of Line Camera it counts lines
uint32_t      AbsFrameID;                   //Latest Absolute frame ID, taking
              into account: Frame, Row and Column numbers (in ProcFG GUI, it is
              not updated in Pause state)
uint32_t      Extracted;                    //Number of Frames read from Proc
              board memory by ProcFG SW
double        MemUse;                       //The fraction of memory used by
              images
double        Fps;                          //Frames per sec.
bool          LineCamera;                   //TRUE: Line Camera, FALSE: Area
              Scan Camera
bool          UpdateFPS;                    //Used by the library to indicate
              that the FPS should be updated
uint32_t      LastRoi;                      //Last ROI
char          Message[ERROR_SIZE];          //Last Message for Status Bar
char          ErrorMessage[ERROR_SIZE];     //Last Error Message
int           ErrorID;                      //Error ID
};
```

7.6.5. CAMERA_FORMAT

Description:

The following declares the Camera Formats as received from the FPGA.

Declaration:

```
enum CAMERA_FORMAT {Raw, Mono, Planar, Bayer, RGB, RGBA, YUV,
         YCbCr601, YCbCr709};
```

7.6.6. CONFIG_FORMAT

Description:

The following declares the Formats as defined in the Format parameter of `CAMERALINK` used in GetConfig() and SetConfig().

Declaration:

```
enum  CONFIG_FORMAT {cMono, cBayer, cRGB, cRGBA};
```

7.6.7. BUFFER_HANDLE

Description:

The following declares the Buffer Handle. This is the Buffer handle that is returned by AnnounceBuffer().

Declaration:

```
typedef HANDLE  BUFFER_HANDLE;
```

7.6.8. DATA_HANDLE

Description:

The following declares the GIL Buffer Handle Structure. A handle to this structure is returned by AnnounceBuffer().

Declaration:

```
struct DATA_HANDLE
{
 DMA_BUFFER_HANDLE   DMAHandle;       //DMA Buffer Handle, created by
                                     CreateDMABuffer()
 void*               pUser;              //Pointer to User Data
 UINT32              BufferSizeBytes;    //Buffer Size in Bytes
};
```

7.6.9. BUFFER_DATA

Description:

The following declares the returned Buffer Information Structure. A reference to this structure is returned by GetNextBufferInfo() and provided as a parameter to the Image CallBack method.

Declaration:

```
struct BUFFER_DATA
{
    //Buffer handle that was returned by the AnnounceBuffer
    BUFFER_HANDLE       hBuffer;
     //Buffer Pointer
     uint8_t *           pBuffer;
     //Pointer to User Data that was supplied in AnnounceBuffer()
     void *              pUser;
     //Buffer Size in bytes
     uint32_t            BufferSizeBytes;
     //Number of bytes written into the buffer.
    //It should be <= Buffer size in bytes
     uint32_t            ActualSizeFilled;
    //Offset of the data in the buffer in number of Bytes
     uint32_t            Offset;
     //XPadding of the data in the buffer in number of Bytes
     uint32_t            BufferInfoXPadding;
     //Width of the data in the buffer in number of pixels
     uint32_t            BufferInfoWidth;
     //Height of the data in the buffer in number of lines
     uint32_t            BufferInfoHeight;
    //XOffset of the data in the buffer in number of pixelsfrom the image
    //origin to area of interest, using Real World Coordinates
     uint32_t            BufferInfoXOffset;
     //YOffset of the data in the buffer in number of linesfrom the image
     //origin to area of interest, using Real World Coordinates
     uint32_t            BufferInfoYOffset;
     //Strip Width in Bytes
     uint32_t            StripWidthBytes;
    //Strip Height in Lines
     uint32_t            StripHeight;
    //Needed Buffer Size in Bytes
     uint32_t            NeededBufferSize;
    //Row Number
     uint32_t            RowNumber;
    //Column Number
     uint32_t            ColumnNumber;
    //A sequentially incremented number of the current frame. The wrap around
    is 32 bits
     uint32_t            FrameID;
    //Absolute frame ID, taking into account: Frame, Row and Column numbers
     uint64_t            AbsFrameID;
    //Buffer Columns Status:
    //0:  All columns are relevant
    //1:  Some columns are not relevant
    //-1: All columns are not relevant
     int                 BufferColumnsStatus;
```

