# DexelaDetector API

Here are the classes, structs, unions and interfaces with brief descriptions:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BusScanner</strong></td>
<td>This class is used to scan the different interfaces and give information about devices found.</td>
</tr>
<tr>
<td><strong>DetStatus</strong></td>
<td>Structure to hold the detector current status.</td>
</tr>
<tr>
<td><strong>DevInfo</strong></td>
<td>A structure to hold device information.</td>
</tr>
<tr>
<td><strong>DexelaDetector</strong></td>
<td>This class is used to control any interface-type Detector and acquire images from it. It will provide all the basic functionality required for all different Dexela detectors. For interface specific functionality please see the interface specific classes (e.g. <strong>DexelaDetectorGE</strong>, <strong>DexelaDetectorCL</strong>).</td>
</tr>
<tr>
<td><strong>DexelaDetectorCL</strong></td>
<td>This class is used to control CameraLink Type Detectors. It will give access to functions that are not available to other interface-type detectors. <strong>Note:</strong> For all standard detector function calls please see the <strong>DexelaDetector</strong> class (these</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>DexelaDetectorGE</strong></td>
<td>This class is used to control GigE Type Detectors. It will give access to functions that are not available to other interface-type detectors. <strong>Note:</strong> For all standard detector function calls please see the <strong>DexelaDetector</strong> class (these functions are also available to <strong>DexelaDetectorGE</strong> objects)</td>
</tr>
<tr>
<td><strong>DexelaException</strong></td>
<td>This class contains information about any possible error's in the API. In the event of a problem a <strong>DexelaException</strong> will be thrown. <strong>Note:</strong> It is suggested that you wrap your code in a try-catch block to ensure that if any errors occur you can detect (and properly handle them) in your code.</td>
</tr>
<tr>
<td><strong>DexImage</strong></td>
<td>This class is used to store and handle the images acquired from a detector.</td>
</tr>
<tr>
<td><strong>GeometryCorrectionParams</strong></td>
<td>A structure used to specify the new image dimensions for geometry correction</td>
</tr>
</tbody>
</table>
BusScanner Class Reference

This class is used to scan the different interfaces and give information about devices found. More...

#include <BusScanner.h>
Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BusScanner</strong>(void)</td>
<td>Constructor for <strong>BusScanner</strong>. More...</td>
</tr>
<tr>
<td>~<strong>BusScanner</strong>(void)</td>
<td>Destructor for <strong>BusScanner</strong>. More...</td>
</tr>
<tr>
<td>int <strong>EnumerateDevices</strong>()</td>
<td>This method will enumerate all devices (regardless of interface) currently connected to the system. More...</td>
</tr>
<tr>
<td>int <strong>EnumerateGEDevices</strong>()</td>
<td>This method will enumerate all GigE devices currently connected to the system. More...</td>
</tr>
<tr>
<td>int <strong>EnumerateCLDevices</strong>()</td>
<td>This method will enumerate all CameraLink devices currently connected to the system. More...</td>
</tr>
</tbody>
</table>

**DevInfo** **GetDevice**(int index)
This method will return a **DevInfo** object for the device at the specified index. This object contains all necessary information for establishing a connection with the detector (**DexelaDetector**(DevInfo &defInfo)).

**Note**: To determine how many devices are currently connected to the system you can call **EnumerateDevices**

**Note2**: This method will call **EnumerateDevices** automatically if it has not previously been called More...
### DevInfo

**GetDeviceCL (int index)**  
This method will return a `DevInfo` object for the CameraLink device at the specified index. This object contains all necessary information for establishing a connection with the detector (`DexelaDetector(DevInfo &defInfo)`).

**Note:** To determine how many CL devices are currently connected to the system you can call `EnumerateCLDevices`.

**Note2:** This method will call `EnumerateCLDevices` automatically if it has not previously been called.
<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ScanMockSetter</strong></td>
</tr>
</tbody>
</table>
Detailed Description

This class is used to scan the different interfaces and give information about devices found.
Constructor & Destructor Documentation

**BusScanner::BusScanner ( void )**

Constructor for **BusScanner**.

**BusScanner::~BusScanner ( void )**

Destructor for **BusScanner**.
Member Function Documentation

**int BusScanner::EnumerateCLDevices ( )**

This method will enumerate all CameraLink devices currently connected to the system.

**Returns**

The count of the number of CL devices found

**Exceptions**

DexelaException

**int BusScanner::EnumerateDevices ( )**

This method will enumerate all devices (regardless of interface) currently connected to the system.

**Returns**

The count of the number of devices found

**Exceptions**

DexelaException

**int BusScanner::EnumerateGEDevices ( )**

This method will enumerate all GigE devices currently connected to the system.

**Returns**

The count of the number of GigE devices found

**Exceptions**
**DexelaException**

**DevInfo BusScanner::GetDevice ( int index )**

This method will return a `DevInfo` object for the device at the specified index. This object contains all necessary information for establishing a connection with the detector 
(`DexelaDetector(DevInfo &defInfo)`).

**Note:** To determine how many devices are currently connected to the system you can call `EnumerateDevices`  
**Note2:** This method will call `EnumerateDevices` automatically if it has not previously been called

**Parameters**

index The index of the device to access. If this index is out of bounds an exception will be thrown.

**Returns**

The `DevInfo` object with the detector at the desired index

**Exceptions**

DexelaException

**DevInfo BusScanner::GetDeviceCL ( int index )**

This method will return a `DevInfo` object for the CameraLink device at the specified index. This object contains all necessary information for establishing a connection with the detector 
(`DexelaDetector(DevInfo &defInfo)`).

**Note:** To determine how many CL devices are currently connected to the system you can call `EnumerateCLDevices`  
**Note2:** This method will call `EnumerateCLDevices` automatically if it has not previously been called

**Parameters**

index The index of the device to access. If this index is out of bounds an exception will be thrown.
Returns
The **DevInfo** object with the detector at the desired index

Exceptions
**DexelaException**

**DevInfo** `BusScanner::GetDeviceGE ( int index )`

This method will return a **DevInfo** object for the GigE device at the specified index. This object contains all necessary information for establishing a connection with the detector (`DexelaDetector(DevInfo &defInfo)`).

**Note:** To determine how many GigE devices are currently connected to the system you can call `EnumerateGEDevices`

**Note2:** This method will call `EnumerateGEDevices` automatically if it has not previously been called

**Parameters**

- **index** The index of the device to access. If this index is out of bounds an exception will be thrown.

**Returns**
The **DevInfo** object with the detector at the desired index

Exceptions
**DexelaException**

The documentation for this class was generated from the following files:

- **BusScanner.h**
- **BusScanner.cpp**

Generated on Tue Nov 25 2014 10:22:44 for DexelaDetector API by **doxygen** 1.8.7
### DetStatus Struct

Structure to hold the detector current status. [More...](#)

```c
#include <DexDefs.h>
```
### Public Attributes

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ExposureModes</strong></td>
<td>exposureMode</td>
<td>The currently set exposure mode</td>
</tr>
<tr>
<td><strong>FullWellModes</strong></td>
<td>fullWellMode</td>
<td>The currently set Full Well mode</td>
</tr>
<tr>
<td><strong>float</strong></td>
<td>exposureTime</td>
<td>The currently set exposure time</td>
</tr>
<tr>
<td><strong>bins</strong></td>
<td>binLevel</td>
<td>The currently set bin level</td>
</tr>
<tr>
<td><strong>ExposureTriggerSource</strong></td>
<td>triggerSource</td>
<td>The currently set Trigger Source</td>
</tr>
<tr>
<td><strong>BOOL</strong></td>
<td>testMode</td>
<td>True if the detector test mode is set to on</td>
</tr>
</tbody>
</table>
Detailed Description

Structure to hold the detector current status.
Member Data Documentation

**bins** `DetStatus::binLevel`

The currently set bin level

**ExposureModes** `DetStatus::exposureMode`

The currently set exposure mode

**float** `DetStatus::exposureTime`

The currently set exposure time

**FullWellModes** `DetStatus::fullWellMode`

The currently set Full Well mode

**BOOL** `DetStatus::testMode`

True if the detector test mode is set to on

**ExposureTriggerSource** `DetStatus::triggerSource`

The currently set Trigger Source

The documentation for this struct was generated from the following file:
DexelaDetector API

DevInfo Struct

Reference

A structure to hold device information. More...

#include <DexDefs.h>
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>model</td>
<td>The Device Model Number</td>
</tr>
<tr>
<td>int</td>
<td>serialNum</td>
<td>The Device Serial Number</td>
</tr>
<tr>
<td>char</td>
<td>iface</td>
<td>Pointer to the parameter needed for opening detector</td>
</tr>
<tr>
<td>int</td>
<td>unit</td>
<td>Unit number for cameralink detectors</td>
</tr>
<tr>
<td></td>
<td>transport</td>
<td>Low level tranport library</td>
</tr>
</tbody>
</table>

**DetectorInterface iface**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>param</td>
<td>Pointer to the parameter needed for opening detector</td>
</tr>
</tbody>
</table>

**TransportLib transport**

Low level tranport library
Detailed Description

A structure to hold device information.
# Member Data Documentation

<table>
<thead>
<tr>
<th>DetectorInterface DevInfo::iface</th>
</tr>
</thead>
<tbody>
<tr>
<td>The interface type (e.g. GigE or CameraLink)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int DevInfo::model</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Device Model Number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>char DevInfo::param[50]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the parameter needed for opening detector</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int DevInfo::serialNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Device Serial Number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int DevInfo::unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit number for cameralink detectors</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:

- **DexDefs.h**
This class is used to control any interface-type Detector and acquire images from it. It will provide all the basic functionality required for all different Dexela detectors. For interface specific functionality please see the interface specific classes (e.g. DexelaDetectorGE, DexelaDetectorCL). More...

