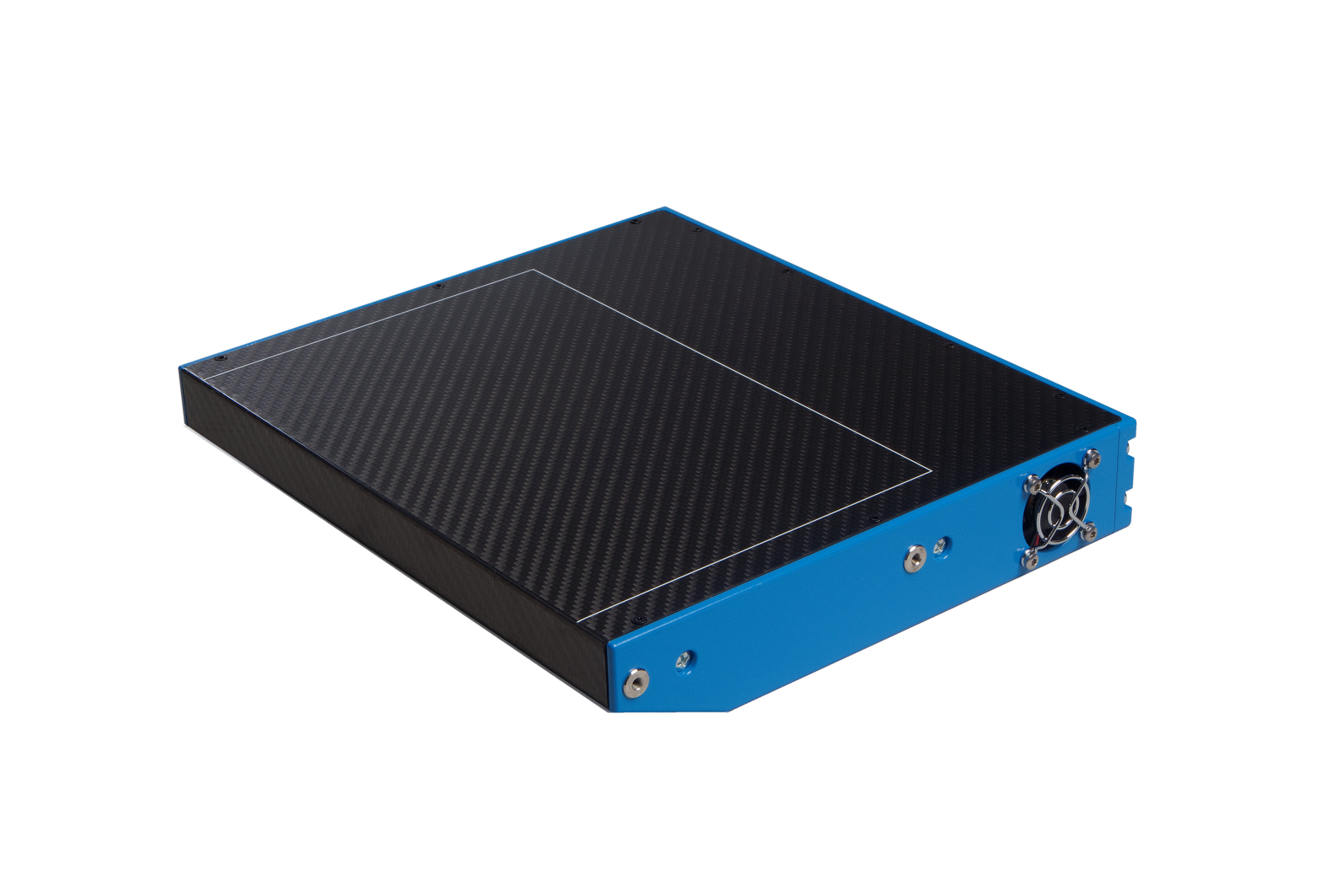
**Dexela X-ray Detector Software User Manual**



Document no.: DX-001452

Version: 1.0

Date: 02/05/13

|  |
| --- |
| For a complete listing of our global offices, visit www.perkinelmer.com  ©2014 PerkinElmer, Inc. All rights reserved. The PerkinElmer logo and design are registered trademarks of PerkinElmer, Inc. or its subsidiaries,  in the United States and other countries. All other trademarks not owned by PerkinElmer, Inc. or its subsidiaries that are depicted herein are the property of their respective owners. PerkinElmer reserves the right to change this document at any time without notice and disclaims liability for editorial, pictorial or typographical errors. |

Contact Information:

**Dexela Limited, a PerkinElmer company**

Wenlock Business Centre

50 – 52 Wharf Rd

London N1 7EU

+44 20 7148 3107

www.dexela.com

[www.Perkinelmer.com](http://www.Perkinelmer.com)

*For enquiries and technical support:*

[MI.support@perkinelmer.com](mailto:DexelaSupport@perkinelmer.com)

Contents

[1 General 6](#_Toc399415231)

[1.1 Definitions and Abbreviations 6](#_Toc399415232)

[1.2 Information and Warnings 7](#_Toc399415233)

[1.3 System Requirements 8](#_Toc399415234)

[2 Installation Guide 9](#_Toc399415235)

[2.1 Installing Peripherals 10](#_Toc399415236)

[2.2 Installing the SDK 12](#_Toc399415237)

[2.3 DriverInstallerGUI 14](#_Toc399415238)

[3 SCap 17](#_Toc399415239)

[3.1 Overview 17](#_Toc399415240)

[3.2 The SCap User Interface 17](#_Toc399415241)

[3.3 Image File Formats 19](#_Toc399415242)

[3.4 Image Window Functions 19](#_Toc399415243)

[3.5 Control Panel Button Functions 20](#_Toc399415244)

[3.6 Control Panel Check Box Functions 21](#_Toc399415245)

[3.7 Examples of Imaging Tasks 28](#_Toc399415246)

[3.8 Troubleshooting 30](#_Toc399415247)

[4 Sorting Schemes 31](#_Toc399415248)

[4.1 Sorting 1512 31](#_Toc399415249)

[4.2 Sorting 2315 32](#_Toc399415250)

[4.3 Sorting 2923 32](#_Toc399415251)

[5 Image Corrections 33](#_Toc399415252)

[5.1 Offset Correction 33](#_Toc399415253)

[5.2 Linearity Correction 33](#_Toc399415254)

[5.3 Gain Correction 33](#_Toc399415255)

[5.4 Geometric Correction 35](#_Toc399415256)

[5.5 Defect Correction 36](#_Toc399415257)

[6 DexelaDetector API 38](#_Toc399415258)

[6.1 DexelaDetector Class 38](#_Toc399415259)

[6.2 BusScanner Class 41](#_Toc399415260)

[6.3 DexImage Class 41](#_Toc399415261)

[6.4 DexelaException Class 43](#_Toc399415262)

[6.5 DexelaDetector API Wrapper Libraries 43](#_Toc399415263)

[6.6 DexelaDetector API Example Source Code 44](#_Toc399415264)

[Appendix A – SMV File Format 45](#_Toc399415265)

|  |  |
| --- | --- |
| References: | |
| **Detector Information** | The detector hardware and functionality is fully described in the Detector Technical Manual. |
|  |  |
|  |  |

# Introduction

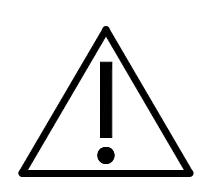
Dexela X-ray detectors can be controlled in primarily one of two ways. User’s desiring to create their own application should use the DexelaDetector API libraries. For a simple demonstration of basic detector control and image acquisition the SCap utility can be used. This manual will give descriptions and instructions for use of both these products. The manual also contains supplementary information that may be necessary that describes proper procedures for image corrections, data-sorting and installation instructions.

## Definitions and Abbreviations

Table 1: Definition and abbreviations

|  |  |
| --- | --- |
| **Application Programming Interface (API)** | A set of software libraries that can be used to create software applications. |
| **Camera**  **Link (CL)** | **Camera-Link communications protocol for transmitting images.** |
| **Dark Correction** | Synonym for Offset Correction [5.1]. |
| **Defect Map** | An image of the same size as the input image. It contains labels that describe the defect type of each pixel. Label '0' means no defect. |
| **Digital**  **Imaging Communications in Medicine (DICOM)** | Standard for handling, storing, printing, and transmitting information in medical imaging. |
| **Direct**  **Memory**  **Access (DMA)** | A feature of computerized systems that allows certain hardware subsystems to access main system memory independently of the central processing unit (CPU). |
| **Flood Calibration** | Synonym for Gain correction [5.3]. |
| **Flood Image** | **Flat field image. Typical recommended value is ~60-80% of the full-well** |
| **GigE** | **Gig**abit **E**thernet |
| **Network**  **Interface**  **Controller (NIC)** | **N**etwork **I**nterface **C**ontroller / **N**etwork **I**nterface **C**ard |
| **No.** | Number |
| **Read-out columns** | In the centre of each slice there are two side-by-side columns that are responsible for the read-out. The read-out pixels behave similarly to the other sensor pixels but have a lower gain. |
| **SCap** |  |
| **Software**  **Development**  **Kit** | **S**oftware **D**evelopment **K**it |
| **Slice** | Each full resolution sensor is composed of multiple sections that are read-out separately. This allows for readout parallelism and therefore speedup. |
| **SMV file** | An image file format that contains 512 bytes of header data. More detailed description is given in Appendix C – SMV file format. |

## Information and Warnings

*WARNING: IF USED AS PART OF A MEDICAL DEVICE, IMAGE AND FRAME BUFFER SHOULD BE CLEARED BEFORE ACQUIRING A NEW IMAGE.*

## System Requirements

Dexela software is compatible with computers that meet the following minimum requirements:

* Processor: Pentium 4 or above
* Memory: 4 GB
* Disk space: 1 GB
* Operating system: Windows XP (32-bit) or Windows 7 (32-bit and 64-bit)

**Note**: The system requirements above allow for the use of Dexela detectors but cannot guarantee any data rate. For detectors providing high data rates, we recommend using higher performance systems.