```
    //Transfer completion code:
    //0: Successful Completion
    //-1: Completion Error.
    //-2  In case of ROI List, the ROI is not available
    //-3: In case of ROI List, the ROI is out of bounds
    //-4: The Buffer is too small
     int                BufferCompletionCode;
    //1: Color Image, 0: Gray Scale Image
     uint32_t           ColorImage;
    //Number of bits per color.
    //For Gray Scale Image: 1-16. For Color Image: 5-16
     uint32_t           BitsPerColor;
     //The number of the first pixel in the first byte
     //when BitsPerColor is 1-4, and the image is Gray Scale
     uint32_t           FirstByteStartPixel;
     //1: an ROI List Image, 0: a regular Image
     uint32_t           RoiListImage
    //1: Last ROI (in case of an ROI List mode), 0: Not Last ROI
     uint32_t           LastRoi;
    //1: One Frame per Strip, 0: More than 1 Frame Per Strip
     uint32_t           OneFramePerStrip;
    //1: Line Camera, 0: Area Scan Camera
     uint32_t           LineCamera;
    //enum: Raw, Mono, Planar, Bayer, RGB, RGBA, YUV, YCbCr601, YCbCr709
     CAMERA_FORMAT      CameraFormat;
   //Camera Sub-Format. In case of YUV/YCbCr601/CbCr709
    //it is 411, 422, 444. In case of Bayer it is  0-GR, 1-RG, 2-GB, 3-BG.
   //In case of other formats, it is meaningless
    uint32_t           CameraSubFormat;
    //Number of Planar Planes
    uint32_t           PlanarPlanes;
    //XOffset of the data from the strip origin, in number of pixels
    uint32_t           Frame_XOffset;
    //YOffset of the data from the strip origin, in number of lines
    uint32_t           Frame_YOffset;
};
```

7.6.10. ROI_INFO

Description:

The following declares the structure of an ROI list element.

Declaration:

```
struct ROI_INFO
{
    uint32_t     FrameNumX;          //Line Camera: Frame number
                                     //along X. Area Scan Camera: Not used
    uint32_t     FrameNumY;          //Line Camera: Frame number along Y. Area
                                     //Scan Camera: Frame Number
    float        RoiXoffset;  //Offset of the ROI from the Frame origin along X
    float        RoiYoffset;  //Offset of the ROI from the Frame origin along Y
    uint32_t     RoiWidth;           //ROI Width
    uint32_t     RoiHeight;          //ROI Height
    uint32_t     LastRoi;            //Last ROI
};
```

7.6.11. CAMERA_TYPE

Description:

The following declares the various camera types.

Declaration:

```
typedef enum CAMERA_TYPE
{
   NOIP,
   CameraLink,
   cXp,
   HDMI,
   SDI,
   USERIP
};
```

7.6.12. GRAB_MODE

Description:

The following declares the various ProcFG grab modes.

Declaration:

```
typedef enum GRAB_MODE
{
    GRAB_NEXT,                      //Next Image (All Images)
    GRAB_LATEST                     //Latest Image

};
```

7.6.13. CameraInfo

Description:

The following declares the various fields of each camera returned by Init().

Declaration:

```
typedef struct {
    uint32_t count;
          struct
               {
                 char  CameraName[MAX_CAMERA_NAME];      //Camera Name
                 uint32_t             FgID;              //FG ID
                 uint32_t             CameraType;        //Camera Type
               }CameraInfo[MAX_CAMERAS];
} CAMERA_INFO;
```

7.6.14. CONFIG_INFO

Description:

The following declares the various fields in the `CONFIG_INFO` structure used in GetConfig() and SetConfig(). See a description of a configuration file with comments describing the parameters in ***section 4.10.1***, Configuration File Example.

Declaration:

```
struct ROI {
    uint32_t            Width;      //ROI Width. Affects grabbed image Width
    uint32_t            Height;     //ROI Height. Affects grabbed image Height
    uint32_t            OffsetX;  //World ROI OffsetX (in Output Coordinates).
                                    Affects only returned image parameters
    uint32_t            OffsetY;     //World ROI OffsetY (in Output
                                      Coordinates). Affects only returned
                                      image parameters
    uint32_t            DeltaX;     //World DeltaX (in Output Coordinates).
                                      Affects only returned image parameters
    uint32_t            DeltaY;     //World DeltaY (in Output Coordinates).
                                      Affects only returned image parameters
    float               StripOffsetX; //Strip OffsetX (shift along X). Affects
                                        grabbed image
    float               StripOffsetY; //Strip OffsetY (shift along Y). Affects
                                        grabbed image
    float               RowDeltaX;     //Row DeltaX when multiple ROIs per
                                       Frame. Affects grabbed image
    float               RowDeltaY;      //Row DeltaY when multiple ROIs per
                                        Frame. Affects grabbed image
    float               ColDeltaX;      //Column DeltaX when multiple ROIs per
                                        Frame. Affects grabbed image
    float               ColDeltaY;       //Column DeltaY when multiple ROIs per
                                         Frame. Affects grabbed image
    uint32_t            FramesPerRow;    //Frames Per Row
};