#include <DexelaDetector.h>

Inheritance diagram for DexelaDetector:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DexelaDetector</strong> <em>(DevInfo &amp;devInfo)</em></td>
<td>Constructor for DexelaDetector. This version uses the DevInfo struct returned from a GetDevice, GetDeviceGE or GetDeviceCL call.</td>
<td>More</td>
</tr>
<tr>
<td><strong>DexelaDetector</strong> *(DetectorInterface transport, int unit, const char <em>params)</em></td>
<td>Constructor for DexelaDetector. This version assumes you know the interface and the correct parameters to connect to the detector.</td>
<td>More</td>
</tr>
<tr>
<td>virtual <strong>~DexelaDetector</strong> <em>(void)</em></td>
<td>Destructor for DexelaDetector.</td>
<td>More</td>
</tr>
<tr>
<td>virtual void <strong>OpenBoard</strong> ()</td>
<td>Opens the connection to the detector. Every open should be matched with a close to free resources.</td>
<td>More</td>
</tr>
<tr>
<td>void <strong>OpenBoard</strong> <em>(int NumBufs)</em></td>
<td>Opens the connection to the detector and sets the number of buffers to use/allocate. Every open should be matched with a close to free resources.</td>
<td>More</td>
</tr>
<tr>
<td>void <strong>CloseBoard</strong> ()</td>
<td>Closes the connection to the detector.</td>
<td>More</td>
</tr>
<tr>
<td>int <strong>GetBufferXdim</strong> <em>(void)</em></td>
<td>Get the x dimension of the transport buffer (in bytes)</td>
<td>More</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><strong>GetIntBufferYdim (void)</strong></td>
<td>Get the y dimension of the transport buffer (in pixels) More...</td>
<td></td>
</tr>
<tr>
<td><strong>GetIntNumBuffers (void)</strong></td>
<td>Get the number of internal buffers that are currently allocated for the detector. More...</td>
<td></td>
</tr>
<tr>
<td><strong>GetIntCapturedBuffer (void)</strong></td>
<td>Gets the number of the buffer just captured. This can be used to determine which buffer to read-out. More...</td>
<td></td>
</tr>
<tr>
<td><strong>GetIntFieldCount (void)</strong></td>
<td>Gets the number of fields(frames) captured so far. More...</td>
<td></td>
</tr>
<tr>
<td>*<em>void ReadBuffer (int bufNum, byte <em>buffer)</em></em></td>
<td>Reads the specified transport buffer into the passed in buffer (byte*). Note: <strong>GetIntCapturedBuffer</strong> can be used to get the number of the lastest buffer to be filled. More...</td>
<td></td>
</tr>
<tr>
<td><strong>void ReadBuffer (int bufNum, DexImage &amp;img, int iZ=0)</strong></td>
<td>Reads the specified transport buffer into the passed in <strong>DexImage</strong> object at the passed in plane. Note: <strong>GetIntCapturedBuffer</strong> can be used to get the number of the lastest buffer to be filled. More...</td>
<td></td>
</tr>
<tr>
<td>*<em>void WriteBuffer (int bufNum, byte <em>buffer)</em></em></td>
<td>Writes data to the specified transport buffer. More...</td>
<td></td>
</tr>
</tbody>
</table>
void **SetFullWellMode** (FullWellModes fwm)
Sets the full well mode parameter of the detector. More...

void **SetExposureMode** (ExposureModes mode)
Sets the ExposureMode parameter of the detector More...

void **SetExposureTime** (float timems)
Sets the exposure time parameter of the detector More...

void **SetBinningMode** (bins flag)
Sets the binning mode of the detector. More...

void **SetTestMode** (BOOL SetTestOn)
Enables/disables test mode. The detector will output a generated test pattern if this mode is turned on. More...

void **SetTriggerSource** (ExposureTriggerSource ets)
Sets the trigger source setting on the detector More...

void **SetNumOfExposures** (int num)
Sets the number of exposures to acquire after a trigger. This is only relevant in Sequence_Exposure and Frame_Rate_exposure modes of operation. More...