# Installation Instructions

Dexela detectors are available in two interfaces, CameraLink (CL) or Gigabit Ethernet (GigE).

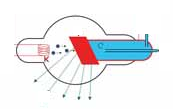
The following sections will give describe the equipment and steps required to install both interfaces.

Host computer

Detector

x-ray generator

Power supply unit



Camera Link or Ethernet

Sync cables

Data / control

DC power

*X-ray system (e.g. medical, industrial)*

Figure 1: Detector used in a Typical X-ray Imaging System

## Installing Peripherals

This section includes instructions for installing and configuring the:

* CameraLink Frame Grabber
* GigE Network Interface Card (NIC)

### Installing the CameraLink Frame Grabber

CameraLink detectors require the installation of a frame grabber in the host computer. PerkinElmer provides a complete solution offering Epix frame grabber cards in combination with these detectors. We strongly recommend that you use this bundle because the API provided by PerkinElmer for these detectors (DexelaDetectorAPI) is specifically built for them. The API is designed to handle all detector control, image acquisition, and correction functionality. Alternative frame grabbers may work, but they will involve more complex integration work (such as separate image corrections and performing detector control by serial commands).

PerkinElmer optionally supplies the Epix E4, E8 (dual cable) or EB1 (single cable) Camera Link card with its CameraLink detectors.



To install the frame grabber:

Plug the card into a spare PCIe x4, x8 or x16 slot. A PCIe x16 slot is the preferred option. On many PCs, at least one PCIe x16 slot is wired directly into DMA memory, which provides maximum speed.

### Installing the GigE Network Interface Card



A NIC is required to interface with GigE detectors. The recommended NIC is a PCI Express version of the I210-T1 series.

To install the NIC:

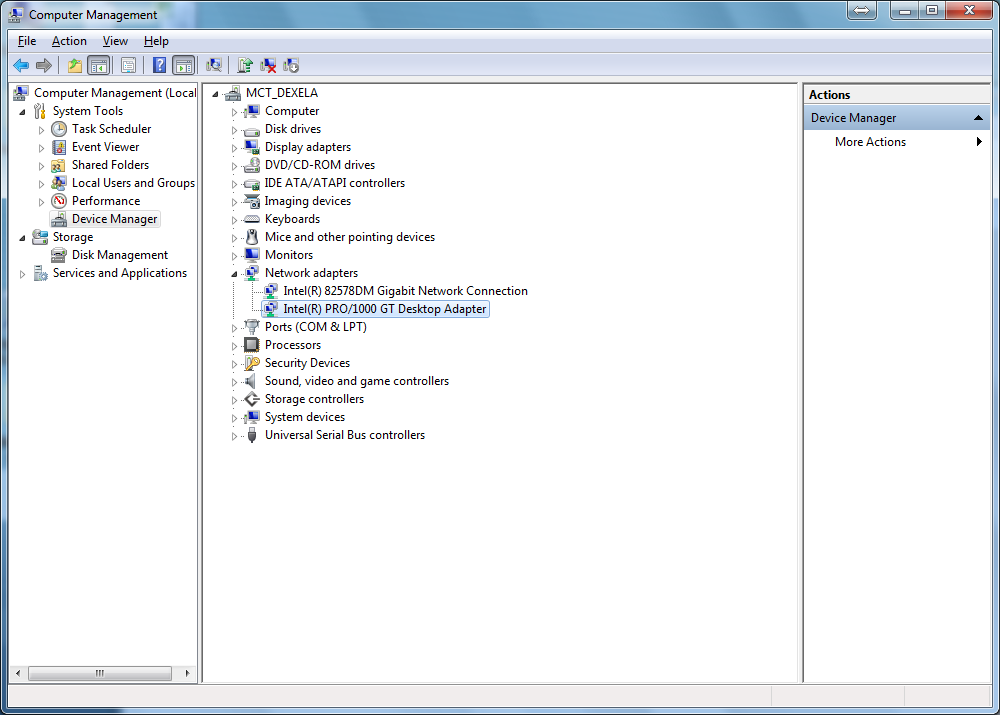
Plug the card into a spare PCIe slot on the motherboard. Since the data-rates of GigE are significantly lower than for CameraLink, there is no requirement on the amount of lanes required.,Pulgging this card into a PCIe x1 slot will provide sufficient bandwidth.

### Configuring the NIC

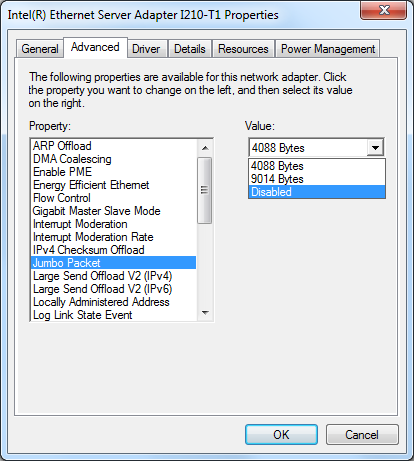
To get the best data rates, you need to configure the NIC for the best performance.

To configure the NIC:

1. Go to My Computer, right-click, and choose **Manage**.
2. Click **Device Manager**.



1. Click **Network Adapter**.
2. Right-click on **I210-T1**.
3. Click the **Advanced** tab.



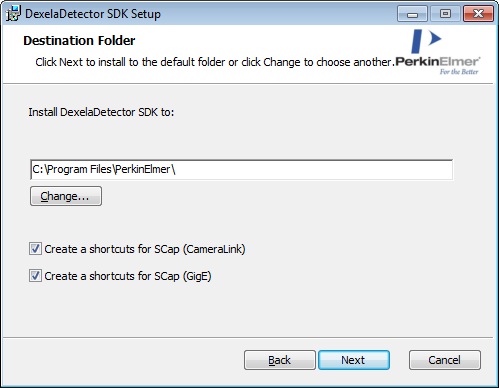
1. Enable **Jumbo Packet** (9014 or maximum allowable)
2. Increase the Receive Buffers to **2048** (or maximum allowable).
3. Click **OK**.

## Installing the SDK

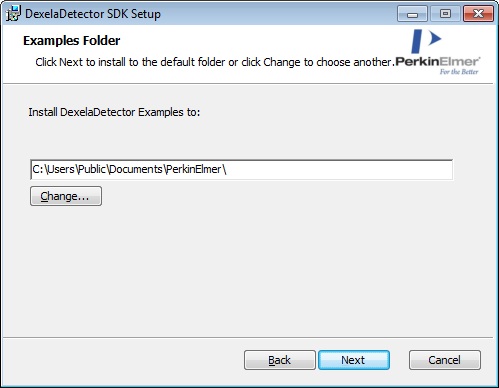
The DexelaDetector SDK comes with a Windows-based installer (.msi). The only pre-requisite for the installer is that the host system have version 4 of the Microsoft .NET Framework installed. To download and install Microsoft .NET Framework, go to <http://www.microsoft.com/en-us/download/details.aspx?id=17851>.

To install the SDK, run the installer, and follow the on-screen instructions. For more detailed information, see the following information.

The following figure shows the installer window that allows you to specify where to install the main components of the DexelaDetector SDK. These components include the library and header files for the DexelaDetector API as well as the SCap application. (For more information, see Section 3.0). The default location is shown in the figure. This window also allows you to specify which (if any) short-cuts to create for the SCap application.



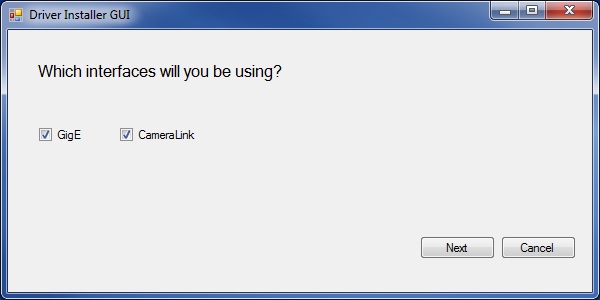
The following figure shows the installer window that allows you to specify where to save the source code examples. This requires a separate location because of Windows permissions on certain folders (for example, Program Files). By default the examples will be saved in the Public Documents folder (as shown below).



## Installing the CameraLink and GigE Drivers

When the SDK installation is finished, the installer launches the DriverInstallerGUI program. This program is used to install all of the necessary drivers for CameraLink and GigE detectors.

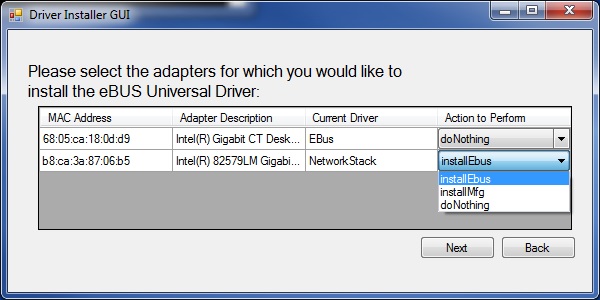
The following figure shows the initial screen for the DriverInstallerGUI. On this window, you can select the interfaces with which you will be working. Based on your selection, the appropriate drivers will be installed.



When the installation is finished, a window confirms the success or failure of the installation.

### Configuring the GigE Driver

The following Driver Installer GUI window shows all network interface cards that are detected on the host system. For each NIC, the current installed driver is shown, as well as three options for actions to perform: Install Ebus, Install Mfg, or do Nothing.



The Install Ebus option installs the EBus Universal Pro driver on the desired NIC. This is the recommended driver for use with Dexela detectors. **Note:** Installing this driver on a NIC will prevent it from accessing the Internet so make sure to select the correct NIC.

The Install Mfg option installs the manufacturer’s (that is, Intel) driver on the desired NIC. This driver can be used with the detector control and image acquisition API (DexelaDetector API), but it may not provide optimal performance (for example, image data is more likely to drop).