struct CAMERALINK {
    uint32_t            NumParallelPixels; //Number of parallel pixels (1-10)
    uint32_t            NumZones;           //Number of zones (1-10)
    ZONE_DIRECTION      ZonesDirection;     //Zones direction
    bool                IgnoreFVALMode;     //IgnoreFVAL mode
    bool                IgnoreDVALMode;     //IgnoreDVAL mode
    uint32_t            BitsPerColor;       //Number of Bits per color
                                              (8,10,12,14,16)
    CONFIG_FORMAT       Format;     //Format. enum: cMono, cBayer, cRGB, cRGBA
    uint32_t            SubFormat;  //Sub-Format for Bayer: 0-GR, 1-RG, 2-GB,
                                      3-BG
};

struct ACQUSITION {
    uint32_t            AcqFramesCnt;         //No. of frames /lines to
                                                grab;0 = continuous
    uint32_t            SleepMs;             //Sleep ms (0,1,2,…)
    GRAB_MODE           GrabMode;            //Grab mode. enum:
                                              RPOCFG_GRAB_NEXT,
                                              PROCFG_GRAB_LATEST
    bool                ReverseYMode;         //ReverseY mode
    bool                RoiListMode;          //RoiList mode
```

```
    bool                ExternalSource      //External source: true =
                                            enable start/stop from
                                            external source – see
                                            section 7.7.3
    char                RoiListPath[512];   //RoiList Path See Note 1
    uint8_t             FG_ID;              //FG_ID
};

struct LOG {
    uint8_t             LogSizeMB;                  //Log Size in MBytes
    uint32_t            LogVerbosity;               //Log Verbosity(0,1,2)
};

struct  CONFIG_INFO
{
    ROI                 Roi;
    CAMERALINK          Cl;
    ACQUSITION          Acq;
    LOG                 Log;
};
```

7.6.15. SYS_INFO

The following declares the various fields in the **SYS_INFO** structure used in `GetSysInfo()`.

```
struct  SYS_INFO {
    uint32_t            BoardSN;            // Board Serial Number
    bool                IsMaster;           // Board Master/Slave
    uint32_t            FwVer;              // Firmware version
    uint32_t            FwBuild;            // Firmware build
    uint32_t            FgVer;              // FG firmware version
    uint32_t            GilRegsVer;         // Gil Reg version
    uint32_t            GilTestPaternVer;   // Gil Test Pattern version
    uint32_t            GilAcqVer;          // Gil Aquesition version
    uint32_t            GilUserIpVer;       // Gil User IP version
    uint32_t            GilOutToMemVer;     // Gil Out of Memory version
    uint32_t            spare[20];          // reserved
  };
```

7.6.16. ERROR_ID

Description:

The following declares the various return codes (Error IDs) returned by GetLastError().

Declaration:

```
typedef enum ERROR_ID
{
   EID_NO_ERROR,           //No Error
   EID_OLD_RBF,            //This is an old version of the RBF
           //that is not supported by this version of ProcFG,
           //or the selected board is not supported by ProcFG.
   EID_TRANSFER_TO_LIB,    //Application initialization problem
           //(Transferring Cameras info to the library).
   EID_BOARD_INIT,         //Unable to open Gidel Board.
   EID_RBF_LOAD,           //The RBF file could not be loaded.
   EID_INIT_ERROR,         //ProcFG Initialization issue.
   EID_INIT_OPEN_TMP,      //Unable to open a temporary file in the
           //Logs Folder.
   EID_TRANSFER_FG_PARAMETERS_TO_LIB, //Application   initialization
           //problem (Transferring FG Parameters to the library).
   EID_FG_PARAMETERS_DAT_OPEN_READ,   //The FG Parameters dat file
           //(FG_Parameters.dat) cannot be opened for read.
   EID_FG_PARAMETERS_DAT_OPEN_WRITE,  //The FG Parameters dat file
           //(FG_Parameters.dat) cannot be opened for write.
   EID_FG_PARAMETERS_DAT_FORMAT,      //The format of the FG
           //Parameters dat file (FG_Parameters.dat) is not correct.
   EID_FG_PARAMETERS_DAT_WRITE,       //The FG Parameters dat file
           //(FG_Parameters.dat) cannot be written.
   EID_JUST_PRINT_TREE                //For Gidel use. End of
           //Printing Tree using OutputDebugString().
   EID_START_MONITORING_THREAD,       //Start Monitoring Thread:
           //Create Thread failure
   EID_CONFIGURATION_FILE_OPEN,       //The Configuration File cannot
           //be opened
   EID_NO_CAMERA_FG_ID,               //No camera matches the
           //Camera FG_ID
   EID_FG_NOT_INITIALIZED_GET_PARS_FROM_HW,
           //GetFrameParametersFromHw when ProcFG was not
           //initialized yet
   EID_FG_NOT_INITIALIZED_GET_MEMORY_SIZE,   //GetMemorySize when
           //ProcFG was not initialized yet
   EID_FG_NOT_INITIALIZED_GET_GRABBER_STATE, //GetGrabberState when
           //ProcFG was not initialized yet
   EID_FG_NOT_INITIALIZED_GRAB,              //Grab when ProcFG was
           //not initialized yet
   EID_FG_NOT_INITIALIZED_STOP_ACQSUISITION, //StopAcquisition when
           //ProcFG was not initialized yet
   EID_FG_NOT_INITIALIZED_GRABBR_RESET,      //GrabberReset when
           //ProcFG was not initialized yet
```