int **GetNumOfExposures** ()
Gets the number of exposures setting from the detector. This is only relevant in Sequence_Exposure and
**Frame_Rate_exposure** modes of operation. More...

```c
void SetGapTime (float timems)
Sets the gap-time setting of the detector. When run in **Frame_Rate_exposure** mode the detector will insert this gap period between consecutive frames in an image sequence.
**Note:** The minimum time for the gap-time setting is the current readout-time for the detector. Attempting to write anything smaller to the detector will result in a gap-time equal to the readout-time. More...
```

```c
float GetGapTime ()
Gets the current gap-time setting of the detector. When run in **Frame_Rate_exposure** mode the detector will insert this gap period between consecutive frames in an image sequence.
**Note:** The minimum time for the gap-time setting is the current readout-time for the detector. Attempting to write anything smaller to the detector will result in a gap-time equal to the readout-time. More...
```

```c
bool IsConnected ()
Check to see if the connection to the detector is open (i.e. **OpenBoard**) More...
```

```c
ExposureModes GetExposureMode ()
Gets the ExposureMode parameter of the detector. More...
```

```c
float GetExposureTime ()
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DetStatus</strong></td>
<td>GetDetectorStatus () Returns the current settings of the detector in the form of a <strong>DetStatus</strong> object. More...</td>
</tr>
<tr>
<td><strong>ExposureTriggerSource</strong></td>
<td>GetTriggerSource () Gets the current trigger source setting from the detector More...</td>
</tr>
<tr>
<td><strong>BOOL</strong></td>
<td>GetTestMode () Gets the current state of the detector test mode (on/off) More...</td>
</tr>
<tr>
<td><strong>FullWellModes</strong></td>
<td>GetFullWellMode () Gets the current detector well-mode. More...</td>
</tr>
<tr>
<td><strong>bins</strong></td>
<td>GetBinningMode () Gets the current state of the detector binning mode More...</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>GetSerialNumber () Gets the detector serial number. More...</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>GetModelNumber () Gets the detector model number. More...</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>GetFirmwareVersion () Gets the detector firmware version number. More...</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td>GetFirmwareBuild (int &amp;iDayAndMonth, int &amp;iYear, int &amp;iTime) Gets the detector firmware build date.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DetectorInterface</td>
<td></td>
</tr>
<tr>
<td>GetTransportMethod ()</td>
<td>Returns the communication method (i.e. interface) for the detector object.</td>
</tr>
<tr>
<td>double GetReadOutTime ()</td>
<td>This method will return the read-out time of the detector (in ms) for it's current binning mode.</td>
</tr>
<tr>
<td>bool IsCallbackActive ()</td>
<td>This method will inform the user if the callback mode (i.e. background thread) is currently active.</td>
</tr>
<tr>
<td>bool IsLive ()</td>
<td>This method will inform the user if detector is currently in Live mode.</td>
</tr>
<tr>
<td>void Snap (int buffer, int timeout)</td>
<td>Snaps an image into the specified buffer. Note: If the detector trigger source is set to Internal Software, this call will automatically trigger the detector.</td>
</tr>
<tr>
<td>int ReadRegister (int address, int sensorNum=1)</td>
<td>Reads the specified register from the detector. The sensor number corresponds to the desired sensor from which to read the register. The SensorNumber will default to 1 (master-sensor) if not specified otherwise by the user.</td>
</tr>
<tr>
<td>void WriteRegister (int address, int value, int sensorNum=0)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>Write</code></td>
<td>Writes the value to the specified register from the detector. The sensor number corresponds to the desired sensor to which the value will be written. The SensorNumber will default to 0 (broadcast to all sensors) if not specified otherwise by the user. More...</td>
</tr>
<tr>
<td><code>ClearCameraBuffer (int i)</code></td>
<td>Clears (i.e. zero-out) the specified camera buffer. More...</td>
</tr>
<tr>
<td><code>ClearBuffers ()</code></td>
<td>Clears all the camera buffers More...</td>
</tr>
<tr>
<td><code>LoadSensorConfigFile (char *filename)</code></td>
<td>Loads the sensor configuration file into the detector. This file will write values to the ADC offset registers for each sensor in the detector. More...</td>
</tr>
<tr>
<td><code>SoftReset (void)</code></td>
<td>Cycles the power on the detector More...</td>
</tr>
<tr>
<td><code>GoLiveSeq (int start, int stop, int numBuf)</code></td>
<td>Sets the host computer up to be ready to recieve images into the specified buffer range. More...</td>
</tr>
<tr>
<td><code>GoLiveSeq ()</code></td>
<td>Sets the host computer up to be ready to recieve images. This call will use all available buffers in a circular fashion (i.e. ring-buffer). More...</td>
</tr>
<tr>
<td><code>GoUnLive ()</code></td>
<td>Exits live mode. The host computer will...</td>
</tr>
</tbody>
</table>
null

void **SoftwareTrigger** ()
Sends a trigger to the detector (will only work if the trigger source is set to **Internal_Software**) More...

void **EnablePulseGenerator** (float frequency)
This function will enable the pulse generator software trigger signal. In this mode the software trigger can be continuously sent to the detector at the desired frequency.

**Note:** In order to use this mode the trigger source should be set to **Internal_Software**

**Note2:** To actually enable the pulse train you must call **ToggleGenerator**. A **SoftwareTrigger** call will not work when in this mode. More...

void **EnablePulseGenerator** ()
This function is identical to **EnablePulseGenerator**, except that the frequency of the pulse train is set automatically. The frequency will be set such as to ensure continuous image acquisition for the current detector/binning mode.

**Note:** In order to use this mode the trigger source should be set to **Internal_Software**

**Note2:** To actually enable the pulse train you must call **ToggleGenerator**. A **SoftwareTrigger** call will not work when in this mode. More...
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void DisablePulseGenerator ()</code></td>
<td>This function will disable Pulse Generator mode. After calling this you should be able to use the <strong>SoftwareTrigger</strong> call. More...</td>
</tr>
<tr>
<td><code>void ToggleGenerator (BOOL onOff)</code></td>
<td>This function will control the pulse train. <strong>Note:</strong> In order to use this mode the pulse generator must be enabled. See <strong>EnablePulseGenerator</strong>. More...</td>
</tr>
<tr>
<td><code>void WaitImage (int timeout)</code></td>
<td>This function will wait for the specified amount of time for an image to arrive. <strong>Note:</strong> If the image arrives before then it will return as soon as it does (i.e. it won't wait for the duration of the timeout period). If the image does not arrive in the specified time a <strong>DexelaException</strong> will be thrown. More...</td>
</tr>
<tr>
<td><code>void SetCallback (IMAGE_CALLBACK func)</code></td>
<td>Sets the user defined callback function to be called for every image arrival event. More...</td>
</tr>
<tr>
<td><code>void StopCallback ()</code></td>
<td>This function will terminate the callback loop (i.e. will wait for all spawned threads to finish executing). More...</td>
</tr>
<tr>
<td><code>void CheckForCallbackError ()</code></td>
<td>This function will check to see if any errors have occurred in the background thread that is running when using callbacks. This thread is activated after a call to <strong>SetCallback</strong> and terminated with a call to <strong>StopCallback</strong>. If no error has...</td>
</tr>
</tbody>
</table>
occurred this method will just return. If an error has occurred a DexelaException will be thrown. More...

void CheckForLiveError ()
This function will check to see if any errors have occurred in the background thread that is running when using live-mode. This thread is activated after a call to GoLiveSeq and terminated with a call to GoUnLive. If no error has occurred this method will just return. If an error has occurred a DexelaException will be thrown. More...

void SetPreProgrammedExposureTimes (int numExposures, float *exposuretimes_ms)
This method will set the exposure times for pre-programmed exposure mode. More...

void SetROICoordinates (unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight)
This method set the coordinates of the ROI when detector runs in ROI mode.

Parameters
- **usStartColumn**: Sensor column to start the ROI read out from.
- **usStartRow**: Sensor width to start the ROI read out from.
- **usROIWidth**: Width (number of columns) of the ROI.
### usROIHeight

**Height (number of columns) of the ROI.**

### Exceptions

**DexelaException**

### void GetROICoordinates (unsigned short &usStartColumn, unsigned short &usStartRow, unsigned short &usROIWidth, unsigned short &usROIHeight)

This method retrieves the coordinates of the region of interest (ROI) set within the detector. [More...](#)

### void EnableROIMode (bool bEnableROI)

This method activates or deactivates the ROI mode of the detector. [More...](#)

### bool GetROIState ()

This method retrieves the enabled state of the region of interest of the detector. [More...](#)

### unsigned short GetSensorHeight (unsigned short uiSensorID=1)

Gets the height of the sensor in pixels. [More...](#)

### unsigned short GetSensorWidth (unsigned short uiSensorID=1)

Gets the width of the sensor in pixels. [More...](#)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool IsFrameCntWithinImage ()</code></td>
<td>Checks if framecounter is displayed within the image data (being the 2nd pixel).</td>
</tr>
<tr>
<td><code>void EnableFrameCntWithinImage (unsigned short usEnable)</code></td>
<td>Enables displaying the framecounter in the image (being the 2nd pixel). Positive flag enables it.</td>
</tr>
<tr>
<td><code>void SetSlowed (bool flag)</code></td>
<td>This method can specify to the api that the detector being used is a slowed-down detector (e.g. mammo detector). This should not be necessary as the API should be able to determine most of the time whether the firmware version is a slowed down one. However, for certain older detectors/firmwares this may not be possible. In this case this method can be used to inform the library that the firmware is slowed down and it will use the correct read-out times.</td>
</tr>
<tr>
<td><code>void SetReadoutMode (ReadoutModes mode)</code></td>
<td>Sets the ReadoutMode parameter of the detector.</td>
</tr>
<tr>
<td><code>ReadoutModes GetReadoutMode ()</code></td>
<td>Gets the ReadoutModes parameter of the detector.</td>
</tr>
<tr>
<td><code>int QueryReadoutMode (ReadoutModes mode)</code></td>
<td>Query the detector to see if the desired readout mode is present (i.e. available). <strong>Note:</strong> Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that</td>
</tr>
</tbody>
</table>
it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...

int **QueryExposureMode** ([ExposureModes](#) mode)
Query the detector to see if the desired exposure mode is present (i.e. available).

**Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...

int **QueryTriggerSource** ([ExposureTriggerSource](#) ets)
Query the detector to see if the desired trigger source is present (i.e. available).

**Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...
int QueryFullWellMode (FullWellModes fwm)
Query the detector to see if the desired full-well mode is present (i.e. available).
**Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...

int QueryBinningMode (bins flag)
Query the detector to see if the desired binning mode is present (i.e. available).
**Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...
## Protected Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boost::shared_ptr&lt; baseDetector &gt;</code></td>
<td><code>base</code></td>
</tr>
<tr>
<td><code>boost::shared_ptr&lt; gigEDetector &gt;</code></td>
<td><code>gigeDet</code></td>
</tr>
<tr>
<td><code>boost::shared_ptr&lt; camLinkDetector &gt;</code></td>
<td><code>clDet</code></td>
</tr>
<tr>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>Friends</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td></td>
</tr>
<tr>
<td>baseBusScanner</td>
<td></td>
</tr>
<tr>
<td>MockSetter</td>
<td></td>
</tr>
<tr>
<td>DexelaDetectorPy</td>
<td></td>
</tr>
<tr>
<td>Dex_CL</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

This class is used to control any interface-type Detector and acquire images from it. It will provide all the basic functionality required for all different Dexela detectors. For interface specific functionality please see the interface specific classes (e.g. DexelaDetectorGE, DexelaDetectorCL).
Constructor & Destructor Documentation

DexelaDetector::DexelaDetector ( DevInfo & devInfo )

Constructor for DexelaDetector. This version uses the DevInfo struct returned from a GetDevice, GetDeviceGE or GetDeviceCL call.

Parameters
  devInfo The DevInfo object for the desired detector.

Exceptions
  DexelaException

DexelaDetector::DexelaDetector ( DetectorInterface transport, int unit, const char * params )

Constructor for DexelaDetector. This version assumes you know the interface and the correct parameters to connect to the detector.

Parameters
  transport The DetectorInterface for the detector (i.e. CL or GIGE)
  unit The unit number for CL type detectors. For GIGE detectors this can be set to 0
  params The parameter string for connection to the detector. For GIGE detectors this should be the detector IP address. For CL detectors this parameter will be ignored.

Exceptions
DexelaException

DexelaDetector::~DexelaDetector ( void )

Destructor for DexelaDetector.
void DexelaDetector::CheckForCallbackError ()

This function will check to see if any errors have occurred in the background thread that is running when using callbacks. This thread is activated after a call to SetCallback and terminated with a call to StopCallback. If no error has occurred this method will just return. if an error has occurred a DexelaException will be thrown.

Exceptions
   DexelaException

void DexelaDetector::CheckForLiveError ()

This function will check to see if any errors have occurred in the background thread that is running when using live-mode. This thread is activated after a call to GoLiveSeq and terminated with a call to GoUnLive. If no error has occurred this method will just return. if an error has occurred a DexelaException will be thrown.

Exceptions
   DexelaException

void DexelaDetector::ClearBuffers ()

Clears all the camera buffers

Exceptions
   DexelaException
void DexelaDetector::ClearCameraBuffer ( int i )

Clears (i.e. zero-out) the specified camera buffer.

Parameters
   i The buffer number to clear.

Exceptions
   DexelaException

void DexelaDetector::CloseBoard ( )

Closes the connection to the detector.

Exceptions
   DexelaException

void DexelaDetector::DisablePulseGenerator ( )

This function will disable Pulse Generator mode. After calling this you should be able to use the SoftwareTrigger call.

Exceptions
   DexelaException

void DexelaDetector::EnableFrameCntWithinImage ( unsigned short us )

Enables displaying the framecounter in the image (being the 2nd pixel).

Parameters
   usEnable Enables / disabled frame counter within image.

Exceptions
DexelaException

void DexelaDetector::EnablePulseGenerator ( float frequency )

This function will enable the pulse generator software trigger signal. In this mode the software trigger can be continuously sent to the detector at the desired frequency.

**Note:** In order to use this mode the trigger source should be set to Internal_Software

**Note2:** To actually enable the pulse train you must call ToggleGenerator. A SoftwareTrigger call will not work when in this mode.

**Parameters**

- **frequency** The frequency that the software trigger signal will be run at.

**Exceptions**

- DexelaException

void DexelaDetector::EnablePulseGenerator ( )

This function is identical to EnablePulseGenerator, except that the frequency of the pulse train is set automatically. The frequency will be set such as to ensure continuous image acquisition for the current detector/binning mode.

**Note:** In order to use this mode the trigger source should be set to Internal_Software

**Note2:** To actually enable the pulse train you must call ToggleGenerator. A SoftwareTrigger call will not work when in this mode.

**Exceptions**

- DexelaException

void DexelaDetector::EnableROIMode ( bool bEnableROI )
This method activates or deactivates the ROI mode of the detector.

**Parameters**

*bEnableROI* This parameter can have the following values:
- value 0: Disable ROI mode
- value 1: Enable ROI mode

**Exceptions**

*DexelaException*

**bins DexelaDetector::GetBinningMode ( )**

Gets the current state of the detector binning mode

**Returns**

A member of the *bins* enumeration detailing the current detector binning mode.

**Exceptions**

*DexelaException*

**int DexelaDetector::GetBufferXdim ( void )**

Get the x dimension of the transport buffer (in bytes)

**Returns**

The x dimension of the transport buffer (in bytes)

**Exceptions**

*DexelaException*

**int DexelaDetector::GetBufferYdim ( void )**

Get the y dimension of the transport buffer (in pixels)
Returns
The y dimension of the transport buffer (in pixels)

Exceptions
DexelaException

int DexelaDetector::GetCapturedBuffer ( void )

Gets the number of the buffer just captured. This can be used to determine which buffer to read-out.

Returns
The number of the last buffer, which the last captured image was written to.

Exceptions
DexelaException

DetStatus DexelaDetector::GetDetectorStatus ( )

Returns the current settings of the detector in the form of a DetStatus object.

Returns
A DetStatus structure containing settings retrieved.

Exceptions
DexelaException

ExposureModes DexelaDetector::GetExposureMode ( )

Gets the ExposureMode parameter of the detector.

Returns
The ExposureModes enumeration member that the detector is
Currently set to.

**Exceptions**

*DexelaException*

---

**float** `DexelaDetector::GetExposureTime()`

Gets the exposure time parameter of the detector (ms).

**Returns**

The exposure time (ms) that the detector is currently set to.

**Exceptions**

*DexelaException*

---

**int** `DexelaDetector::GetFieldCount()`

Gets the number of fields (frames) captured so far.

**Returns**

The number of fields captured.

**Exceptions**

*DexelaException*

---

**void** `DexelaDetector::GetFirmwareBuild(int & iDayAndMonth, int & iYear, int & iT ime)`

Gets the detector firmware build date.

**Note:** This feature may not be supported on older detectors

**Parameters**

*iDayAndMonth* The day and month of the firmware build
iYear
The year of the firmware build (YYYY format).

iT ime
The time of the firmware build (hhmm format).

Exceptions
DexelaException

int DexelaDetector::GetFirmwareVersion ()

Gets the detector firmware version number.

Returns
The detector firmware version number as read from the detector.

Exceptions
DexelaException

FullWellModes DexelaDetector::GetFullWellMode ()

Gets the current detector well-mode.

Returns
A member of the FullWellModes enumeration detailing the current detector Full-Well mode.

Exceptions
DexelaException

float DexelaDetector::GetGapTime ()

Gets the current gap-time setting of the detector. When run in Frame_Rate_exposure mode the detector will insert this gap period between consecutive frames in an image sequence.

Note: The minimum time for the gap-time setting is the current
readout-time for the detector. Attempting to write anything smaller to the detector will result in a gap-time equal to the readout-time.

**Returns**
- The gap-time in ms.

**Exceptions**
- DexelaException

```cpp
int DexelaDetector::GetModelNumber ()
```

Gets the detector model number.

**Returns**
- The detector model number as read from the detector.

**Exceptions**
- DexelaException

```cpp
int DexelaDetector::GetNumBuffers ( void )
```

Get the number of internal buffers that are currently allocated for the detector.

**Returns**
- The number of internal buffers allocated

**Exceptions**
- DexelaException

```cpp
int DexelaDetector::GetNumOfExposures ( )
```

Gets the number of exposures setting from the detector. This is only relevant in **Sequence_Exposure** and **Frame_Rate_exposure** modes of operation.
Returns

The number of exposures that will be acquired each trigger received.

Exceptions

DexelaException

ReadoutModes DexelaDetector::GetReadoutMode ( )

Gets the ReadoutModes parameter of the detector.

Returns

The ReadoutModes enumeration member that the detector is currently set to.

Exceptions

DexelaException

double DexelaDetector::GetReadOutTime ( )

This method will return the read-out time of the detector (in ms) for it's current binning mode.

Returns

The read-out time (in ms) for the current settings of the detector.

Exceptions

DexelaException

void DexelaDetector::GetROICoordinates ( unsigned short & usStartCol, unsigned short & usStartRow, unsigned short & usROIWidth, unsigned short & usROIHeight )
This method retrieves the coordinates of the region of interest (ROI) set within the detector.

**Parameters**

- **usStartColumn**: Index of first column of ROI.
- **usROIWidth**: Height (number of columns) of the ROI.
- **usStartRow**: Index of first row of ROI.
- **usROIHeight**: Height (number of rows) of the ROI.

**Exceptions**

- **DexelaException**

---