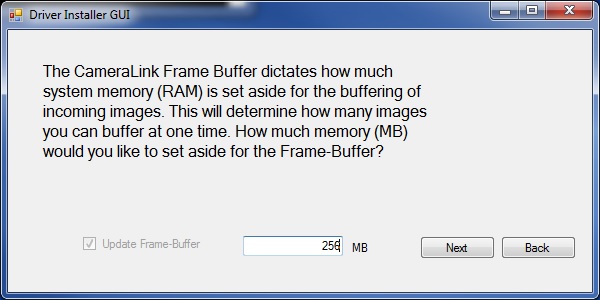
The do Nothing option leaves the desired NIC as is (that is, it will not install any new drivers).

### Configuring the CameraLink Frame-Buffer

As previously mentioned, the CameraLink versions of Dexela detectors are used with frame-grabbers from Epix Inc. These frame-grabbers require that a certain portion of system memory (RAM) be blocked off for use by the frame-grabber. This portion of memory is known as the frame-buffer, and it will not be accessible to any other programs or processes. The frame-buffer will buffer incoming images that are transmitted from the detector.

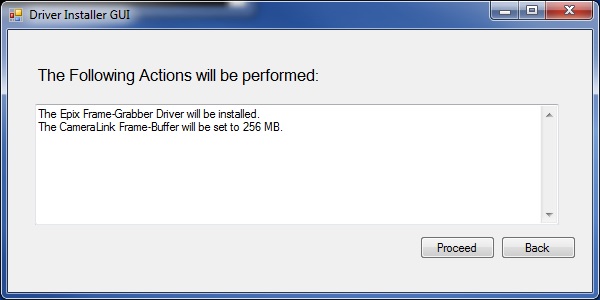
The recommended size of the frame-buffer can vary greatly depending on the nature of the application in which the detector is being used. For example, in systems that acquire a large amount of data quickly, a larger buffer may be required. At the very minimum, the frame-buffer should be large enough to fit a single image from the detector (although we recommend you allow for a few images).

If the DriverInstallerGUI is being run for the first time, and it determines that the frame-buffer has not been allocated, this step is necessary. Otherwise, you can skip this step by clearing the Update Frame-Buffer check box in the Driver Installer GUI window (see below).

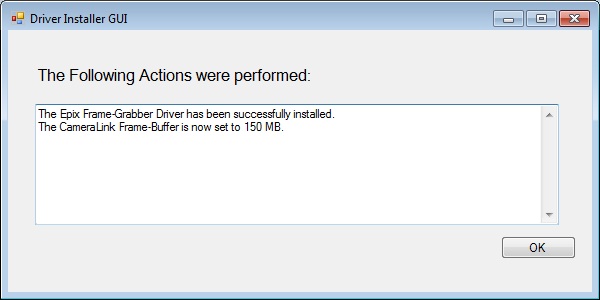


### Installing the Driver

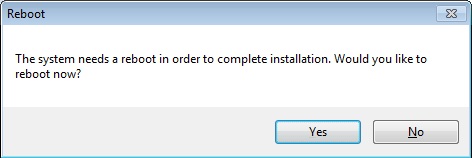
Once all settings have been configured, the Driver Installer GUI window displays a list of actions that it will perform. This gives you a chance to go back and make any desired changes.



Click **Proceed** to perform the listed actions. Once the actions are completed (some further input may be necessary), a similar list indicates the success or failure of each attempted action (see below).



At this point, the application informs you if a reboot is required (it may be required on certain actions), and it will give you the option to reboot immediately or later.



**Warning:** Clicking **Yes** immediately performs a system reboot. Any unsaved work may be lost.

# SCap

**SCap** is a utility for driving Dexela X-ray detectors. It can be used for demonstrating, testing, or evaluating the detector. This utility is installed as part of the DexelaDetector SDK.

The detector is a programmable data acquisition system with internal memory. It communicates with the host computer using a combined data and control link, which may be CameraLink or Gigabit Ethernet, depending on the detector model. The detector either contains one image sensor or more image sensors arranged in a tiled array.

Each detector is factory-calibrated for optimum performance. The calibration settings are held in a non-volatile memory in the detector.

## SCap User Interface

The SCap window is composed of four main areas:

**Menu Bar**: For accessing program functions using menus.

**Image Window**: For displaying and manipulating images. This window includes a menu for enhancing images and saving files.

**Control Panel**: Displays basic image statistics, and provides buttons for commonly used functions.

**Status Bar**: Displays detector hardware status.

Menu bar

Control panel

Status bar

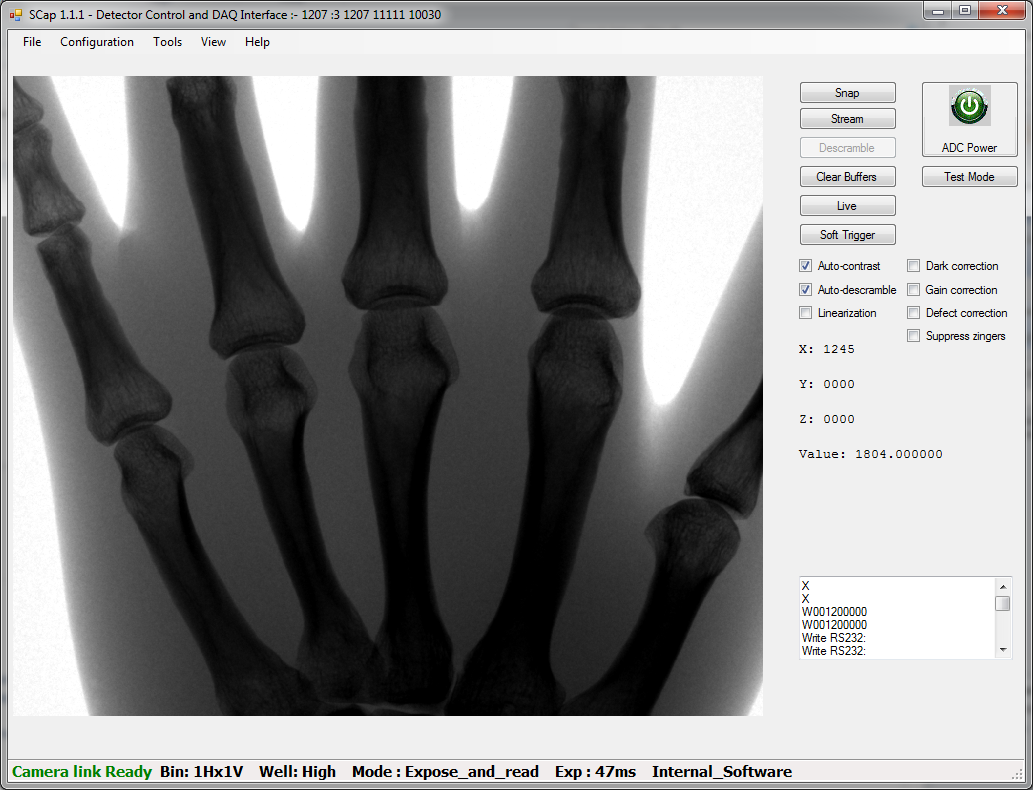


Image window

## Image File Formats

SCap handles images in two formats:

* 16-bit integer format containing 14-bit pixel data (0 to 16383)
* 32-bit floating point format

The 16-bit format is used for most images: captured images, dark images, and corrected images. Some images are stored in floating point format: flood calibration images and images converted into relative density values (see **Counts to Density** below). Both types can be stored.

For compatibility with image viewing and image processing software, menu options allow conversion of floating point images into 16-bit formats. For instance, **Save File 16 bit x 1000** multiplies the floating point image by 1000 before truncating to 16 bits.

## Image Window Functions

The image window has various functions associated with it.

### Left Mouse Button/Mouse Scroll Wheel Actions

The left mouse button and mouse scroll wheel can be used to access the following functions:

**Open a saved image:** Drag and drop a file(s) (SMV, DICOM, or TIFF format) into the image window to display it. Multiple images are alphabetically sorted by file name before being displayed as a stack.

**Pan image:** Hold the left mouse button to drag the image around the window.

**Brightness increase/decrease:** Hold down the **Shift** key and left mouse button while moving the mouse left or right.

**Contrast increase/decrease:** Hold down the **Shift** key and the left mouse button while moving the mouse forward or backward.

**Auto-contrast on region:** Hold down the **Alt** key and the left mouse button to select a region. New brightness and contrast will be applied to the entire image based on this region.

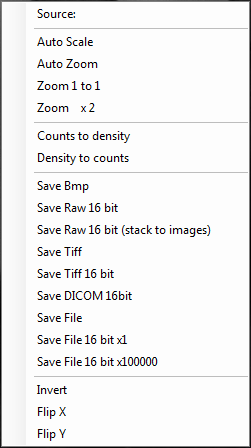
**Zoom:** Hold down the **Shift** key while turning the mouse scroll wheel. At higher levels of magnification, the pixel values will appear.

**Zoom on region:** Hold down the **Alt** and **Shift** keys, and hold down the left mouse button to select an area to zoom.

**Live ROI Histogram:** Hold down the **Ctrl** key, and hold down the left mouse button to select an area to histogram. A new window open with the live histogram. You can use the same button combination to select an area to zoom into in the histogram.

### RightMouse Button Context Menu

Click the right mouse button to access the following functions:

**Source:**  Shows the path and file name of an image when loaded from a file.

**Auto Scale:** Automatically adjusts the brightness and contrast of the image so that the grey levels are optimally presented.

**Auto Zoom:** Resizes the image so that it fits into the image window.

**Zoom 1 to 1:** Resizes the image into native resolution so that each image pixel is mapped onto a single display pixel.

**Zoom x2:** Resizes the image to a zoom by 2, based on the native resolution.

**Counts to density:** Converts the displayed pixel values into X-ray density values by creating an inverted log image. The pixel values are converted to floating point. The grey values are compressed, making the image easier to view without using brightness and contrast adjustments. The image is also inverted to give a film-like presentation. The blank count input should generally be set to the intensity (photon count) of the background.