```
EID_FG_NOT_INITIALIZED_BOARD_RESET,       //BoardReset when ProcFG
        //was not initialized yet
EID_FG_NOT_INITIALIZED_NEXT_FRAME,        //NextFrame when ProcFG
        //was not initialized yet
EID_FG_NOT_INITIALIZED_PREVIOUS_FRAME,    //PreviousFrame when
        //ProcFG was not initialized yet
EID_FG_NOT_INITIALIZED_CONTINUE,          //Continue when ProcFG
        //was not initialized yet
EID_FG_NOT_INITIALIZED_ABORT_ACQUISITON,  //AbortAcquisition when
        //ProcFG was not initialized yet
EID_FG_NOT_INITIALIZED_GET_RUNTIME_INFO,  //GetRunTimeInfo when
        //ProcFG was not initialized yet
EID_FG_NOT_INITIALIZED_SET_IMAGE_CALLBACK,//SetImageCallBack when
        //ProcFG was not initialized yet
EID_FG_NOT_INITIALIZED_ANNOUNCE_BUFFER,   //AnnounceBuffer when
        //ProcFG was not initialized yet
EID_FG_NOT_INITIALIZED_RETURN_BUFFER,     //ReturnBuffer when
        //ProcFG was not initialized yet
EID_FG_NOT_INITIALIZED_QUEUE_BUFFER,      //QueueBuffer when
        //ProcFG was not initialized yet
EID_FG_NOT_INITIALIZED_QUEUE_ROI,         //QueueROI when ProcFG
        //was not initialized yet
EID_FG_NOT_INITIALIZED_GET_NEXT_BUFFER_INFO,//GetNextBufferInfo
        //when ProcFG was not initialized yet
EID_FG_NOT_INITIALIZED_FLUSH_BUFFERS,     //FlushBuffers when
        //ProcFG was not initialized yet
EID_CONTROL_THREAD_CREATE,                //Control thread:
        //create failure
EID_CONTROL_THREAD_TIMEOUT,               //Control thread:
        //exit  timeout
EID_THREAD_NOT_STARTED_GET_CAMERA_MODE,   //GetCameraMode when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_IS_MXN,            /IsAreaMxN when Thread
        //was not started yet
EID_THREAD_NOT_STARTED_GET_MEM_SIZE,      //GetMemorySize when
        //Thread was not started yet
EID_GET_MEM_SIZE_REG_NOT_CREATED,         //GetMemorySize when
        Registers not created yet
EID_THREAD_NOT_STARTED_GET_FG_STATE,      //GetGrabberState when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_GET_FRAME_WIDTH,   //GetFrameWidth when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_GET_FRAME_HEIGHT,  //GetFrameHEIGHT when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_GET_1ZONE,         //GetReorder1ZoneOnly
        //when Thread was not started yet
EID_THREAD_NOT_STARTED_GET_CAMERA_FORMAT, //GetCameraFormat when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_GET_CAMERA_SUB_FORMAT,//GetCameraSubFormat
        //when Thread was not started yet
EID_THREAD_NOT_STARTED_GET_CAMERA_BPC,
        //GetCameraBitsPerColor when Thread was not started yet
EID_THREAD_NOT_STARTED_GET_CAMERA_COLOR,  //GetCameraColor when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_GET_CAMERA_PLANAR_PLANES,
        //GetCameraPlanarPlanes when Thread was not started yet
EID_THREAD_NOT_STARTED_GRABBER_RESET,     //GrabberResetM when
      //Thread was not started yet
EID_THREAD_NOT_STARTED_BOARD_RESET,       //BoardReset when Thread
        //was not started yet
```