```cpp
bool DexelaDetector::GetROIState()
```

This method retrieves the enabled state of the region of interest of the detector.

**Parameters**

- **bEnableROI**: This parameter can have the following values:
  - value 0: ROI mode disabled
  - value 1: ROI mode enabled

**Exceptions**

- **DexelaException**

---

```cpp
unsigned short DexelaDetector::GetSensorHeight(unsigned short uiSensorID = 1)
```

Gets the height of the sensor in pixels.

**Parameters**

- **usSensorID**: ID of sensor FPGA to send the query to. For "global reads" sensor FPGA 1 has to be queried, using usSensorID=1/param>

**Exceptions**
### DexelaException

**unsigned short**

`DexelaDetector::GetSensorWidth ( unsigned short uiSensorID = 1 )`

Gets the width of the sensor in pixels.

**Parameters**

- **usSensorID** ID of sensor FPGA to send the query to. For "global reads" sensor FPGA 1 has to be queried, using `usSensorID=1`.

**Exceptions**

- **DexelaException**

---

**int DexelaDetector::GetSerialNumber ( )**

Gets the detector serial number.

**Returns**

- The detector serial number as read from the detector.

**Exceptions**

- **DexelaException**

---

**BOOL DexelaDetector::GetTestMode ( )**

Gets the current state of the detector test mode (on/off)

**Returns**

- Boolean value containing the current state of the detector test mode.

**Exceptions**

- **DexelaException**
DetectorInterface DexelaDetector::GetTransportMethod ( )

Returns the communication method (i.e. interface) for the detector object.

Returns
The DetectorInterface enumeration used by the detector

Exceptions
DexelaException

ExposureTriggerSource DexelaDetector::GetTriggerSource ( )

 Gets the current trigger source setting from the detector

Returns
A member of the ExposureTriggerSource enumeration detailing the current trigger source of the detector.

Exceptions
DexelaException

void DexelaDetector::GoLiveSeq ( int start, int stop, int numBuf )

Sets the host computer up to be ready to receive images into the specified buffer range.

Parameters

| start | Number of the first buffer to use for acquisition |
| stop  | Number of the last buffer to use for acquisition |
| numBuf| Number of frames to acquire. If this is set to 0 the buffer will be circular (i.e. ring-buffer). |
Exceptions
   DexelaException

void DexelaDetector::GoLiveSeq ( )

Sets the host computer up to be ready to recieve images. This call will use all available buffers in a circular fashion (i.e. ring-buffer).

Exceptions
   DexelaException

void DexelaDetector::GoUnLive ( )

Exits live mode. The host computer will no longer be ready to receive transmitted images.

Exceptions
   DexelaException

bool DexelaDetector::IsCallbackActive ( )

This method will inform the user if the callback mode (i.e. background thread) is currently active.

Returns
   A boolean value indicating whether the callback mode is active.

Exceptions
   DexelaException

bool DexelaDetector::IsConnected ( )

Check to see if the connection to the detector is open (i.e. OpenBoard)
Returns
A boolean indicating whether the connection to detector is open.

Exceptions
DexelaException

bool DexelaDetector::IsFrameCntWithinImage ( )

Checks if framecounter is displayed within the image data (being the 2nd pixel).

Parameters
usEnable 0 if not enabled; 1 if enabled

Exceptions
DexelaException

bool DexelaDetector::IsLive ( )

This method will inform the user if detector is currently in Live mode.

Returns
A boolean value indicating whether the detector is in Live mode.

Exceptions
DexelaException

void DexelaDetector::LoadSensorConfigFile ( char * filename )

Loads the sensor configuration file into the detector. This file will write values to the ADC offset registers for each sensor in the detector.

Parameters
filename The path to the sensor configuration to use.

Exceptions

DexelaException

**void DexelaDetector::OpenBoard ( )**

Opens the connection to the detector. Every open should be matched with a close to free resources.

Exceptions

DexelaException

Reimplemented in **DexelaDetectorCL**, and **DexelaDetectorGE**.

**void DexelaDetector::OpenBoard ( int NumBufs )**

Opens the connection to the detector and sets the number of buffers to use/allocate. Every open should be matched with a close to free resources.

**Parameters**

NumBufs Number of buffers to use/allocate

Exceptions

DexelaException

**int DexelaDetector::QueryBinningMode ( bins flag )**

Query the detector to see if the desired binning mode is present (i.e. available).

**Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may
still be possible to use the feature in this case but no guarantees can
be made on whether or not it will work.

Parameters

mode The bins enumeration member to be checked for.

Returns

An integer value representing whether the feature is present. A
value of 1 indicates that the feature is present. A value of 0
indicates that the feature is not present. A value of -1 indicates
that it is unknown whether the feature is present.

Exceptions

DexelaException

int
DexelaDetector::QueryExposureMode (ExposureModes mode)

Query the detector to see if the desired exposure mode is present
(i.e. available).
Note: Older detectors may not support the querying of features. In
this case a value of -1 will be returned indicating that it is unknown
whether the feature is present. In this case it is possible that the
feature is present but that the detector is unable to report so. It may
still be possible to use the feature in this case but no guarantees can
be made on whether or not it will work.

Parameters

mode The ExposureModes enumeration member to be
checked for.

Returns

An integer value representing whether the feature is present. A
value of 1 indicates that the feature is present. A value of 0
indicates that the feature is not present. A value of -1 indicates
that it is unknown whether the feature is present.

Exceptions
int DexelaDetector::QueryFullWellMode ( FullWellModes fwm )

Query the detector to see if the desired full-well mode is present (i.e. available).

Note: Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work.

Parameters
mode The FullWellModes enumeration member to be checked for.

Returns
An integer value representing whether the feature is present. A value of 1 indicates that the feature is present. A value of 0 indicates that the feature is not present. A value of -1 indicates that it is unknown whether the feature is present.

Exceptions
DexelaException

int DexelaDetector::QueryReadoutMode ( ReadoutModes mode )

Query the detector to see if the desired readout mode is present (i.e. available).

Note: Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work.
Parameters

mode The ReadoutModes enumeration member to be checked for.

Returns
An integer value representing whether the feature is present. A value of 1 indicates that the feature is present. A value of 0 indicates that the feature is not present. A value of -1 indicates that it is unknown whether the feature is present.

Exceptions
DexelaException

int DexelaDetector::QueryTriggerSource (ExposureTriggerSource ets)

Query the detector to see if the desired trigger source is present (i.e. available).

Note: Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work.

Parameters

mode The ExposureTriggerSource enumeration member to be checked for.

Returns
An integer value representing whether the feature is present. A value of 1 indicates that the feature is present. A value of 0 indicates that the feature is not present. A value of -1 indicates that it is unknown whether the feature is present.

Exceptions
DexelaException
void DexelaDetector::ReadBuffer ( int bufNum, byte * buffer )

Reads the specified transport buffer into the passed in buffer (byte*).

**Note:** GetCapturedBuffer can be used to get the number of the lastest buffer to be filled.

**Parameters**
- **bufNum** The index of the transport buffer to read from.
- **buffer** The user-created (byte*) buffer to write the image to.

**Exceptions**
- DexelaException

void DexelaDetector::ReadBuffer ( int bufNum, DexImage & img, int iZ = 0 )

Reads the specified transport buffer into the passed in DexImage object at the passed in plane.

**Note:** GetCapturedBuffer can be used to get the number of the lastest buffer to be filled.

**Parameters**
- **bufNum** The index of the transport buffer to read from.
- **img** The DexImage object that the image will be written into.
- **iZ** The plane that the image should be written to in the DexImage object (defaults to 0).

**Exceptions**
- DexelaException
int DexelaDetector::ReadRegister ( int  address,
                                   int  sensorNum = 1
                               )

Reads the specified register from the detector. The sensor number corresponds to the desired sensor from which to read the register. The SensorNumber will default to 1 (master-sensor) if not specified otherwise by the user.

**Parameters**
- **address** The address of the desired register to read.
- **sensorNum** The sensor number from which to read the register (defaults to master-sensor).

**Returns**
The integer value of the register.

**Exceptions**
- DexelaException

---

void DexelaDetector::SetBinningMode ( bins  flag )

Sets the binning mode of the detector.

**Parameters**
- **flag** The bins enumeration member to be set.

**Exceptions**
- DexelaException

---

void DexelaDetector::SetCallback ( IMAGE_CALLBACK  func )

Sets the user defined callback function to be called for every image arrival event.

**Parameters**
func The call back function (**IMAGE_CALLBACK**) to be called for every image arrival event.

**Exceptions**

**DexelaException**

```cpp
void DexelaDetector::SetExposureMode ( ExposureModes mode )
```

Sets the ExposureMode parameter of the detector

**Parameters**

- **mode** The **ExposureModes** enumeration member to be set.

**Exceptions**

**DexelaException**

```cpp
void DexelaDetector::SetExposureTime ( float timems )
```

Sets the exposure time parameter of the detector

**Parameters**

- **timems** The exposure time (in milliseconds) to be set. **Note:** if you attempt to set an exposure time smaller than the current read-out time for the detector, it will be set to minimum.

**Exceptions**

**DexelaException**

```cpp
void DexelaDetector::SetFullWellMode ( FullWellModes fwm )
```

Sets the full well mode parameter of the detector.

**Parameters**
The FullWellModes enumeration member to be set.

Exceptions

DexelaException

void DexelaDetector::SetGapTime ( float timems )

Sets the gap-time setting of the detector. When run in Frame_Rate_exposure mode the detector will insert this gap period between consecutive frames in an image sequence.

Note: The minimum time for the gap-time setting is the current readout-time for the detector. Attempting to write anything smaller to the detector will result in a gap-time equal to the readout-time.

Parameters

  timems The gap-time in ms.

Exceptions

  DexelaException

void DexelaDetector::SetNumOfExposures ( int num )

Sets the number of exposures to acquire after a trigger. This is only relevant in Sequence_Exposure and Frame_Rate_exposure modes of operation.

Parameters

  num The number of exposures to acquire for each trigger received.

Exceptions

  DexelaException

void DexelaDetector::SetPreProgrammedExposureTimes ( int numE )
This method will set the exposure times for pre-programmed exposure.

**Parameters**

- **numExposures**
  The number of exposures to set. This number should correspond to the number of exposures that will be acquired in pre-programmed exposure mode and should be between 2-4.

- **exposuretimes_ms**
  An array of floating point numbers representing exposure times for pre-programmed exposure. The number of exposure times should correspond to the numExposures parameter.

**Exceptions**

- **DexelaException**

---

```cpp
void DexelaDetector::SetReadoutMode (ReadoutModes mode)
```

Sets the ReadoutMode parameter of the detector.

**Parameters**

- **mode**
  The `ReadoutModes` enumeration member to be set.

**Exceptions**

- **DexelaException**

---