**Density to counts:** Performs the inverse operation of **Counts to density.**

**Save BMP:** Saves the image in 16-bit bitmap format.

**Save Raw 16 bit:** Saves the image without header data as 16-bit unsigned integers.

**Save Raw 16 bit (stack to images):** Saves an image sequence as individual 16-bit unsigned integers images without header data.

**Save TIFF:** Saves the image in TIFF (Tagged Image File Format).

**Save TIFF 16 bit:** Saves the 32-bit floating point image as 16-bit integer TIFF, for example, 12.3456 –> 12

**Save DICOM 16 bit:** Saves the image in a 16-bit DICOM-compatible format, for example, 12.3456 –> 12

**Save File:** Saves the image in a raw image format (.SMV) with the following properties: image data as 32-bit float, Little-endian, header size = 512 bytes.

**Save File 16 bit x 1:** Saves the image as 16-bit SMV, for example, 12.3456 –> 12

**Save File 16 bit x 100000:** Multiplies the 32-bit floating point image by 100,000, and saves it as 16-bit SMV, for example, 0.003456 –> 3456.00

**Invert:** Inverts the grey scale look-up table used for final display. The original data is not affected.

**Flip X:** Flips the image X axis used for final display. The original data is not affected.

**Flip Y:** Flips the image Y axis used for final display. The original data is not affected.

## Control Panel Functions

This section describes the functionality of all controls that can be found in SCap’s control panel (see figure above).

### Control Panel Button Functions

The following describes the functionality of the buttons that are found in the control pane:

**Snap:** Captures and displays a single image.

**Stream:** Captures and displays images continuously without pausing between images. The image update rate will depend on the exposure time selected in the Settings menu. Clicking **Stop** on the same button freezes the last image.

**Clear Buffers:** Clears the PC frame buffer memory, and sets the image display to zero pixel values (black).

**Test Mode: Generates a** grey scale test pattern instead of the sensor image. This can be used to verify that the data link is operating. The same button changes to **Normal**. Click to exit the test mode.

**Descramble:** If the Auto-descramble check-box is not selected, the image is received in an interlaced format due to the design of the CMOS sensor. Pressing the Descramble button sorts the image into the correct format for display.

**ADC Power:** Turns power on and off to the detector electronics. Green indicates power is on; amber indicates power is off.

**Live:** Puts the detector into **Live** mode. Image capture is triggered by a signal received at the Sync In connector or by a software trigger. To exit Live mode press the button (which should now say **UnLive**) again.

**Soft Trigger:** When the program is in **Live** mode, pressing this button sends a software trigger signal.

### *Check Box Functions*

The following describes the effect of selecting each of the check-boxes in the control panel:

**Auto-Contrast**: Automatically adjusts the brightness and contrast of each image so that the grey levels are expanded to the range of the display.

**Auto-Descramble**: Automatically converts the raw interleaved image from the detector into a legible format. *This option is selected by default.*

**Linearization**: Applies a software linearization to each image captured.

**Dark Correction**: Automatically performs dark correction on each captured image. The correction is only performed if there is a dark image available.

**Gain Correction**: Automatically performs gain correction (flood field correction) on each captured image. The correction will only be performed if a flood image is available.

**Defect Correction**: Applies defect corrections as specified in the loaded defect map (see Section 3.9 for more information).

**Suppress Zingers**:Removes isolated white pixels that may appear from direct conversion of X-ray photons in the CMOS sensor.

### *Numerical Data Display*

The following describes the labels (that are automatically updated) that can be found in the control panel.

**X**: x coordinate of the cursor on the displayed image.

**Y**: y coordinate of the cursor on the displayed image.

**Z**: Index number of image when a sequence is displayed. First image = 0.

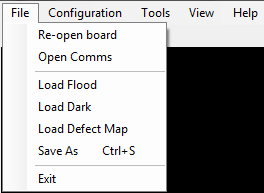
**Value**: Intensity value of the pixel at the cursor position.

## Menu bar functions

The following sections describe the functionalities that can be accessed through the various menus that are found in SCap’s menu-bar (see figure above).

### *File Menu*

The following functionalities can be accessed through the file menu.

**Re-open Board:** Resets the interface to the detector and the Camera Link card, if installed.

**Open Comms**: Opens the serial communications channel. Sets parameters like baud-rate, parity bits, stop bits, and so on.

**Load Flood:** Loads a user-selectable flood image for image correction. Several flood images can be stored on disk (for different energies or source-image distances) and loaded individually, as required.

**Load Dark:** Loads a user-selectable dark image for Offset Correction.

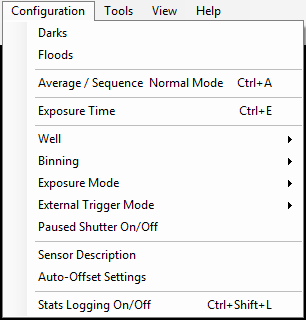
**Load Defect Map:** Loads a user-selectable defect map image for Defect Correction.

**Save As:** Saves the displayed image in TIFF 16-bit format. Other formats can be saved using the right-mouse button context menu while the cursor is over the image.

**Exit:** Exits the SCap program.

### *Configuration Menu*

The following functionalities can be accessed through the Configuration menu.

**Darks:** Enters the dark image acquisition settings: exposure time, number of averaged images, and optionally, the path for the dark image file. If no path is provided, the image is stored in the current working folder (usually where **scap.exe** is located). A new file is created each time a dark image is acquired; old files are not automatically overwritten.

**Floods:** Enters the flood image acquisition settings: exposure time, number of averaged images, and optionally, the path for the flood image file. If no path is provided, the image is stored in the current working folder (normally where **scap.exe** is located). A new file is created each time a flood image is acquired; old files are not automatically overwritten.

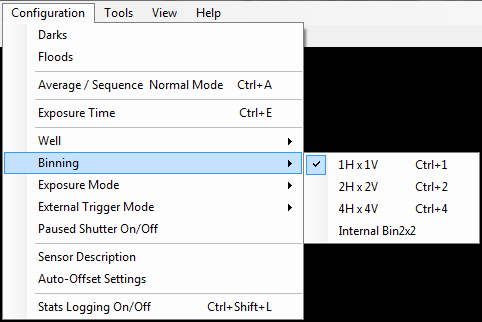
**Average/Sequence Normal Mode:** Enters settings for frame averaging and image sequence for Normal exposure mode: exposure time per frame and number of frames.

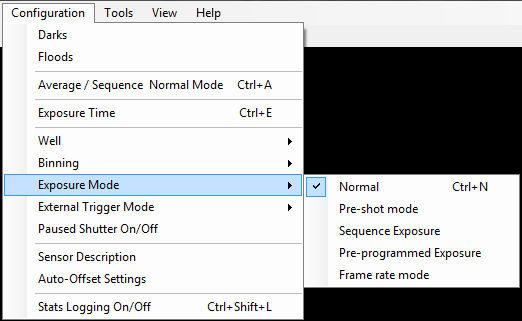
**Exposure time:** Enters exposure time per frame in milliseconds (normally a minimum of 37ms for unbinned 2923, 1512, 2315).

***Note****:* The time of the actual exposure window is defined by the total amount of the set time tinput subtracted by the time treadout, which the detector needs to read out one image frame. In other words, the exposure time in SCap already includes the detector read-out.

**Well:** Selects High Full Well or Low Full Well mode. High Full Well is used for high dynamic range operation in static imaging (single frame). Low Full Well mode is used for dynamic imaging such as fluoroscopy where a low dose per frae is applied.

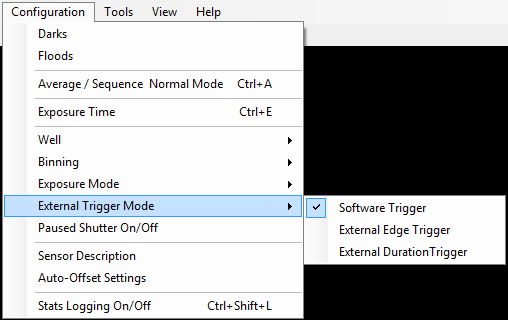
**Note**: For more information regarding the detector full well modes, refer to the manual that came with your detector.

**Binning:** Selects one of 3 different modes of on-sensor pixel binning. Depending on the detector model, an internal 2x2 sensor binning (where binning is done in the FPGA) can also be selected, which improves noise performance.

**Exposure mode:** 5 modes are available:

* In **Normal** mode, a single frame is acquired after each trigger (**Snap** button, Software Trigger, or External Trigger).
* The **Pre-shot** mode sets the detector into an external edge mode sequence of two exposures. After selecting this option, you can choose the exposure time for each exposure. The detector should run in **Live** mode before setting this mode. When the exposure is finished the detector returns to **Normal** mode for the next exposure.
* In **Sequence Exposure** mode, a series of exposures is captured with no gap between frames. The number of frames and exposure time are set in a dialog box that appears when you click this option.
* **Pre-programmed Exposure** mode is similar to pre-shot mode, but here a sequence can consist of 2- 4 frames. Each frame has an individually programmable exposure time. The number of exposures and the exposure time for each frame is set in a dialog box that appears when you click this option. For example, this option can be used for making composite exposures where the subject has a high density ratio.
* In **Frame Rate** mode, a series of exposures is captured with a configurable additional gap (minimum current detector readout-time) between frames. The number of frames, gap time, and exposure time are set in a dialog box that appears when you click this option.