```
EID_THREAD_NOT_STARTED_GET_RUN_TIME_INFO, //GetRunTimeInfo when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_STOP_ACQUISITION,  //StopAcquisition when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_SET_CAMERAS_DAT,   //SetCamerasDat when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_SET_FG_PARAMETERS, //SetFgParameters when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_GRAB,              //Grab when Thread was
        //not started yet
EID_THREAD_NOT_STARTED_NEXT_FRAME,        //NextFrame when Thread
        //was not started yet
EID_THREAD_NOT_STARTED_PREVIOUS_FRAME,    //PreviousFrame when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_CONTINUE,          //Continue when Thread
        //was not started yet
EID_THREAD_NOT_STARTED_ABORT_ACQUISITION, //AbortAcquisition when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_ANNOUNCE_BUFER,    //AnnounceBuffer when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_RETURN_BUFFER,     //ReturnBuffer when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_QUEUE_BUFFER,      //QueueBuffer when
        //Thread was not started yet
EID_THREAD_NOT_STARTED_QUEUE_ROI,         //QueueROI when Thread
        //was not started yet
EID_THREAD_NOT_STARTED_GET_NEXT_BUFFER_INFO,//GetNextBufferInfo
        //when Thread was not started yet
EID_THREAD_NOT_STARTED_FLUSH_BUFFERS,     //FlushBuffers when
        //Thread was not started yet
EID_ANNOUNCE_MAX_BUFFERS,                 //AnnounceBuffer:
        //Max Number of buffers exceeded
EID_BUFFER_TOO_BIG,                       //AnnounceBuffer:
        //Buffer is bigger than half of the Memory size
EID_REMOVE_DMA_BUFFER,                    //ReturnBuffer:
        //RemoveDMABuffer failed
EID_RETURN_BUFFER_NO_ANNOUNCED_BUFFERS,   //ReturnBuffer:
        //There are no announced buffers
EID_QUEUE_BUFFER_ANNOUNCED_EXCEEDED,      //QueueBuffer:
        //The Number of Announced buffers exceeded
EID_ROI_QUEUE_FULL,                       //QueueROI: ROI Queue
        //is full
EID_START_IMAGE_CALL_BACK_CREATE_THREAD,
        //StartImageCallBackThread: Create Thread failure
EID_FLUSH_BUFFERS_REMOVE_DMA_BUFFER,      //FlushBuffers:
        //RemoveDMABuffer() failed
EID_ANNOUNCED_BUFFER_SMALL,               //The announced buffer
        //is too small
EID_LAST_ROI_DISPLAYED,                   //Last ROI is displayed
EID_FG_NOT_READY,                         //Grab when ProcFG is
        //not ready
EID_GRAB_STOPPED_ABORT_COMMAND,           //Grab Stopped
        //Automatically: by Abort Command
EID_GRAB_STOPPED_STOP_COMMAND,            //Grab Stopped
        //Automatically: by Stop Command
EID_GRAB_STOPPED_ABORT_RESET_ACQ_COUNTER, //Grab Stopped
        //Automatically: by Abort/Reset/Acq Counter
EID_GRAB_STOPPED_BUFFERS_NUMBER_SMALL,    //Grab Stopped
        //Automatically: Number of buffers should be at least 2
```

```
EID_GRAB_STOPPED_LINE_WIDTH_BIG,          //Grab Stopped
        //Automatically: Camera Line Width is too big for the
        //current RBF
EID_BITS_PER_COLOR_SMALL,                 //Color Image with
        //BitsPerColor 1-4 is Not supported
EID_GRAB_STOPPED_MISSING_CL_CLOCKS,       //Grab Stopped
        //Automatically: Missing Camera Link clock(s)
EID_GRAB_STOPPED_FPGA_PARAMETER_WIDTH_LT8,//Grab Stopped
        //Automatically: Illegal FPGA parameters (Width<8)
EID_GRAB_STOPPED_FPGA_PARAMETER_HEIGHT_0, //Grab Stopped
        //Automatically: Illegal FPGA parameters (Height=0)
EID_GRAB_STOPPED_FPGA_PARAMETER_BPC_0,    //Grab Stopped
        //Automatically: Illegal FPGA parameters (BitsPerColor=0)
EID_GRAB_STOPPED_FPGA_PARAMETER_BPC_GT16, //Grab Stopped
        //Automatically: Illegal FPGA parameters BitsPerColor>16)
EID_GRAB_STOPPED_FPGA_PARAMETER_FRAME_FORMAT, //Grab Stopped
        //Automatically: Illegal FPGA parameters (FrameFormat>8)
EID_GRAB_STOPPED_FPGA_PARAMETER_PLNAR_PLANES,//Grab Stopped
        //Automatically: Illegal FPGA parameters
        //(CameraPlanarPlanes=0)
EID_GRAB_STOPPED_IMAGE_TOO_BIG,           //Grab Stopped
        //Automatically: The received image is bigger than
        //MemorySize/2
EID_GRAB_STOPPED_FRAME_OUTSIDE_X,         //Grab Stopped
        //Automatically: The requested frames are outside
        //X limits
EID_GRAB_STOPPED_FRAME_OUTSIDE_Y          //Grab Stopped
        //Automatically: The requested frames are outside
        //Y limits
EID_GRAB_STOPPED_COLUMN_DELTAY_0,         //Grab Stopped
        //Automatically: Column DeltaY = 0 };
```

7.6.17. FRAME_PARAM

Description:

The following declares the various fields of the structure FRAME_PARAM returned by GetFrameParameters ().

Declaration:

```
struct  FRAME_PARAM                 //Frame Parameters Structure Declaration
{
 bool      CameraMode;   //Camera Mode: 1:Line Camera, 0:Area Scan(Frame)Camera
 uint32_t        Width;            //Frame Width in Pixels
 uint32_t        Height;           //Frame Height. 1 if Line Camera
 CAMERA_FORMAT   CameraFormat;     //Camera Format: enum  CAMERA_FORMAT {Raw,
                                   //Mono, Planar, Bayer, RGB, RGBA, YUV,
                                   //YCbCr601, YCbCr709}
 uint8_t   CameraSubFormat;    //Camera Sub-Format. Bayer: 0-GR, 1-RG,
                              //2-GB, 3-BG. YUV/YCbCr601/YCbCr709: 411/422/444
 uint8_t   CameraBitsPerColor;      //Number of Bits Per Color
 bool      CameraColor;             //Camera Color. 1: Color image,
                                   //0: Gray scale image
 uint32_t  CameraPlanarPlanes;      //Number of Planar Planes. 1 if not Planar
};
```

7.7. API Application Examples

Gidel provides example project that demonstrate the use of ProcFG API methods to create the skeleton of a simple Frame Grabber application.

7.7.1. FgExample

FgExample is a **Console** application for Windows and Linux. The example is both for CameraLink and for cXp. The example does not use GenTL and does not perform Camera Configuration. It uses ProcFG API for grabbing.

In case of cXp, a ***Configuration application*** (Gidel's CameraConfig or a 3rd party application) can be used for camera configuration. The Configuration application should be started in order to enable grabbing. The Configuration applications use GenTL that uses Gidel's GenTL configuration file (GenTlConfig.ini). For information on GenTL, GenTlConfig.ini and camera configuration, refer to the ***CameraConfig Data Book***.

7.7.2. TriggerExample Console Example

The **TriggerExample** is a **Console** application for Windows and Linux. The default path of the example is:

<installation dir>\ GidelIPs\Examples \TriggerExample

This simple example demonstrates how to use the Global registers to configure the ProcFG triggering. In this example, the triggering source can be selected either from encoders or from an auto-generate a trigger signal with a 50% duty cycle. The camera will receive the trigger signal via the CC1 line.

In this example, you will need to run three applications in the following order:

1. Run the **ProcFG** application.

2. Use your applications to configure the camera via the serial communication to receive triggering from the CC1 line. For Camera Link cameras, in the case of Windows, use **clsergdl.dll** that utilizes the serial communication. For explanation on using **clsergdl.dll**, refer to the Camera Link Specifications.

3. Run the **TriggerExample** application using the following extensions:

   a. **-a** for internal auto-triggering
   b. **-e #** (e.g., -e 50) for encoder triggering every **#** encoder ticks/steps.

In the **TriggerExample** application code, the internal trigger's period is defined in µS for a total of 33K µS, i.e. 33 mS. Accordingly, the ProcFG will display a frame rate of 30 frames/sec.

For explanation of the example's code, please refer to the **TriggerExample** source code's comments.

7.7.3. Camera Triggering and I/O Control

External and internal signal sources can be used for triggering and controlling the user camera, for general system and peripheral control, and for controlling the grabber's start/stop acquisition. The I/O connectivity can be software configured and controlled via the **global_regs** class. For detailed information on using the **global_regs**, refer to ***Global Regs - Grabber IO Control Data Book***. In addition to the **global_regs** connectivity configuration, in the software application, the external source acquisition control (start/stop) must be enabled via the CONFIG_FIG structure as detailed in section **7.6.14 CONFIG_INFO** (see the ExternalSource field).

The TriggerExample (see section 7.7.2) demonstrates how to use the global_regs to trigger the user camera in two ways: 1. From an internal FPGA source. 2. From an external I/O source.

For more information on the HawkEye's external I/O connectors, refer to your HawkEye's Data Book.

For Camera Link cameras, all CC lines can be used as triggering sources. For CoaXPress cameras, only CC0 can be used for triggering as defined in the CoaXPress protocol.

8. References

List of reference documents:

- *HawkEye-20GigE Frame Grabber Data Book*
- *HawkEye-CL Camera Link Frame Grabber Data Book*
- *HawkEye-20GigE Frame Grabber Data Book*
- *Camera Link Specifications V.2.0*
- *CoaXPress Standard specifications Version 1.1.1*