```cpp
void DexelaDetector::SetSlowed (bool flag)
```

This method can specify to the API that the detector being used is a slowed-down detector (e.g. mammo detector). This should not be necessary as the API should be able to determine most of the time whether the firmware version is a slowed down one. However, for certain older detectors/firmwares this may not be possible. In this case this method can be used to inform the library that the firmware...
is slowed down and it will use the correct read-out times.

**Parameters**

- `numExposures` A boolean flag indicating whether the detector is slowed down or not.

**Exceptions**

- `DexelaException`

```c
void DexelaDetector::SetTestMode ( BOOL SetTestOn )
```

Enables/disables test mode. The detector will output a generated test pattern if this mode is turned on.

**Parameters**

- `SetTestOn` if set to `true` The test pattern is turned on.

**Exceptions**

- `DexelaException`

```c
void DexelaDetector::SetTriggerSource ( ExposureTriggerSource ets )
```

Sets the trigger source setting on the detector

**Parameters**

- `ets` A member of the `ExposureTriggerSource` enumeration to be set to the detector

**Exceptions**

- `DexelaException`

```c
void DexelaDetector::Snap ( int buffer, int timeout )
```
Snaps an image into the specified buffer. **Note:** If the detector trigger source is set to `Internal_Software`, this call will automatically trigger the detector.

**Parameters**

- **buffer**  
  The buffer number to snap to. The number of available buffers can be found by calling the `GetNumBuffers` method.

- **timeout**  
  The amount of time (in ms) that the library will wait for an image before throwing a timeout exception.

**Exceptions**

- `DexelaException`

void DexelaDetector::SoftReset ( void )

Cycles the power on the detector

**Exceptions**

- `DexelaException`

void DexelaDetector::SoftwareTrigger ( )

Sends a trigger to the detector (will only work if the trigger source is set to `Internal_Software`)

**Exceptions**

- `DexelaException`

void DexelaDetector::StopCallback ( )

This function will terminate the callback loop (i.e. will wait for all spawned threads to finish executing).
**Exceptions**

*DexelaException*

---

```cpp
void DexelaDetector::ToggleGenerator ( BOOL onOff )
```

This function will control the pulse train.

**Note:** In order to use this mode the pulse generator must be enabled. See *EnablePulseGenerator*.

**Parameters**

- **onOff** Boolean that will control the state of the pulse train (true = on, false = off).

**Exceptions**

*DexelaException*

---

```cpp
void DexelaDetector::WaitImage ( int timeout )
```

This function will wait for the specified amount of time for an image to arrive.

**Note:** If the image arrives before then it will return as soon as it does (i.e. it won't wait for the duration of the timeout period). If the image does not arrive in the specified time a *DexelaException* will be thrown.

**Parameters**

- **timeout** The timeout period (in ms) for which to wait before throwing a *DexelaException*.

**Exceptions**

*DexelaException*

---

```cpp
void DexelaDetector::WriteBuffer ( int bufNum, byte * buffer )
```
Writes data to the specified transport buffer.

**Parameters**
- `bufNum` The index of the transport buffer to write to.
- `buffer` The user-created data (byte*) buffer to write to the transport buffer.

**Exceptions**
- `DexelaException`

```cpp
void DexelaDetector::WriteRegister ( int address,  
    int value,  
    int sensorNum = 0  
)
```

Writes the value to the specified register from the detector. The sensor number corresponds to the desired sensor to which the value will be written. The `SensorNumber` will default to 0 (broadcast to all sensors) if not specified otherwise by the user.

**Parameters**
- `address` The address of the desired register to write to.

**Parameters**
- `value` The value to write into the register
- `sensorNum` The sensor number that the register will be written to. This defaults to 0 which is a broadcast to all detector sensors.

**Exceptions**
- `DexelaException`

The documentation for this class was generated from the following
files:

- DexelaDetector.h
- DexelaDetector.cpp
DexelaDetector API

DexelaDetectorCL
Class Reference

This class is used to control CameraLink Type Detectors. It will give access to functions that are not available to other interface-type detectors.

**Note:** For all standard detector function calls please see the DexelaDetector class (these functions are also available to DexelaDetectorCL objects) More...

```
#include <DexelaDetectorCL.h>
```

Inheritance diagram for DexelaDetectorCL:

```
DexelaDetector

DexelaDetectorCL
```
Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DexelaDetectorCL</strong> (*<em>DetectorInterface transport, int unit, const char <em>params</em></em>)</td>
<td>Constructor for <em>DexelaDetectorCL</em>. This version assumes you know the interface and the correct parameters to connect to the detector. More...</td>
</tr>
<tr>
<td><strong>DexelaDetectorCL</strong> (<strong>DevInfo &amp;devInfo</strong>)</td>
<td>Constructor for <em>DexelaDetectorCL</em>. Identical to the <em>DexelaDetectorCL constructor</em>, except with an additional check for the correct (CameraLink) interface. <strong>Note:</strong> A <strong>Exceptions</strong></td>
</tr>
<tr>
<td></td>
<td>DexelaException DevInfo object is not correct (i.e. CL)</td>
</tr>
<tr>
<td>virtual <strong>~DexelaDetectorCL</strong> (<strong>void</strong>)</td>
<td>Destructor for <em>DexelaDetectorCL</em>. More...</td>
</tr>
<tr>
<td><strong>void</strong> <strong>PowerCLInterface</strong> (<strong>bool flag</strong>)</td>
<td>Function to turn the CameraLink interface on and off More...</td>
</tr>
<tr>
<td><strong>void</strong> <strong>OpenBoard</strong> ()</td>
<td>Identical to the <em>OpenBoard</em> call. The only difference is a check to make sure the detector has the correct (CL) interface.</td>
</tr>
</tbody>
</table>
void **OpenBoard** *(int NumBufs)*

Identical to the **OpenBoard** call. The only difference is a check to make sure the detector has the correct (CL) interface. More...

---

**Public Member Functions inherited from** **DexelaDetector**

**DexelaDetector** *(DevInfo &devInfo)*

Constructor for **DexelaDetector**. This version uses the **DevInfo** struct returned from a **GetDevice**, **GetDeviceGE** or **GetDeviceCL** call. More...

**DexelaDetector** *(DetectorInterface transport, int unit, const char *params)*

Constructor for **DexelaDetector**. This version assumes you know the interface and the correct parameters to connect to the detector. More...

**virtual**  **~DexelaDetector** *(void)*

Destructor for **DexelaDetector**. More...

---

void **OpenBoard** *(int NumBufs)*

Opens the connection to the detector and sets the number of buffers to use/allocate. Every open should be matched with a close to free resources. More...

---

void **CloseBoard** *(void)*

Closes the connection to the detector. More...

---

int **GetBufferXdim** *(void)*

Get the x dimension of the transport buffer (in bytes) More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int GetBufferYdim (void)</strong></td>
<td>Get the y dimension of the transport buffer (in pixels)</td>
</tr>
<tr>
<td><strong>int GetNumBuffers (void)</strong></td>
<td>Get the number of internal buffers that are currently allocated for the detector.</td>
</tr>
<tr>
<td><strong>int GetCapturedBuffer (void)</strong></td>
<td>Gets the number of the buffer just captured. This can be used to determine which buffer to read-out.</td>
</tr>
<tr>
<td><strong>int GetFieldCount (void)</strong></td>
<td>Gets the number of fields(frames) captured so far.</td>
</tr>
<tr>
<td>*<em>void ReadBuffer (int bufNum, byte <em>buffer)</em></em></td>
<td>Reads the specified transport buffer into the passed in buffer (byte*). Note: GetCapturedBuffer can be used to get the number of the lastest buffer to be filled.</td>
</tr>
<tr>
<td><strong>void ReadBuffer (int bufNum, DexImage &amp;img, int iZ=0)</strong></td>
<td>Reads the specified transport buffer into the passed in DexImage object at the passed in plane. Note: GetCapturedBuffer can be used to get the number of the lastest buffer to be filled.</td>
</tr>
<tr>
<td>*<em>void WriteBuffer (int bufNum, byte <em>buffer)</em></em></td>
<td>Writes data to the specified transport buffer</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>void SetFullWellMode(FullWellModes fwm)</code></td>
<td>Sets the full well mode parameter of the detector.  More...</td>
</tr>
<tr>
<td><code>void SetExposureMode(ExposureModes mode)</code></td>
<td>Sets the ExposureMode parameter of the detector  More...</td>
</tr>
<tr>
<td><code>void SetExposureTime(float timems)</code></td>
<td>Sets the exposure time parameter of the detector  More...</td>
</tr>
<tr>
<td><code>void SetBinningMode(bins flag)</code></td>
<td>Sets the binning mode of the detector.  More...</td>
</tr>
<tr>
<td><code>void SetTestMode(BOOL SetTestOn)</code></td>
<td>Enables/disables test mode. The detector will output a generated test pattern if this mode is turned on.  More...</td>
</tr>
<tr>
<td><code>void SetTriggerSource(ExposureTriggerSource ets)</code></td>
<td>Sets the trigger source setting on the detector  More...</td>
</tr>
<tr>
<td><code>void SetNumOfExposures(int num)</code></td>
<td>Sets the number of exposures to acquire after a trigger. This is only relevant in Sequence_Exposure and Frame_Rate_exposure modes of operation.  More...</td>
</tr>
<tr>
<td><code>int GetNumOfExposures()</code></td>
<td>Gets the number of exposures setting</td>
</tr>
</tbody>
</table>
from the detector. This is only relevant in **Sequence_Exposure** and **Frame_Rate_exposure** modes of operation. More...

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <strong>SetGapTime</strong> (float timems)</td>
<td>Sets the gap-time setting of the detector. When run in <strong>Frame_Rate_exposure</strong> mode the detector will insert this gap period between consecutive frames in an image sequence. <strong>Note:</strong> The minimum time for the gap-time setting is the current readout-time for the detector. Attempting to write anything smaller to the detector will result in a gap-time equal to the readout-time. More...</td>
</tr>
<tr>
<td>float <strong>GetGapTime</strong> ()</td>
<td>Gets the current gap-time setting of the detector. When run in <strong>Frame_Rate_exposure</strong> mode the detector will insert this gap period between consecutive frames in an image sequence. <strong>Note:</strong> The minimum time for the gap-time setting is the current readout-time for the detector. Attempting to write anything smaller to the detector will result in a gap-time equal to the readout-time. More...</td>
</tr>
<tr>
<td>bool <strong>IsConnected</strong> ()</td>
<td>Check to see if the connection to the detector is open (i.e. <strong>OpenBoard</strong>) More...</td>
</tr>
<tr>
<td><strong>ExposureModes</strong> <strong>GetExposureMode</strong> ()</td>
<td>Gets the ExposureMode parameter of the detector. More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>float \textbf{GetExposureTime} ()</td>
<td>Gets the exposure time parameter of the detector (ms). \textit{More...}</td>
</tr>
<tr>
<td><strong>DetStatus</strong> \textbf{GetDetectorStatus} ()</td>
<td>Returns the current settings of the detector in the form of a <strong>DetStatus</strong> object. \textit{More...}</td>
</tr>
<tr>
<td><strong>ExposureTriggerSource</strong> \textbf{GetTriggerSource} ()</td>
<td>Gets the current trigger source setting from the detector \textit{More...}</td>
</tr>
<tr>
<td>BOOL \textbf{GetTestMode} ()</td>
<td>Gets the current state of the detector test mode (on/off) \textit{More...}</td>
</tr>
<tr>
<td><strong>FullWellModes</strong> \textbf{GetFullWellMode} ()</td>
<td>Gets the current detector well-mode. \textit{More...}</td>
</tr>
<tr>
<td><strong>bins</strong> \textbf{GetBinningMode} ()</td>
<td>Gets the current state of the detector binning mode \textit{More...}</td>
</tr>
<tr>
<td>int \textbf{GetSerialNumber} ()</td>
<td>Gets the detector serial number. \textit{More...}</td>
</tr>
<tr>
<td>int \textbf{GetModelNumber} ()</td>
<td>Gets the detector model number. \textit{More...}</td>
</tr>
<tr>
<td>int \textbf{GetFirmwareVersion} ()</td>
<td>Gets the detector firmware version number. \textit{More...}</td>
</tr>
<tr>
<td>void \textbf{GetFirmwareBuild} (int &amp;iDayAndMonth, int &amp;iYear, int &amp;iTime)</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DetectorInterface.GetTransportMethod()</td>
<td>Returns the communication method (i.e. interface) for the detector object.</td>
</tr>
<tr>
<td>double GetReadOutTime()</td>
<td>This method will return the read-out time of the detector (in ms) for its current binning mode.</td>
</tr>
<tr>
<td>bool IsCallbackActive()</td>
<td>This method will inform the user if the callback mode (i.e. background thread) is currently active.</td>
</tr>
<tr>
<td>bool IsLive()</td>
<td>This method will inform the user if detector is currently in Live mode.</td>
</tr>
<tr>
<td>void Snap(int buffer, int timeout)</td>
<td>Snaps an image into the specified buffer.</td>
</tr>
<tr>
<td>int ReadRegister(int address, int sensorNum=1)</td>
<td>Reads the specified register from the detector. The sensor number corresponds to the desired sensor from which to read the register. The SensorNumber will default to 1 (master-sensor) if not specified otherwise by the user.</td>
</tr>
</tbody>
</table>
### `WriteRegister` (int address, int value, int sensorNum=0)
 Writes the value to the specified register from the detector. The sensor number corresponds to the desired sensor to which the value will be written. The SensorNumber will default to 0 (broadcast to all sensors) if not specified otherwise by the user. More...

### `ClearCameraBuffer` (int i)
 Clears (i.e. zero-out) the specified camera buffer. More...

### `ClearBuffers`
 Clears all the camera buffers More...

### `LoadSensorConfigFile` (char *filename)
 Loads the sensor configuration file into the detector. This file will write values to the ADC offset registers for each sensor in the detector. More...

### `SoftReset`
 Cycles the power on the detector More...

### `GoLiveSeq` (int start, int stop, int numBuf)
 Sets the host computer up to be ready to receive images into the specified buffer range. More...

### `GoLiveSeq`
 Sets the host computer up to be ready to receive images. This call will use all available buffers in a circular fashion (i.e. ring-buffer). More...

### `GoUnLive`

Exits live mode. The host computer will no longer be ready to receive transmitted images. More...

void **SoftwareTrigger** ()
Sends a trigger to the detector (will only work if the trigger source is set to \texttt{Internal\_Software}) More...

void **EnablePulseGenerator** (float frequency)
This function will enable the pulse generator software trigger signal. In this mode the software trigger can be continuously sent to the detector at the desired frequency.

**Note:** In order to use this mode the trigger source should be set to \texttt{Internal\_Software}

**Note2:** To actually enable the pulse train you must call \texttt{ToggleGenerator}. A \texttt{SoftwareTrigger} call will not work when in this mode. More...

void **EnablePulseGenerator** ()
This function is identical to \texttt{EnablePulseGenerator}, except that the frequency of the pulse train is set automatically. The frequency will be set such as to ensure continuous image acquisition for the current detector/binning mode.

**Note:** In order to use this mode the trigger source should be set to \texttt{Internal\_Software}

**Note2:** To actually enable the pulse train you must call \texttt{ToggleGenerator}. A \texttt{SoftwareTrigger} call will not work when in this mode. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DisablePulseGenerator()</code></td>
<td>This function will disable Pulse Generator mode. After calling this you should be able to use the <code>SoftwareTrigger</code> call.</td>
</tr>
<tr>
<td><code>ToggleGenerator(BOOL onOff)</code></td>
<td>This function will control the pulse train. <strong>Note:</strong> In order to use this mode the pulse generator must be enabled. See <code>EnablePulseGenerator</code>.</td>
</tr>
<tr>
<td><code>WaitImage(int timeout)</code></td>
<td>This function will wait for the specified amount of time for an image to arrive. <strong>Note:</strong> If the image arrives before then it will return as soon as it does (i.e. it won't wait for the duration of the timeout period). If the image does not arrive in the specified time a <code>DexelaException</code> will be thrown.</td>
</tr>
<tr>
<td><code>SetCallback(IMAGE_CALLBACK func)</code></td>
<td>Sets the user defined callback function to be called for every image arrival event.</td>
</tr>
<tr>
<td><code>StopCallback()</code></td>
<td>This function will terminate the callback loop (i.e. will wait for all spawned threads to finish executing).</td>
</tr>
<tr>
<td><code>CheckForCallbackError()</code></td>
<td>This function will check to see if any errors have occurred in the background thread that is running when using</td>
</tr>
</tbody>
</table>
callbacks. This thread is activated after a call to `SetCallback` and terminated with a call to `StopCallback`. If no error has occurred this method will just return. If an error has occurred a `DexelaException` will be thrown. More...