**External Trigger Mode: 3 different external trigger modes are available:**

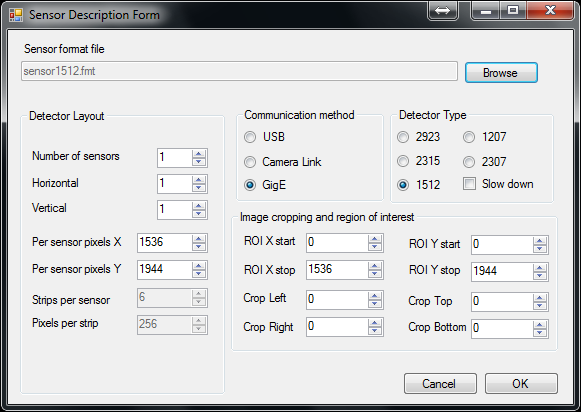
**Software Trigger:** The detector is triggered by software. In principal, the image data that is acquired using the SCap control elements use this trigger mode (for example, **Snap** and **Stream** buttons as well as **Darks**, **Floods** and **Average** / **Sequence** collection modes).

**External Edge Trigger:** Exposures are initiated by signalling the sync in port. The exposure time is the one set within SCap (**Configuration > Exposure Time**). The signal starts the exposure in the same way the software trigger does.

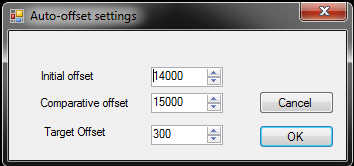
**External Duration Trigger:** In this mode, the duration of the sync in signal is used to define the exposure time of the sensor.

**Note**: For more information about the Detector Trigger Modes, refer to the manual that came with your detector.

**Sensor Description *(advanced setup only): The sensor Description form can be accessed by clicking Sensor Description in the Configuration menu. This Form can be used to configure the detector description paramters.***

A *sensor format (.FMT) file* is associated with each Camera Link detector type. This file contains data that defines the hardware and firmware properties of the detector when it is using **Camera Link**. For **GigE**, this file is not used. Select the **GigE** option and the type of detector being used.

The basic properties of the detector are displayed in the dialog box. The displayed image size and the size of the Region of Interest (ROI) can be edited. Cropping is applied only if the **Defect Correction** box has been selected in the Control Panel.

**Auto-offset Settings *(advanced setup only)*: *The Auto-Offset settings form can be accessed by clicking Auto-Offset settings in the Configuration menu. This allows you to setup the ADC offset offsets calibration (found in the Tools menu).***

This allows you to set a **Target Offset** (normally 300) that is the digital value representing black or no X-rays. After ADC Offset calibration, the black value will be around 300. **Initial Offset** and **Comparative Offset** are set to default values that should not be changed.

**Stats Logging On/Off *(advanced setup only);*** Starts a log file that records the results of **Image Properties** (see the *View menu*). Control messages passing to and from the detector are recorded. You can edit the path name of the log file by editing the .XML file **scap.exe.config** located in the SCap executable folder. The XML attribute that should be edited is called initializeData and can be found in the listeners tag with the name MyListener.

### *Tools Menu*

The following functionalities can be accessed through the tools menu.

**Snap:** Acquires a single frame. This function is also provided by the **Snap** button.

**Collect Dark:** Begins acquiring and averaging dark frames to create a stored dark image. Parameters can be set in **Configuration** –**Darks**.

**Collect Flood:** Begins acquiring and averaging flood frames to create a stored flood map image. The flood image is corrected with the previously stored dark image and subsequently normalised and converted to a floating point map containing values normalised to 1.0. Parameters can be set in **Configuration** – **Floods.**

**Collect Average:** Begins acquiring and averaging frames. The result is then displayed in the image window. It gets corrected according to the check box options selected in the Control Panel. Parameters can be set in **Configuration** – **Average/Sequence.**

**Subtract Dark:** Subtracts the currently loaded dark image from the displayed image.

**Flood Correct:** Multiplies the displayed image by the currently loaded flood image.

**Full Correction:** Applies a combined dark subtraction and flood correction to the displayed image.

**Defect Correction:** Applies a defect correction to the displayed image according to the currently loaded defect map: See **File** – **Load Defect Map**.

**Suppress Zingers:** Removes isolated white pixels that may appear due to hardware characteristics.

**HDR:** If SCap has a stack of images with the same image dimensions in its viewing pane, this option combine the images into one image using PerkinElmer’s HDR library.

**Sharpen:** The current image plane is sharpened using a simple 3x3 sharpening convolution.

**Interpolate:** SCap interpolates when zooming in or out of the image. This option allows you to choose between Bi-linear, Bi-cubic interpolation methods, or no interpolation.

**Serial Comms *(advanced setup only***)**:** Allows you to send serial command strings to the detector.

**Note**: This is an advanced function and can lead to the detector being made inoperable.

**Line Profile:** Displays the profile of the selected horizontal line and optional following lines.

**Load Sensor Config (Advanced setup only):** Loads the sensor configuration file into the detector. This only configures the dark offsets and is deprecated for normal use.

**Make flood from image:** Stores the currently displayed image as the default flood image.

**Make dark from image:** Stores the currently displayed image as the default dark image.

**Make average form stack:** The stack of images in the viewer will be replaced by the mean average of the intensity.

**Add constant:** You can add an arbitrary constant to the pixel values.

**Subtract Image:** You will be asked to select an image that will be subtracted from the currently displayed image. The result is displayed subsequently.

**Get Std Dev:** Reports the standard deviation of the image.

**Refresh Serial Number:** Retrieves the on-board settings for serial number, model number, and firmware version, which are displayed in the title bar of SCap.

**Firmware Upload: Some detectors have the ability to have their firmware uploaded by SCap.**

**Note**: This is an advanced function and can lead to the detector being made inoperable.

**Calibrate ADC Offsets***:* In the detector, each CMOS sensor is divided into slices, each having a separate black level or offset. This function performs an automatic balancing of the offsets when the sensor is not illuminated by X-rays so that the strips are displayed at approximately the same black level. Do not perform this procedure unless advised to do so by a technical support representative.

### *SCap_Menu_Configuration_ViewView Menu*

The following functionalities can be accessed through the View menu.

**Current Dark:** Displays the currently loaded dark image.

**Current Flood:** Displays the currently loaded flood correction map. This map is normalised to 1.0; brighter regions of the map (>1.0) correspond to regions of the detector area that have a lower x-ray-to-light conversion factor.

**Defect Map:** Displays the currently loaded defect map indicating the position and classification of each defective pixel in the image sensor.

**Image Properties:** Displays information about the currently displayed image, with file names and paths of currently loaded dark-, flood-, and defect images.

The numbers are the individual sensor associated pixel averages for the six vertical strips. This can be used to check the uniformity between strips and between sensors within a dark image.

### *SCap_Menu_HelpHelp Menu*

The following functionalities can be accessed through the Help menu.

**About:** Displays version information for SCap.

## Examples of Imaging Tasks

SCap provides a collection of functions that can be used to perform all of the steps necessary to produce a fully corrected digital image using a wide choice of techniques. The following are examples of typical tasks required for digital imaging.

1. Calibrating the Detector

The following example sequence calibrates the detector

1. ***Launch SCap.***   
   A dialog box asks if the default configuration file should be loaded. Click **Yes** .
2. ***Set Configuration Menu Settings***:
   1. In the **Configuration** menu, select **High Full Well** and **1x1 binning**.
   2. Set **Exposure Mode** to **Normal**, and **External** **Trigger** **Mode** to **Software** **Trigger**.
3. ***Check Detector Operation****:*
   1. Press **Snap** and verify that a dark image is captured.
   2. Move the cursor around the image window and observe the pixel **Value** displayed in the Control Panel. The pixel values should be in the range 300 ± 50.
4. ***Determine Approximate Exposure Time****:*
   1. Set the exposure time by selecting **Configuration – Exposure Time**. Start with 100ms.
   2. Start the x-ray beam then immediately click **Snap** orControl-X**.**
   3. Examine the image that appears, and note the brightness level in **Value** as the cursor is passed over the image. This should be in the region 14,000 – 16,000. If it is above or below these limits, reduce or increase the **Exposure Time** accordingly. If the minimum exposure time results in values outside of this range, reduce the X-ray tube current.
   4. Record the value of exposure time that gives the required image brightness.
5. ***Prepare the X-ray Source:***
   1. Set the required kV and mA on the X-ray generator.
   2. Set the beam time to the exposure time determined in step 5 plus two seconds.
6. ***Configure Calibration Settings****:* 
   1. Select **Configuration – Darks**, set the exposure time as noted in step 5, and set **No of Frames** to 5.
   2. Select **Configuration – Floods**, set the exposure time as noted in step 5, and set **No of Frames** to 5.
7. ***Create Dark Image***: With X-rays off, select **Tools – Collect Dark**.   
   The dark image will be created. This process will take a few seconds.
8. ***Create Flood Image***: Start the X-ray beam then immediately select **Tools – Collect Flood.**   
   The flood image will be exposed during which a progress bar is displayed in the status bar. If the beam terminates while the progress bar is red, set a longer beam time or alternatively reduce the **No of Frames** in **Configuration – Floods**. A second progress bar indicates that the software is calculating the image, and the detector is no longer acquiring frames.
9. ***Take a Blank Test Image***: Check all the boxes in the Control Panel. Start the x-ray beam with the previous settings used in the Flood Image, and immediately press **Snap**.   
   After a short time, a processed image appears in the image window. This should appear as a uniform noisy grey image without obvious structures or artefacts; this indicates that the Detector Calibration images are correct.
10. Capturing and Saving a Single X-ray Image with Manual Triggering

The following example sequence generates a fully corrected image, assuming that the calibration images have been acquired as in A).