- *GenICam GenTL Standard Version 1.5*
- *GenICam GenCP Generic Control Protocol Version 1.2*
- *CameraConfig Data Book*
- *GigE API Data Book*
- *InitCam API User Manual*
- *Proc API Data Book*
- *Global Regs - Grabber IO Control Data Book*

9. Glossary

Table 8: Table of Acronyms

Acronyms	Description
ACK	Acknowledge
AVI	Audio-Video Interleaved
CC	Camera Control [signals]
CL	Camera Link
Codec	Coder-Decoder
CXP	CoaXPress
DDR	Double Data Rate
DMA	Direct memory access
DRAM	Dynamic Random Access Memory
DUT	Design Under Test
DVAL	Data Valid
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
FVAL	Frame Valid
GENICAM	Generic Interface for Cameras
GENTL	GenICam's Generic Transport Layer Interface
GENCP	GenICam's Generic Control Protocol
GIL	Gidel Imaging Library
GS	Grayscale, e.g., GS8 = Greyscale 8 bits/pixel
HDL	Hardware Description Language
IP	Intellectual Property Core
LVAL	Line Valid
LVDS	Low Voltage Differential Signaling
PCIe	PCI express interface
PSDB	Proc Daughterboard
RGB24 (30,36)	Red-Green-Blue, 24 bits/pixel (30,36)
REQ	Request
ROI	Region Of Interest
RTL	Register Transfer Logic
RX	Receive
TX	Transmit

10.Revision History

Table 9: Revision History

Date	Description	Changes
February 2013	ProcFG V1.0 Beta	Initial release
September 2013	ProcFG V1.0.1	• Support for Windows Server 2008 64-bit. • Addition of **zooming** and **panning** to image display with zoom factor display in GUI. Zooming can be done using the mouse wheel or the “+” / ”-” keys. Panning can be done by dragging the mouse with the left button pressed or by the 4 arrow keys. • Addition of **Rectangular Zoom** allowing selecting a rectangle to be zoomed by dragging on the image with the **Right** mouse button pressed. • Addition **Define Zoom Center**: left clicking the mouse defines the pixel under the cursor as the zoom center. • Addition of **double-clicking** the **left** mouse button, toggles between two zoom states: 100% zoom and zoom to Full Window. • Addition of **Move to Corner** and **Move to Center**: when **double-clicking** the **left** mouse button with the **Ctrl Key** pressed changes the zoom center either to one of the four corners or to the image center respective to the pointer location. Pressing the **Home** key toggles between the four corners or to the image center.

Date	Description	Changes
		• Addition of **Next Frame**, **Previous Frame**, **Save Image** buttons to the Toolbar and corresponding items to the Control Menu. • Addition of **display data** window showing hexadecimal data representation of pixels around the mouse position. Cancelled the hexadecimal display of pixel values in the four image corners. • Addition of Keyboard Accelerators, e.g., **G** to start grabbing, **A** to abort, etc. • Addition of Quick Guide reference in Help menu. • Addition of **Ignore FVAL** option. • Removed Single Frame mode from the Acquisition tab. • Changed the location of the **Buttons** in the Toolbar and the Control Menu. • Changed some of the Toolbar button images. • Removed the four unused lines (**Roi Move Left / Up/Right / Down**) from the Control Menu. • Various bug fixes.
November 2013	ProcFG 1.1	• Support for image acquisition from multiple cameras • Addition of ***FPGA_templates*** folder • Location of RBF and template set has been changed from ***app\RBFs*** to ***FPGA_templates\RBFs***
April 2014	ProcFG 1.2	• Support for Win-7 32-bits and Win-XP 64-bits • Changed FG_Cameras.dat settings file

Date	Description	Changes
		• Camera Link GUI has been changed to include additional image information • Support for Bayer and RGBA formats • Addition of Frame Information area at the bottom of the application window • Moved Camera Name List box to the bottom of the Acquisition tab • Support for multiple camera types, e.g., Line and Area cameras simultaneously, based on multiple Frame Grabbers
June 2015	ProcFG 2.0	• Addition of API methods allowing the user to control the ProcFG library (for Windows and Linux) • Two new examples demonstrating the ProcFG API usage • A new example demonstrating the Proc registers API usage • Addition of **Profile** window showing a graph of pixel values along a line • Addition of new Keyboard Accelerators, e.g., **P** to show/hide the Profile window • In the **Misc** tab, the Save Folder location has changed from **Logs** to **..\..\FgImages**.
July 2016	ProcFG 2.0	• Removed reference to AN234 requiring manual setup of clser
April 2017	ProcFG 2.5	• Support for HawkEye_CL frame grabber • Support for Windows 10 • Addition of PofLoader application for loading binary files on Arria10 and Stratix V devices

Date	Description	Changes