```cpp
void CheckForLiveError ()
This function will check to see if any errors have occurred in the background thread that is running when using live-mode. This thread is activated after a call to `GoLiveSeq` and terminated with a call to `GoUnLive`. If no error has occurred this method will just return. If an error has occurred a `DexelaException` will be thrown. More...
```

```cpp
void SetPreProgrammedExposureTimes (int numExposures, float *exposuretimes_ms)
This method will set the exposure times for pre-programmed exposure mode. More...
```

```cpp
void SetROICoordinates (unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight)
This method set the coordinates of the ROI when detector runs in ROI mode.

Parameters

- **usStartColumn**: Sensor column to start the ROI read out from.
- **usStartRow**: Sensor width to start the ROI read out from.
- **usROIWidth**: Width (number of
Exceptions
DexelaException

void GetROICoordinates (unsigned short &usStartColumn, unsigned short &usStartRow, unsigned short &usROIWidth, unsigned short &usROIHeight)
This method retrieves the coordinates of the region of interest (ROI) set within the detector. More...

void EnableROIMode (bool bEnableROI)
This method activates or deactivates the ROI mode of the detector. More...

bool GetROIState ()
This method retrieves the enabled state of the region of interest of the detector. More...

unsigned short GetSensorHeight (unsigned short uiSensorID=1)
Gets the height of the sensor in pixels. More...

unsigned short GetSensorWidth (unsigned short uiSensorID=1)
Gets the width of the sensor in pixels. More...
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool IsFrameCntWithinImage()</code></td>
<td>Checks if framecounter is displayed within the image data (being the 2nd pixel).</td>
</tr>
<tr>
<td><code>void EnableFrameCntWithinImage(unsigned short usEnable)</code></td>
<td>Enables displaying the framecounter in the image (being the 2nd pixel).</td>
</tr>
<tr>
<td><code>void SetSlowed(bool flag)</code></td>
<td>This method can specify to the api that the detector being used is a slowed-down detector (e.g. mammo detector). This should not be necessary as the API should be able to determine most of the time whether the firmware version is a slowed down one. However, for certain older detectors/firmwares this may not be possible. In this case this method can be used to inform the library that the firmware is slowed down and it will use the correct read-out times.</td>
</tr>
<tr>
<td><code>void SetReadoutMode(ReadoutModes mode)</code></td>
<td>Sets the ReadoutMode parameter of the detector.</td>
</tr>
<tr>
<td><code>ReadoutModes GetReadoutMode()</code></td>
<td>Gets the ReadoutModes parameter of the detector.</td>
</tr>
<tr>
<td><code>int QueryReadoutMode(ReadoutModes mode)</code></td>
<td>Query the detector to see if the desired readout mode is present (i.e. available). Note: Older detectors may not support</td>
</tr>
</tbody>
</table>
the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...

int QueryExposureMode (ExposureModes mode)
Query the detector to see if the desired exposure mode is present (i.e. available).

Note: Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...

int QueryTriggerSource (ExposureTriggerSource ets)
Query the detector to see if the desired trigger source is present (i.e. available).

Note: Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...
int QueryFullWellMode (FullWellModes fwm)

Query the detector to see if the desired full-well mode is present (i.e. available). **Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...

int QueryBinningMode (bins flag)

Query the detector to see if the desired binning mode is present (i.e. available). **Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. More...
### Additional Inherited Members

- **Protected Attributes inherited from** `DexelaDetector`
  - `boost::shared_ptr< baseDetector > base`
  - `boost::shared_ptr< gigEDetector > gigeDet`
  - `boost::shared_ptr< camLinkDetector > clDet`
Detailed Description

This class is used to control CameraLink Type Detectors. It will give access to functions that are not available to other interface-type detectors.

**Note:** For all standard detector function calls please see the DexelaDetector class (these functions are also available to DexelaDetectorCL objects)
Constructor & Destructor Documentation

DexelaDetectorCL::DexelaDetectorCL ( DetectorInterface transport
int unit,
const char * params )

Constructor for DexelaDetectorCL. This version assumes you know the interface and the correct parameters to connect to the detector.

Parameters

- **transport** The DetectorInterface for the detector (i.e. CL)
- **unit** The unit number for CL type detectors.
- **params** The parameter string for connection to the detector. For CL detectors this parameter will be ignored.

Exceptions

DexelaException

---

DexelaDetectorCL::DexelaDetectorCL ( DevInfo & devInfo )

Constructor for DexelaDetectorCL. Identical to the DexelaDetector constructor, except with an additional check for the correct (CameraLink) interface.

**Note:** A

Exceptions

DexelaException is thrown if the interface of the DevInfo object is not correct (i.e. CL)

Parameters

- **devInfo** The DevInfo object for the desired detector. This can be obtained from the GetDeviceCL method
Exceptions

DexelaException

DexelaDetectorCL::~DexelaDetectorCL ( void )

Destructor for DexelaDetectorCL.
Member Function Documentation

**void DexelaDetectorCL::OpenBoard ( )**

Identical to the `OpenBoard` call. The only difference is a check to make sure the detector has the correct (CL) interface.

**Exceptions**
DexelaException

Reimplemented from DexelaDetector.

**void DexelaDetectorCL::OpenBoard ( int NumBufs )**

Identical to the `OpenBoard` call. The only difference is a check to make sure the detector has the correct (CL) interface.