1. ***Launch SCap***:   
   If a dialog box asks if the default configuration file should be loaded, click **Yes** .
2. ***Configuration Menu Settings***: Set **Exposure Mode** to **Normal** and **External** **Trigger** **Mode** to **Software** **Trigger**. Select all of the check boxes in the Control Panel.
3. ***Adjusting X-ray Dose****:* The X-ray generator is programmed with the kV setting appropriate for the object being imaged. A trial image is then taken in order to determine the optimal exposure settings for the object. While X-rays are on, acquire a single raw image by pressing **Snap.** Move the cursor around the image, concentrating on the areas of interest, and note the pixel **Value** displayed in the Control Panel. The pixel values should be in the range 12,000 - 15,000 in the lighter, lower density regions. If the values are significantly outside this range, the X-ray exposure time or tube mA can be scaled up or down accordingly.
4. ***X-ray Image:***  Change **Exposure Time** and/or tube mA that resulted out of the prior step. Set the X-ray beam duration to the detector exposure time plus 2 seconds. Start the X-ray beam, then immediately click **Snap** orControl-X**.**  A corrected X-ray image should appear. Use the left-mouse-button functions to adjust brightness, contrast and zoom as needed.
5. ***Save the Image***: Right-click on the image, and save the image in the chosen file format and location.
6. Capturing and Saving a Sequence of Images with External Triggering

The following sequence generates a fully corrected image, assuming that the calibration images have been acquired as in A).

1. ***Configuration Menu Settings***: Set **Exposure Mode** to **Normal**, and **External** **Trigger** **Mode** to **Software Trigger.** Select all of the check boxes in the Control Panel.
2. ***Adjusting X-ray Dose****:* Acquire an image as in B)3, and determine the exposure time and tube mA accordingly.
3. ***Configuration Menu Settings***:
   1. Select **Average/Sequence**, and browse to a destination folder for the images. In the same dialog box, enter the exposure time and the number of images in the sequence.
   2. Set **Exposure Mode** to **Sequence Exposure**.
   3. Set External **Trigger** **Mode** to **External Edge** **Trigger** or **External Duration** **Trigger,** depending on the type of interface to the X-ray generator (For more information, refer to the manual that came with your detector.). Check all the boxes in the Control Panel.
4. ***X-ray Image Sequence:*** Press **Live**. Expose the X-ray beam for as many images as required. On each shot, the image will be captured. It is not saved automatically, so click the right mouse button and save the images as required.

## Troubleshooting

The following describes some basic trouble-shooting scenarios and potential steps to take to address them:

* ***SCap Loses Communication with the Detector***

If the detector is powered off while SCap is running, communication may be lost. Exit SCap, and restart it. If the detector does not respond to commands, try power cycling it.

* ***No Raw Image***

***Check Communication:*** Press **Test Mode**, then **Snap**. A grey scale image should appear. If not, check all cable connections and power supply. Re-boot the PC, and launch SCap.

***Check for a Dark Image***: Select **Auto-contrast** and **Auto-descramble** and clear all other check boxes. Press **Snap** while in normal mode. A grey patterned image should appear containing grey values around 300. If the value is 0, press the calibrate ADC offsets menu option (**Tools > Calibrate ADC offsets**), wait for a few seconds, then press **Snap** again.

* ***No image after corrections***

***Check Dark Image***: Select **View** – **Current Dark**. A grey, patterned image should appear with pixel values around 300 – 500.

***Check Flood Image***: Select **View** – **Current Flood**. A mottled shaded image should appear with pixel values around 0.7 – 1.2.

* ***Poor X-ray Image Quality***

This may be caused by incorrect energy or current in the X-ray generator, but consider the following other factors:

***Dark Image Mismatch***: The dark image used should have approximately the same exposure time as the X-ray image and should be taken with the same binning and well settings.

***Flood Image Mismatch***: The flood image should match the parameters of the X-ray image: same beam geometry (source-image distance), same X-ray settings (energy, focal spot size, and filtration). It should also be taken with the same binning and well settings.

***Underexposure***: Ideally, images should have grey values corresponding to the dynamic range of the detector (0 – 16,383). Where the pixel values are low, image noise will be correspondingly worse.

* ***SCap Hangs, or Reports Missing System Files***

The.Net Framework 2 (or higher) and Visual C++ 2008 Redistribution must be installed on the PC. These are present on most PCs where other software has been installed. If they are missing, they can be downloaded free from the Microsoft® web site.

# Sorting Schemes

The Dexela detectors are high-speed, read-out detectors. To achieve this, they are addressed in sections (sometimes called *taps*) that are transferred to the PC in parallel. Where GigE is the interface,e the detectors are delivered to the GigE outbound buffer in parallel, which has the same result at the PC.

This parallel read-out scheme means that the pixels from the image formed in the camera are not delivered sequentially. This means that a raw image appears jumbled and unintelligible. This section explains the various interlace schemes for the different Dexela detectors.

## Sorting 1512

The 1512 and 1207 detectors consist of a single CMOS chip that for the sake of parallel readout consists of six independent cameras. These six cameras are read out at the same time and delivered to the GigE or Camera Link cards at the same time. Each unbinned camera is 256 pixels wide by   
1944 (864 for 1207) pixels in height. The sorting of the Dexela 1512 and 1207 detectors is by line only and consists of the first pixel from each slice followed by the second pixel from each slice. This is effectively a six-tap camera, but it is unusual to find a camera link or GigE card capable of reconstructing the image from so many taps. You need to descramble the pixels either using DexelaDetectorAPI or by writing code to rearrange the pixels.

The following diagram shows an example of the data coming from the detector. In the case of 1512 and 1207 detectors, the pixels are grouped in 6-pixel blocks (for example, Block 0, Block 1 below). Each block consists of a pixel from each of the six slices in the sensor. In this case, the slices correspond to the position of each pixel within the block (block position). Each pixel from an unbinned block will be 256 pixels away from its neighbour.



Counting blocks and pixel position in the block (from 0) the Sorting formula is:

**Block position \* 256 + Block number**

**Note:** Running the detector in a binning mode will affect the width and height of pixel blocks.

See Appendix B – Incoming Data Sorting Code Example for example code.

## Sorting 2315

The 2315 and 2307 detectors consist of two sensors, both of which are readout at the same time. This results in blocks of 12 pixels that need sorting. The first six pixels within the block correspond to pixels from each of the slices in one sensor, while the next six pixels come from each of the slices in the other sensor. In this case, the first pixel is from the centre of the image rather than the edge. Depending on how the resulting image should be orientated, the following is an example of how to reorder the pixels.



## Sorting 2923

The 2923 consists of four sensors read out in parallel. The four sensors form a matrix of 3072 pixels (0 – 3071) in 3088 (0 – 3087) rows, and the readout results in blocks of 24 pixels. This time, the first 12 pixels are from the top of the image (one from each of the slices in the top two sensors) while the last 12 are from the very bottom (again, from each of the slices in the bottom two sensors). The image fills in from the top to the middle and from the bottom to the middle. Finally, because of the rotation of the sensors in the detector, the pixels from the bottom are backwards to those at the top.

Block

0

Block position

0

1

2

3

4

5

6

7

8

9

10

11

Pixel position

1

257

513

769

1025

1281

1536

1792

2048

2304

2560

2816

Row position

0

Block

0

Block position

12

13

14

15

16

17

18

19

20

21

22

23

Pixel position

3072

2816

2560

2304

2048

1792

1536

1280

1024

768

512

256

Row position

3087

Block

1

Block position

0

1

2

3

4

5

6

7

8

9

10

11

Pixel position

2

258

514

770

1026

1282

1537

1793

2049

2305

2561

2817

Row position

1

Block

1

Block position

12

13

14

15

16

17

18

19

20

21

22

23

Pixel position

3071

2815

2559

2303

2047

1791

1535

1279

1023

767

511

255

Row position

3086

# Image Corrections

As with any X-ray detector, raw images acquired directly from Dexela detectors need some processing (or correcting) before being ready for use. The following sections describe the various corrections that can (and should) be applied to produce optimal images. The manual provides an overview of each type of issue and how it can be corrected. It also provides links to the appropriate sections of the DexelaDetector API description.

## Offset Correction

The dark image is an average of several (for example, 16) dark frames. Offset correction removes static offsets between pixels so that a dark image is presented as an approximately uniform grey value. The Offset Correction consists of subtracting the dark image from the input image. The dark image should be recaptured periodically. During the warming-up period of the system, the dark current of the pixels may change considerably.

## Linearity Correction

Detector pixels have a slightly non-linear response to light, which corresponds to about 2% deviation from the linear response at mid-scale. To ensure that the flood calibration works accurately over the dynamic range of the detector, both the raw image and the flood image should be corrected for non-linearity. This correction can occur in firmware (for those detectors that support this functionality) or in software (see Section 6.3.2)

Figure 1. Linearity Correction

## Gain Correction

The flood calibration image is a floating-point multiplication matrix normalised to 1. The calibration image is created from the average of several (for example, 16) flood images, each of them also dark corrected. Multiplying by the flood calibration image has the effect of reversing the variation in X-ray conversion gain between pixels or regions so that a corrected open field X-ray image will have near-constant grey levels across the field of view.

## Geometric Correction

For those detector models that use multiple sensors, the Geometric Correction has to be applied. It corrects the image data in terms of the distortion that occurs at the borders of adjacent sensors. The distance of the gaps between these borders are assumed to physically measure up to one pixel. The Geometric Correction fills these gaps with interpolated image data. This correction occurs in the defect correction function provided by the DexelaDetectorAPI (see Section 6.3.5).