		• Image capture from CoaXPress cameras by ProcFG GUI and API • Addition of GenTL API methods for CoaXPress • Addition of CameraConfig application for configuration of CoaXPress cameras • Support for 3rd party GenTL applications for configuration of CoaXPress cameras • A new example demonstrating GenTL API usage for CoaXPress • Moved the following files to GenTLbin64/32: FG_Parameters.dat, FG_Cameras.dat, Fg_ProceIII_80.rbf, Fg_ProceIII_110.rbf, Fg_ProceIV_360.rbf • Additions of the following methods for supporting GenTL: GetBufferInfo and GetNextErrorInfo. These methods are for Gidel use only. • Addition of new prototypes to ReturnBuffer and FlushBuffers. They are only for Gidel use only. • The definition of dwTimeOut in the GetNextBufferInfo method has been extended. • Update of **Chapter 6: Image Processing**
September 2017	ProcFG 2.5.1	• Addition of Profile Line Zoom In / Out • Addition of Pan Profile Line along itself • Changed the Show/Hide Profile keyboard accelerator (from P to **F**) • Addition of configuration of GenCP cameras by

Date	Description	Changes
		CameraConfig and 3rd party GenTL applications • Addition of configuration of ProcFG registers/fields by CameraConfig and 3rd party GenTL applications for **CoaXPress**, **GenCP** and **non-GenCP** Camera Link cameras • The GenTL example now supports **CoaXPress**, **GenCP** and **non-GenCP** Camera Link cameras • There are new "#define" parameters in the GenTL example defining with/without Streaming, and with/without **Accessing Camera Registers** • Changed a parameter in the GenTlConfig.ini file
January 2018	ProcFG 2.6	• Update of CameraConfig application: o Addition of macro recording feature o Addition of Toggle button enabling writing to single or multiple cameras o Removal of Refresh button • Support for streaming by 3rd party applications based on GenTL, from cXp cameras • Addition of two examples using both ProcFG API and GenTL • Explanation on Gidel GenTL and CameraConfig has been moved to the CameraConfig Data Book • Support for Halcon software by MVTec - a machine vision software based on GenICam standard.
December 2018	ProcFG 2.7.4	• Changed the prototype of the API method GetVersion() • Addition of a new prototype to the GetVersion API method

Date	Description	Changes
		• Update of the ERROR_ID return codes • Addition of a possibility for a separate CameraConfig installation • Addition of Utilities64 folder including: PofLoader, SofLoader, IpScan and TempMonitor • Addition of ProcFG_GenTL Console Example for Linux • Addition of ProcFG GenTL Console View Example for Windows and Linux • Changed the structure and contents of GenTlConfig.ini
April 2019	ProcFG 2.7.6	• Changed the colors in the Profile window • Changed the size of the Profile window • Addition of Profile statistics in current cursor position • Addition of Profiles history • Addition of a few Tap combinations in case of Camera Link • Addition of Statistics Data tool (Beta version)
July 2019	ProcFG 2.7.7	• Addition of Histogram tool • Release of Statistics Data tool (not as a Beta version) • Changed **the** structure SCAN_PROC_FG • Addition of the enum PROC_BOARD_ERROR • Improved reliability • New firmware binary files (RBF/POF/SOF) files
November 2019	ProcFG 2.7.8	• Improved GUI support for Chinese interface

Date	Description	Changes
		• Update Image Processing chapter
April 2020	ProcFG 2.7.8	Addition of External Triggering section
January 2021	ProcFG 2.8.0	• Restructured API • Revised examples • Improved GUI support for East Asian languages • Additions of XML configuration files • Addition of grabbing start/stop via the external I/O connectors
September 2021	ProcFG 2.8.0	Updated Camera Control (CC) configuration description
December 2021	ProcFG 2.8.1	• Support for GigE Vision cameras • Replaced ProcFGAutoTrigger example with a new TriggerExample demonstrating triggering from encoder and an external source using global_regs • Updated External Triggering section. • Addition of reference to *Global Regs - Grabber IO Control Data Book* explaining how to use Global Regs • Improved CoaXPress cameras discovery performance.
March 2023	ProcFG 2.8.4	Support for multiple instances of ProcFG in the same process.
July 2023	ProcFG 2.8.4	Added explanation on how to enable the start/stop acquisition via an external source
October 2023	ProcFG 2.8.5	• Added the FgFinalize() method. • Support for multiple

		boards and cameras on a single software application instance
January 2024	ProcFG 2.8.6	• Removed reference to FG_Cameras.dat. The camera(s) configuration is read from the firmware and thus the .data file is no longer required.