**Parameters**
NumBufs Number of buffers to use/allocate

**Exceptions**
DexelaException

**void DexelaDetectorCL::PowerCLInterface ( bool flag )**

Function to turn the CameraLink interface on and off

**Parameters**
flag If flag is true the interface will be turned on otherwise it will be turned off

**Exceptions**
DexelaException

The documentation for this class was generated from the following files:

- DexelaDetectorCL.h
- DexelaDetectorCL.cpp

Generated on Tue Nov 25 2014 10:22:44 for DexelaDetector API by doxygen 1.8.7
DexelaDetector API

DexelaDetectorGE
Class Reference

This class is used to control GigE Type Detectors. It will give access to functions that are not available to other interface-type detectors. **Note:** For all standard detector function calls please see the DexelaDetector class (these functions are also available to DexelaDetectorGE objects) More...

#include <DexelaDetectorGE.h>

Inheritance diagram for DexelaDetectorGE:

```
DexelaDetectorGE
     |
     V
DexelaDetector
```
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><code>DexelaDetectorGE (DevInfo &amp;devInfo)</code></strong></td>
<td>Constructor for <strong>DexelaDetectorGE</strong>. Identical to the <strong>DexelaDetector constructor</strong>, except with an additional check for the correct (GIGE) interface. <strong>Note:</strong> A <strong>DexelaException</strong> <strong>DevInfo</strong> object is thrown if the interface of the <strong>DexelaException</strong> <strong>DevInfo</strong> object is not correct (i.e. GIGE)</td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td></td>
</tr>
<tr>
<td><strong><code>DexelaDetectorGE (DetectorInterface transport, int unit, const char *params)</code></strong></td>
<td>Constructor for <strong>DexelaDetectorGE</strong>. This version assumes you know the interface and the correct parameters to connect to the detector. <strong>More...</strong></td>
</tr>
<tr>
<td><strong>virtual</strong> <strong>~DexelaDetectorGE (void)</strong></td>
<td>Destructor for <strong>DexelaDetectorGE</strong>. <strong>More...</strong></td>
</tr>
<tr>
<td><strong>void</strong> <strong>SetPersistentIPAddress</strong> (int firstByte, int secondByte, int thirdByte, int fourthByte)</td>
<td>Function for setting a new persistent IP address for the detector. After power-cycling the detector it should come up with the desired IP address <strong>More...</strong></td>
</tr>
<tr>
<td><strong>void</strong> <strong>OpenBoard ()</strong></td>
<td></td>
</tr>
</tbody>
</table>
Identical to the `OpenBoard` call. The only difference is a check to make sure the detector has the correct (GIGE) interface.

```c
void OpenBoard (int NumBufs)
```

Identical to the `OpenBoard` call. The only difference is a check to make sure the detector has the correct (GIGE) interface.

```c
Public Member Functions inherited from DexelaDetector
```

```c
DexelaDetector (DevInfo &devInfo)
```

Constructor for `DexelaDetector`. This version uses the `DevInfo` struct returned from a `GetDevice`, `GetDeviceGE` or `GetDeviceCL` call.

```c
DexelaDetector (DetectorInterface transport, int unit, const char *params)
```

Constructor for `DexelaDetector`. This version assumes you know the interface and the correct parameters to connect to the detector.

```c
virtual ~DexelaDetector (void)
```

Destructor for `DexelaDetector`.

```c
void OpenBoard (int NumBufs)
```

Opens the connection to the detector and sets the number of buffers to use/allocate. Every open should be matched with a close to free resources.

```c
void CloseBoard ()
```

Closes the connection to the detector.
int `GetBufferXdim` (void)
Get the x dimension of the transport buffer (in bytes) More...

int `GetBufferYdim` (void)
Get the y dimension of the transport buffer (in pixels) More...

int `GetNumBuffers` (void)
Get the number of internal buffers that are currently allocated for the detector. More...

int `GetCapturedBuffer` (void)
Gets the number of the buffer just captured. This can be used to determine which buffer to read-out. More...

int `GetFieldCount` (void)
Gets the number of fields (frames) captured so far. More...

void `ReadBuffer` (int bufNum, byte *buffer)
Reads the specified transport buffer into the passed in buffer (byte*). Note: `GetCapturedBuffer` can be used to get the number of the lastest buffer to be filled. More...

void `ReadBuffer` (int bufNum, `DexImage` &img, int iZ=0)
Reads the specified transport buffer into the passed in `DexImage` object at the passed in plane. Note: `GetCapturedBuffer` can be used to get the number of the lastest buffer to be filled. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void WriteBuffer</td>
<td><code>WriteBuffer (int bufNum, byte *buffer)</code> Writes data to the specified transport buffer.</td>
</tr>
<tr>
<td>void SetFullWellMode</td>
<td><code>SetFullWellMode (FullWellModes fwm)</code> Sets the full well mode parameter of the detector.</td>
</tr>
<tr>
<td>void SetExposureMode</td>
<td><code>SetExposureMode (ExposureModes mode)</code> Sets the ExposureMode parameter of the detector More...</td>
</tr>
<tr>
<td>void SetExposureTime</td>
<td><code>SetExposureTime (float timems)</code> Sets the exposure time parameter of the detector More...</td>
</tr>
<tr>
<td>void SetBinningMode</td>
<td><code>SetBinningMode (bins flag)</code> Sets the binning mode of the detector. More...</td>
</tr>
<tr>
<td>void SetTestMode</td>
<td><code>SetTestMode (BOOL SetTestOn)</code> Enables/disables test mode. The detector will output a generated test pattern if this mode is turned on. More...</td>
</tr>
<tr>
<td>void SetTriggerSource</td>
<td><code>SetTriggerSource (ExposureTriggerSource ets)</code> Sets the trigger source setting on the detector More...</td>
</tr>
<tr>
<td>void SetNumOfExposures</td>
<td><code>SetNumOfExposures (int num)</code> Sets the number of exposures to acquire after a trigger. This is only relevant in Sequence_Exposure and Frame_Rate_exposure modes of operation. More...</td>
</tr>
</tbody>
</table>
int GetNumOfExposures ()
Gets the number of exposures setting from the detector. This is only relevant in Sequence_Exposure and Frame_Rate_exposure modes of operation. More...

void SetGapTime (float timems)
Sets the gap-time setting of the detector. When run in Frame_Rate_exposure mode the detector will insert this gap period between consecutive frames in an image sequence.
Note: The minimum time for the gap-time setting is the current readout-time for the detector. Attempting to write anything smaller to the detector will result in a gap-time equal to the readout-time. More...

float GetGapTime ()
Gets the current gap-time setting of the detector. When run in Frame_Rate_exposure mode the detector will insert this gap period between consecutive frames in an image sequence.
Note: The minimum time for the gap-time setting is the current readout-time for the detector. Attempting to write anything smaller to the detector will result in a gap-time equal to the readout-time. More...

bool IsConnected ()
Check to see if the connection to the detector is open (i.e. OpenBoard) More...
<table>
<thead>
<tr>
<th><strong>ExposureModes</strong></th>
<th><strong>GetExposureMode</strong> ()</th>
<th>Gets the ExposureMode parameter of the detector. More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>float</strong></td>
<td><strong>GetExposureTime</strong> ()</td>
<td>Gets the exposure time parameter of the detector (ms). More...</td>
</tr>
<tr>
<td><strong>DetStatus</strong></td>
<td><strong>GetDetectorStatus</strong> ()</td>
<td>Returns the current settings of the detector in the form of a DetStatus object. More...</td>
</tr>
<tr>
<td><strong>ExposureTriggerSource</strong></td>
<td><strong>GetTriggerSource</strong> ()</td>
<td>Gets the current trigger source setting from the detector More...</td>
</tr>
<tr>
<td><strong>BOOL</strong></td>
<td><strong>GetTestMode</strong> ()</td>
<td>Gets the current state of the detector test mode (on/off) More...</td>
</tr>
<tr>
<td><strong>FullWellModes</strong></td>
<td><strong>GetFullWellMode</strong> ()</td>
<td>Gets the current detector well-mode. More...</td>
</tr>
<tr>
<td><strong>bins</strong></td>
<td><strong>GetBinningMode</strong> ()</td>
<td>Gets the current state of the detector binning mode More...</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td><strong>GetSerialNumber</strong> ()</td>
<td>Gets the detector serial number. More...</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td><strong>GetModelNumber</strong> ()</td>
<td>Gets the detector model number. More...</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td><strong>GetFirmwareVersion</strong> ()</td>
<td>Gets the detector firmware version</td>
</tr>
</tbody>
</table>
### void GetFirmwareBuild (int &iDayAndMonth, int &iYear, int &iTime)

Gets the detector firmware build date.

**Note:** This feature may not be supported on older detectors. More...

### DetectorInterface GetTransportMethod ()

Returns the communication method (i.e. interface) for the detector object. More...

### double GetReadOutTime ()

This method will return the read-out time of the detector (in ms) for its current binning mode. More...

### bool IsCallbackActive ()

This method will inform the user if the callback mode (i.e. background thread) is currently active. More...

### bool IsLive ()

This method will inform the user if detector is currently in Live mode. More...

### void Snap (int buffer, int timeout)

Snaps an image into the specified buffer.

**Note:** If the detector trigger source is set to **Internal_Software**, this call will automatically trigger the detector. More...

### int ReadRegister (int address, int sensorNum=1)

Reads the specified register from the detector. The sensor number corresponds to the desired sensor from which to read the register. The SensorNumber will
WriteRegister (int address, int value, int sensorNum=0)
Writes the value to the specified register from the detector. The sensor number corresponds to the desired sensor to which the value will be written. The SensorNumber will default to 0 (broadcast to all sensors) if not specified otherwise by the user. More...

ClearCameraBuffer (int i)
Clears (i.e. zero-out) the specified camera buffer. More...

ClearBuffers ()
Clears all the camera buffers More...

LoadSensorConfigFile (char *filename)
Loads the sensor configuration file into the detector. This file will write values to the ADC offset registers for each sensor in the detector. More...

SoftReset (void)
Cycles the power on the detector More...

GoLiveSeq (int start, int stop, int numBuf)
Sets the host computer up to be ready to recieve images into the specified buffer range. More...

GoLiveSeq ()
Sets the host computer up to be ready to recieve images. This call will use all
available buffers in a circular fashion (i.e. ring-buffer). More...

void **GoUnLive** ()
Exits live mode. The host computer will no longer be ready to receive transmitted images. More...

void **SoftwareTrigger** ()
Sends a trigger to the detector (will only work if the trigger source is set to **Internal_Software**) More...

void **EnablePulseGenerator** (float frequency)
This function will enable the pulse generator software trigger signal. In this mode the software trigger can be continuously sent to the detector at the desired frequency.

**Note:** In order to use this mode the trigger source should be set to **Internal_Software**

**Note2:** To actually enable the pulse train you must call **ToggleGenerator**. A **SoftwareTrigger** call will not work when in this mode. More...

void **EnablePulseGenerator** ()
This function is identical to **EnablePulseGenerator**, except that the frequency of the pulse train is set automatically. The frequency will be set such as to ensure continuous image acquisition for the current detector/binning mode.