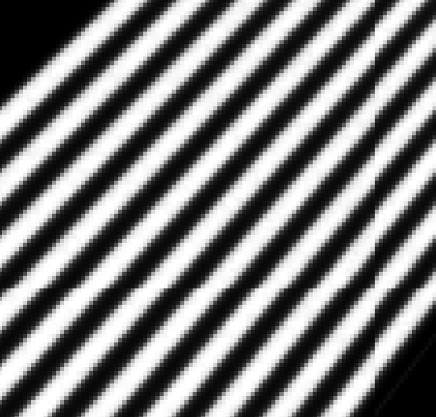
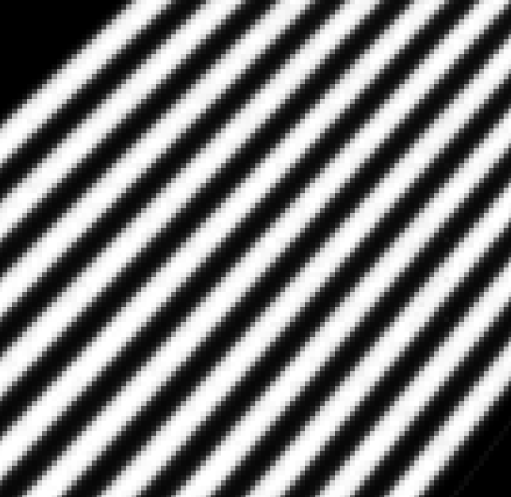


Figure 2. Geometric correction of a 2923 detector, which has 2x2 sensors mounted. The image show a magnification of the centre section where four sensors meet: a) original image and b) corrected image

## Defect Correction

As with any image sensor, the detectors may show some defective lines or pixels. Defective lines or pixels are defined as having a response that deviates by a specified amount from the ideal. To make the behaviour of all pixels appear ideal, Defect Correction is performed*.* It uses a Defect Map that is an image that contains labels for the pixels that need to be corrected or adjusted. On the final processed image, the pixels that are tagged on this map are replaced by interpolated values in order to improve the image presentation. In the DexelaDetectorAPI, this routine is handled by the defect corrections described in Section 6.3.5.

The Defect map is stored as an SMV image (see Appendix C – SMV file format for more information on this file format). The defective pixels are represented by label values, which indicate the type of correction algorithm to be applied.

The following section provides a description of the label values.

Label 1:

Label 1 is used for pixels that are considered *dead* pixels. These pixels generally do not contain useful information so they are replaced by a corrected pixel. The value of this pixel is determined by evaluating the surrounding healthy neighbours. Label 1 can be set for single defective pixels, clusters, rows, and columns.

Labels 2 to 10:

These labels are set to pixels that have some information but are not considered totally *dead*. Certain pixels might function at slightly different gain compared to normal pixels. These pixels are given label values 2 to 10’ depending on how distant they are from the mean of healthy pixels. During correction, they are compared with the healthy neighbours and are *gain* corrected based on the differences. The amount of contribution of the defective pixel is determined by the label value (2 to 10). **Note**: Labels 2-10 are legacy values and will not be used in future defect maps. Instead, all pixels previously marked as 2-10 would be considered as label 1.

Label 11:

Label 11 is assigned to clusters of defective pixels. The correction is done using a large area search in the neighbourhood of the current cluster. The resulting method is quite process intensive and the inclusion of it is often avoided. Generally, label 1 is used, which is a much faster correction method.

Label 12:

The pixels in the read-out columns have the same behaviour as the surrounding columns but function at lower gain. In principal, these columns will get corrected by the standard offset and gain correction process. As a precautionary measure, label 12 is assigned to these columns to perform a local adjustment with respect to the surrounding neighbour pixels. Apart from the read-out columns, this label can be assigned to any column that is healthy but has a slightly different gain compared to the normal pixels.

Labels 16 and 17:

Extreme changes in high frequency contrast in the image may induce artefacts caused by cross-talk in certain sensor types. Labels 16 and 17 address this issue by providing the option of applying a running convolution process to counter the effect. The labels are assigned to the first column of each slice. For standard imaging situations, this correction should not be necessary. Further information can be requested from the support team.

**Note***:* Certain labels are automatically included in each Defect Map. For example, label 12 is set to the read channels that are located in the centre of each slice.

# DexelaDetector API

DexelaDetectorAPI is an API that provides object based C++ libraries for detector control and image acquisition with PerkinElmer CMOS X-ray detectors. This allows you to add detector control and image acquisition capabilities to your own application software. The library also provides functions for performing gain and defect corrections on acquired images. It is available for 32- and 64-bit operating systems.

The following sections include an overview of the classes provided by the API along with some basic information on their uses. For detailed descriptions of all class members and methods (and associated parameters), refer to the API help manuals that come with the SDK.

## DexelaDetector Class

The DexelaDetector class contains the majority of the functionality for communicating with Dexela X-Ray detectors. This class is used for opening a connection to an actual detector. It can then be used for configuring settings (for example, binning, exposure time, exposure mode, and so on), triggering and acquiring images from the detector.

This class is contained in its own DLL (DexelaDetector.dll) and has some dependencies (for certain methods) on the DexImage, InternalUtils, and DexelaException DLLs.

The following sections provide a brief overview of how to use the class. For a detailed list and description of all class methods, refer to the DexelaDetector API help file (chm).

### Detector Connectivity

The DexelaDetector class can be initialized in one of two ways. The easiest way is to use the BusScanner class (see Section 6.2) to first enumerate all Dexela X-ray detectors that are connected to the system. The user can then choose from one of the detectors that are discovered and use the associated DevInfo structure to initialize a DexelaDetector instance.

Alternatively, if you already know which detector you want to connect to and the associated parameters (for example, unit number for CL detectors and IP address for GigE detectors), you can skip this step and directly initialize a DexelaDetector instance (without the use of the BusScanner class).

Once an instance of the class has been created, the connection to the detector can be opened with a call to OpenBoard. This method has two variants: if no parameter is passed in, it will use a default number of image buffers (for CL, this corresponds to the entire frame-buffer; for GigE, it is 20 images). Alternatively, you can pass in the desired number of image buffers to allocate. The API will attempt to allocate the number of buffers and will throw an exception if it is unable to (for example, if the allocated frame-buffer does not fit the desired number of images).

Once the connection to the detector is no longer needed, a call to CloseBoard should be made to ensure that resources are freed up.

### Detector Settings

Detector settings (for example, binning mode, exposure time, trigger source, and so on) can be easily written-to and read-back from the detector. Each setting should have its own get and set method (for example, GetBinningMode, SetBinningMode), which can be invoked from the DexelaDetector class instance.

### Detector Triggering

Dexela detectors can be triggered either with an external source or internally by software. For detailed information on timing and external trigger modes, refer to the manual that came with the detector.

#### Software Trigger Mode

The detector can be put into software trigger mode by setting the trigger source (for example, by calling the SetTriggerSource method) to Internal\_Software. When in this mode, the detector will send an image when it receives a software trigger signal, which can be done by calling the SoftwareTrigger method.

#### Pulse Generator Mode

The detector also has a mode called Pulse Generator Mode in which it is possible to send a continuous stream of software triggers at a specified frequency. The detector can be put into this mode by first setting the trigger source to Internal\_Software then calling the EnablePulseGeneratoreMode method.

If a specific frequency is desired for the pulses, you can specify it; otherwise, this parameter can be omitted and the detector will use a frequency that guarantees a continuous stream of images (that is, the software triggers will be coming fast enough to guarantee that no scrub readouts are required between image transmissions).

When the pulse generator mode is enabled, software triggering of the detector is disabled (that is, a call to the SoftwareTrigger method will have no effect).

The actual stream of trigger signals can be started or stopped by calling the ToggleGenerator method when the Pulse Generator mode is enabled.

### Image Acquisition

The API has several different methods of actually acquiring images from the detector. These methods are described in the following sections.

#### Image Snap

If acquiring only a single frame at a time, the simplest thing to do is to snap the image into the desired buffer. This can be done by calling the Snap method. You can specify the buffer to snap the image into and also the amount of time to wait (that is, timeout) for an image before returning an error. This method will also automatically trigger the detector if the trigger source is set to Internal\_Software.

#### Live Mode

If multiple images are to be captured in a short interval of time, then the snap mode may not suffice. In this case, the detector can be put into live mode by calling the GoLiveSeq method.

In this mode, the detector will automatically acquire images into the frame-buffer where they can then be read out. You can either specify the range of buffers to use (by passing in the start and stop parameters) or use the entire range of the frame-buffer (by using the parameter-less version of the method).

You can also indicate whether a pre-determined amount of images are to be acquired (by setting the numBuf parameter) or whether to simply keep acquiring until stopped by you (either by setting the numBuf parameter to 0 or by calling the parameter-less version of the method).

While in live mode, you can check for image arrivals in two different ways. The field count parameter can be polled (by calling the GetFieldCount method). This parameter will increase every time an image is received by the host PC. Alternatively, you can wait for a specified amount of time by calling the WaitImage method. When called, the WaitImage method returns as soon as an image has been received by the host PC, or will throw an exception if an image has not arrived before the specified time-out period.

The detector automatically exits live mode as soon as the desired number of images have been acquired or when you call the GoUnlive method. Note that an exception will be thrown if the GoUnlive method is called when the detector is not in live mode (the IsLive method can be used to check for the status of live mode before calling GoUnlive).

#### Image Callbacks

Another option for image acquisition is the use of user-specified callback functions. These functions defined by you will be automatically called every time an image is received by the host PC (whether an image is Snapped or the detector is in Live Mode). The signature of the callback function must match the following:

void (\*IMAGE\_CALLBACK)(int fc, int buf, DexelaDetector\* det);

Here, the parameter fc corresponds to the field-count for the image that was captured, and the parameter buf corresponds to the buffer into which the image was captured. This allows the user to know the number of the image that was captured as well as what buffer from which to read the image out. The last parameter is a pointer to the DexelaDetector object itself. This gives access to the instance of the DexelaDetector object from which the image was acquired and all of its associated methods in the callback function itself.

#### Reading Image Buffers

Regardless of the mode of acquisition, images that arrive at the host PC are written to the underlying frame-buffer. This buffer is in system memory (RAM). The image data can be read out of the relevant buffer by calling one of the ReadBuffer methods.

The bufNum parameter in these methods should contain the index number of the buffer from which to read the data out. This number can be determined by calling the GetCapturedBuffer method, which informs you of the index number for the most recent buffer to be filled.

The data being read out from the frame-buffer can either be read directly into a user created memory buffer or into a DexImage Class object. For most scenarios, we recommend using the DexImage version of the method as this will simplify many of the following operations (for example, sorting, corrections, and so on).

### DexelaDetectorCL and DexelaDetectorGE

The DexelaDetectorCL and DexelaDetectorGE classes are derived from the DexelaDetector class and have access to all methods that are defined in the base class.

In addition, each class will have access to methods that are specific to the respective interface.

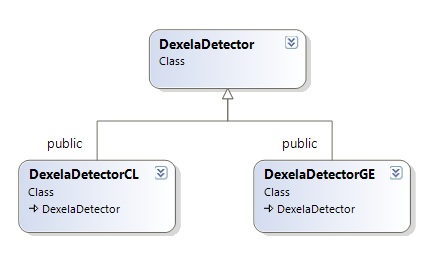


Figure 3. DexelaDetector Class Diagram

The DexelaDetectorGE class will have access to GigE specific methods. For example, it has a method called SetPersistentIPAddress that allows the user to set a new persistent (that is, will persist across power cycles) IP address for the detector.

Similarly, the DexelaDetectorCL class will have access to CameraLink-specific methods. For example, it has a method called PowerCLInterface that provides the user with a means to power up/down the CameraLink interface itself.

## BusScanner Class

The BusScanner class can be used to determine what (if any) DexelaDetectors are currently connected to the system and to return the information necessary (in the form of a DevInfo object) to establish a connection with the detector (that is, for creating a DexelaDetector Class object). The class includes two main methods: EnumerateDevices and GetDevice.

The EnumerateDevices method scans all the busses on the host system to determine what Dexela detectors are connected and returns the number of detectors that are connected to the system. There are also GigE (EnumerateGEDevices) and CameraLink (EnumerateCLDevices) versions of this method that restrict the scanning to the desired interface type.

The GetDevice method returns a DevInfo object for the device at the desired index. This call automatically calls EnumerateDevices if it was not called previously. This method also has interface-specific versions (that is, GetDeviceGE and GetDeviceCL) that can be used if the interface of the detector is known.

## DexImage Class

The DexImage class is used to perform any and all operations on the actual image data received from a Dexela detector. The class can be used to perform reading/writing operations (that is, the ReadImage/WriteImage methods) as well as for all correction operations. The correction operations is described in more detail in the following sections.

### Unscrambling Corrections

As described in Section 4, the data that arrives from Dexela detectors is interlaced and must be sorted to be properly displayed and saved. The DexImage class provides a means to do so with a simple method call (that is, UnscrambleImage).

In order for the DexImage class to be able to apply the proper sorting algorithm, it must know the model number and the binning mode for the detector from which the image was acquired. If the data is read directly into a DexImage object (by use of the DexelaDetector class’ ReadBuffer method), these parameters should be read directly from the detector. If that is not the case, these parameters can be manually configured using the SetImageParameters method.

### Linearity Corrections

As described in Section 5.2, the data from the detectors may have a slightly non-linear response. For detectors that do not implement the linearity correction in firmware, this can be corrected by applying the software linearity correction.

The linearity correction consists of a piece-wise linear approximation where sections are defined by an array of integers (linearization starts). The correction can be applied by calling the LinearizeData method. The method uses a default set of linear sections, but this can be overwritten by user-defined sections (by calling the SetLinearizationStarts method).

### Offset Corrections

Offset corrections (described in more detail in Section 5.1) consist of standard dark-image subtraction. In order to apply this correction, the dark image data should first be loaded by calling the LoadDarkImage method. This method can either take as input dark image data that has been saved to disk or another DexImage instance.

When the dark image data is loaded, a median image will be created from the loaded dark image data. A DexImage instance containing the median dark image data can be obtained by calling the GetDarkImage method.

Once the dark-data is loaded, the correction can be applied by simply calling the SubtractDark method. **Note:** Calling this method overwrites the image data that is, if raw unprocessed image data is required it should be saved before calling this method).

### Gain Corrections

Gain corrections (described in more detail in Section 5.3) consist of dividing the image data by a flat-field (that is, flood) image of known intensity. Similar to offset corrections, the flood-image data must first be loaded into the DexImage object by calling the LoadFloodImage method.

When flood image data is loaded, a median image is first created from the loaded flood-image data. This data is then fixed. This means that the reciprocal of the image is taken and normalized about 1. This is done to speed up the flood correction procedure (multiplication is faster than division). Again, a DexImage instance containing the fixed flood image data can be obtained by calling the GetDarkImage method. **Note**: This data will consist of a single floating-point image.

Once the flood-image is loaded, the correction can be applied by simply calling the FloodCorrection method. **Note:** Calling this method overwrites the image data (that is, if raw unprocessed image data is required it should be saved before calling this method).

### Defect Corrections

The defect corrections consist of the corrections described in Section 5.6. In order for these to be properly applied, a defect map for the given detector and binning mode must first be loaded by calling the LoadDefectMap method.

Once the defect map is loaded, the correction can be applied by calling the DefectCorrection method. **Note:** Calling this method overwrites the image data (that is, if raw unprocessed image data is required it should be saved before calling this method).

### Full Corrections

The offset, gain, and defect corrections can be called in a single step by calling the FullCorrection method. In order for this correction to properly be applied, the dark-image, flood-image, and defect-map data must first be loaded (by use of the LoadDarkImage, LoadFloodImage, and LoadDefectMap methods).

## DexelaException Class

Instead of returning very basic error codes, the DexelaDetector API will throw a DexelaException type exception whenever an error occurs. The reasons for this are that more meaningful error information can be passed on to you.

The class consists of the overwritten *what* method that is standard for exception objects, as well as some custom methods that can offer some more insight into the source of the exception. The *what* method returns a message string that informs you of the possible sources of the exception and what can be done to address the issue.

The top-level source of the error can be determined by calling the GetFunctionName method. This informs the user of the top-level function that threw the exception. This gives you a good starting point from which to debug their application.

For consistency with previous APIs, the DexelaException class can also return a member of the Derr enumeration that corresponds to a basic error code.

The rest of the DexelaException methods (for example, GetFileName, GetLineNumber, GetTransportMessage, GetTransportError code) may not be useful or meaningful to you, but it may be useful if lower-level debugging is required. In this case, you can call these methods and report the return values to PerkinElmer support staff.

## DexelaDetector API Wrapper Libraries

In order to provide support to users that do not use C++ in their applications, some basic wrapper libraries have been developed to support other languages. These libraries wrap the functionality of the C++ classes into their respective languages. Currently, the API includes Python (version 2.7 and 3.4), C and .NET (for example, C# and VB.NET) wrapper libraries. Each wrapper library includes its own help (chm) file that describes the classes/methods/parameters for that particular library.

## DexelaDetector API Example Source Code

Often times, the best way to learn an API is to review source code examples. For this reason, the DexelaDetector SDK includes several different Visual Studio projects that show the basic principles of using the API to control Dexela detectors and acquiring images from them. The C++, C and .NET versions of the library each have a Visual Studio solution that includes multiple examples (in separate projects). The Python version of the library contains a set of scripts that correspond to the projects from the Visual studio solutions. Each of these examples show you a specific mode of operation or feature of the API. The following gives a brief overview of what each source code example shows:

**Note:** Due to Microsoft user access control issues, the examples may not be stored in the same folder as the rest of the API. The default install location for the examples is in the public documents folder (typically, C:\Users\Public\Documents in Windows7).

|  |  |
| --- | --- |
| **Project Name** | **Description:** |
| SingleImageEx | Connects to a detector, initializes it with basic settings, and snaps an image from it. |
| SequenceModeEx | Puts a detector into sequence mode, and acquires a sequence of images from it. |
| PulseGeneratorEx | Puts the detector into PulseGenerator mode, and acquires a sequence of images from it. |
| ImageCallbackEx | Shows how a simple, user-defined callback function can be passed to the API, and how it will be called every time an image is received by the host PC. |
| ImageChunksEx | Shows how multiple threads can be used to write received image data to disk. This can be useful if large sequences of images are to be captured that may not otherwise fit into the frame-buffer (for example, the system may not have enough ram to buffer the entire image sequence). |
| CLDetectorEx | Shows how to use the DexelaDetectorCL class and its associated CameraLink specific methods (for example, it will power-cycle the CameraLink interface). |
| GigEDetectorEx | Shows how to use the DexelaDetectorGE class and its associated GigE specific methods (for example, it will reset the detectors persistent IP address). |
| ImageCorrectionsEx | Shows how to apply all the different image corrections that may need to be applied to get a clean image from the detector. The example loads some sample dark, flood, and defect map images and uses these to correct a raw image that was acquired and saved from a DexelaDetector. |

# Appendix A – SMV File Format

SMV is a file format that contains 512 bytes of header followed by 16-bit integer or 32-bit float image data. Figure 3 and Table 2 show an example of the SMV header.

{

HEADER\_BYTES= 0512;

FILENAME=;

FILETYPE=;

DIM=1;

BYTE\_ORDER=little\_endian;

TYPE=UNSIGNED\_SHORT;

SIZE1=3072;

SIZE2=3888;

SIZE3=1;

SIZE4=1;

}

Figure 3. Example of an SMV Header

The header contains the following information about the image.

|  |  |  |
| --- | --- | --- |
| Label | Description | Possible Values |
| HEADER\_BYTES | Size (in bytes) of the current header | For SMV, the byte count is always 512 |
| FILENAME | Location (path and name) of the file | \path\filename |
| FILETYPE | *Not used* | *Not used* |
| DIM | Dimension of the image | 1 is for a single image  2 for a stack of images |
| BYTE\_ORDER | Byte order of the image data | little\_endian or large\_endian  In the standard case, little\_endian is used |
| TYPE | Image data type | UNSIGNED\_SHORT or FLOAT |
| SIZE1 | Width of the image | *Depending on detector model* |
| SIZE2 | Height of the image | *Depending on detector model* |
| SIZE3 | Number of images saved | *Depending on detector model* |
| SIZE4 | *Not used* | *Not used* |

Table 2. Description of the Labels used in Defect Map