**Note:** In order to use this mode the trigger source should be set to **Internal_Software**
### Note2:
To actually enable the pulse train you must call **ToggleGenerator**. A **SoftwareTrigger** call will not work when in this mode. More...

```c
void DisablePulseGenerator ()
This function will disable Pulse Generator mode. After calling this you should be able to use the **SoftwareTrigger** call. More...
```

```c
void ToggleGenerator (BOOL onOff)
This function will control the pulse train. **Note:** In order to use this mode the pulse generator must be enabled. See **EnablePulseGenerator**. More...
```

```c
void WaitImage (int timeout)
This function will wait for the specified amount of time for an image to arrive. **Note:** If the image arrives before then it will return as soon as it does (i.e. it won't wait for the duration of the timeout period). If the image does not arrive in the specified time a **DexelaException** will be thrown. More...
```

```c
void SetCallback (IMAGE_CALLBACK func)
Sets the user defined callback function to be called for every image arrival event. More...
```

```c
void StopCallback ()
This function will terminate the callback loop (i.e. will wait for all spawned threads to finish executing). More...
```

```c
void CheckForCallbackError ()
```
This function will check to see if any errors have occurred in the background thread that is running when using callbacks. This thread is activated after a call to `SetCallback` and terminated with a call to `StopCallback`. If no error has occurred this method will just return. If an error has occurred a `DexelaException` will be thrown. More...

```cpp
void CheckForLiveError()

This function will check to see if any errors have occurred in the background thread that is running when using live-mode. This thread is activated after a call to `GoLiveSeq` and terminated with a call to `GoUnLive`. If no error has occurred this method will just return. If an error has occurred a `DexelaException` will be thrown. More...
```

```cpp
void SetPreProgrammedExposureTimes(int numExposures, float *exposuretimes_ms)

This method will set the exposure times for pre-programmed exposure mode. More...
```

```cpp
void SetROICoordinates(unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight)

This method set the coordinates of the ROI when detector runs in ROI mode.

Parameters

- **usStartColumn**
  - Sensor column to start the ROI read out from.

- **usStartRow**
  - Sensor width to
**usROIWidth**
- Width (number of columns) of the ROI.

**usROIHeight**
- Height (number of columns) of the ROI.

**Exceptions**
- DexelaException

---

**GetROICoordinates** (unsigned short &usStartColumn, unsigned short &usStartRow, unsigned short &usROIWidth, unsigned short &usROIHeight)
This method retrieves the coordinates of the region of interest (ROI) set within the detector. [More...](#)

**EnableROIState** (bool bEnableROI)
This method activates or deactivates the ROI mode of the detector. [More...](#)

**GetROIState** ()
This method retrieves the enabled state of the region of interest of the detector. [More...](#)

**GetSensorHeight** (unsigned short uiSensorID=1)
Gets the height of the sensor in pixels. [More...](#)
**unsigned short** `GetSensorWidth` (unsigned short `uiSensorID=1`)
Gets the width of the sensor in pixels.
*More...*

**bool** `IsFrameCntWithinImage` ()
Checks if framecounter is displayed within the image data (being the 2nd pixel).
*More...*

**void** `EnableFrameCntWithinImage` (unsigned short `usEnable`)
Enables displaying the framecounter in the image (being the 2nd pixel).
*More...*

**void** `SetSlowed` (bool `flag`)
This method can specify to the api that the detector being used is a slowed-down detector (e.g. mammo detector). This should not be necessary as the API should be able to determine most of the time whether the firmwarwe version is a slowed down one. However, for certain older detectors/firmwares this may not be possible. In this case this method can be used to inform the library that the firmware is slowed down and it will use the correct read-out times.
*More...*

**void** `SetReadoutMode` (**ReadoutModes** `mode`)
Sets the ReadoutMode parameter of the detector.
*More...*

**ReadoutModes** `GetReadoutMode` ()
Gets the ReadoutModes parameter of the detector.
*More...*
**int QueryReadoutMode (ReadoutModes mode)**
Query the detector to see if the desired readout mode is present (i.e. available).

**Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work.  

**More...**

**int QueryExposureMode (ExposureModes mode)**
Query the detector to see if the desired exposure mode is present (i.e. available).

**Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work.  

**More...**

**int QueryTriggerSource (ExposureTriggerSource ets)**
Query the detector to see if the desired trigger source is present (i.e. available).

**Note:** Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int QueryFullWellMode(FullWellModes fwm)</code></td>
<td>Query the detector to see if the desired full-well mode is present (i.e. available). <strong>Note:</strong> Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. <strong>More...</strong></td>
</tr>
<tr>
<td><code>int QueryBinningMode(bins flag)</code></td>
<td>Query the detector to see if the desired binning mode is present (i.e. available). <strong>Note:</strong> Older detectors may not support the querying of features. In this case a value of -1 will be returned indicating that it is unknown whether the feature is present. In this case it is possible that the feature is present but that the detector is unable to report so. It may still be possible to use the feature in this case but no guarantees can be made on whether or not it will work. <strong>More...</strong></td>
</tr>
</tbody>
</table>
### Additional Inherited Members

- **Protected Attributes inherited from** `DexelaDetector`
  - `boost::shared_ptr< baseDetector >` **base**
  - `boost::shared_ptr< gigEDetector >` **gigeDet**
  - `boost::shared_ptr< camLinkDetector >` **clDet**
Detailed Description

This class is used to control GigE Type Detectors. It will give access to functions that are not available to other interface-type detectors. **Note:** For all standard detector function calls please see the **DexelaDetector** class (these functions are also available to **DexelaDetectorGE** objects)
Constructor & Destructor Documentation

DexelaDetectorGE::DexelaDetectorGE (DevInfo & devInfo)

Constructor for DexelaDetectorGE. Identical to the DexelaDetector constructor, except with an additional check for the correct (GIGE) interface.

Note: A

Exceptions

DexelaException is thrown if the interface of the DevInfo object is not correct (i.e. GIGE)

Parameters

devInfo The DevInfo object for the desired detector. This can be obtained from the GetDeviceGE method

Exceptions

DexelaException

DexelaDetectorGE::DexelaDetectorGE (DetectorInterface transport, int unit, const char * params)

Constructor for DexelaDetectorGE. This version assumes you know the interface and the correct parameters to connect to the detector.

Parameters

transport The DetectorInterface for the detector (i.e. GIGE)
unit For GIGE detectors this can be set to 0
params The parameter string for connection to the detector. For GIGE detectors this should be the detector IP address.
Exceptions

DexelaException

DexelaDetectorGE::~DexelaDetectorGE ( void )

Destructor for DexelaDetectorGE.

void DexelaDetectorGE::OpenBoard ( )

Identical to the OpenBoard call. The only difference is a check to make sure the detector has the correct (GIGE) interface.

Exceptions

DexelaException

Reimplemented from DexelaDetector.

void DexelaDetectorGE::OpenBoard ( int NumBufs )

Identical to the OpenBoard call. The only difference is a check to make sure the detector has the correct (GIGE) interface.

Parameters

NumBufs Number of buffers to use/allocate

Exceptions

DexelaException

void DexelaDetectorGE::SetPersistentIPAddress ( int firstByte,
                                          int secondByte,
                                          int thirdByte,
                                          int fourthByte )

Function for setting a new persistent IP address for the detector. After power-cycling the detector it should come up with the desired
IP address

**Parameters**

**firstByte**  The first byte of the desired IP address (e.g. 169 for the address 169.254.70.3)

**secondByte**  The second byte of the desired IP address (e.g. 254 for the address 169.254.70.3)

**thirdByte**  The third byte of the desired IP address (e.g. 70 for the address 169.254.70.3)

**fourthByte**  The fourth byte of the desired IP address (e.g. 3 for the address 169.254.70.3)

**Exceptions**

*DexelaException*

The documentation for this class was generated from the following files:

- **DexelaDetectorGE.h**
- **DexelaDetectorGE.cpp**
DexelaDetector API

DexelaException Class Reference

This class contains information about any possible error's in the API. In the event of a problem a DexelaException will be thrown. **Note:** It is suggested that you wrap your code in a try-catch block to ensure that if any errors occur you can detect (and properly handle them) in your code. [More...]

```c
#include <DexelaException.h>
```

Inheritance diagram for DexelaException:

```plaintext
exception

DexelaException
```
### Public Member Functions

**DexelaException** (const char *message, **Derr** code, int line, const char *filename, const char *function, int transportError, const char *transportMessage)

Constructor for the **DexelaException** Class. More...

**DexelaException** (const **DexelaException** &ex, const char *function)

Copy constructor for the **DexelaException** Class. More...

**~DexelaException** (void) throw ()

**DexelaException** destructor. More...

const char * **what** () const throw ()

Function for retrieving the exception's error message. More...

**Derr** **GetCode** ()

Function for retrieving the exception's **Derr** code. More...

int **GetTransportError** ()

Function for retrieving the exception's low-level transport error code. **Note:** This code (along with line-number, and filename) can be sent to PerkinElmer support for further information about possible causes of the exception. More...

const char * **GetFileName** ()

Function for retrieving the name of the (low-level) source file from which the exception was thrown. **Note:** This information (along with line-number, and transport error) can be sent to PerkinElmer support for further information about possible causes of the exception. More...
<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>GetLineNumber()</td>
<td>Function for retrieving line-number of the source of the (low-level) exception throw. <strong>Note:</strong> This number (along with transport-error and file-name) can be sent to PerkinElmer support for further information about possible causes of the exception. More...</td>
</tr>
<tr>
<td>const char *</td>
<td>GetFunctionName()</td>
<td>Function for retrieving the name of the (top-level) function from which the error was thrown. <strong>Note:</strong> This should help to find the function that is causing the exception. More...</td>
</tr>
<tr>
<td>const char *</td>
<td>GetTransportMessage()</td>
<td>Function for retrieving the (low-level) message from the transport library. <strong>Note:</strong> This message (along with transport-error, line number and file-name) can be sent to PerkinElmer support for further information about possible causes of the exception. More...</td>
</tr>
</tbody>
</table>
Static Public Member Functions

static void LoadErrorStrings (const char *filename)
Detailed Description

This class contains information about any possible error's in the API. In the event of a problem a `DexelaException` will be thrown. 
**Note:** It is suggested that you wrap your code in a try-catch block to ensure that if any errors occur you can detect (and properly handle them) in your code.
Constructor & Destructor Documentation

DexelaException::DexelaException ( const char * message, 
Derr code, 
int line, 
const char * filename, 
const char * function, 
int transportEr, 
const char * transportMessage )

Constructor for the DexelaException Class.

Parameters
- **message**  
The error message, detailing the source of the exception.
- **code**  
The Derr error code associated with the exception.
- **line**  
The (low-level) line-number of the source of the exception.
- **filename**  
The (low-level) name of the source file that was the source of the exception.
- **function**  
The (top-level) name of the function in which the exception was thrown.
- **transportEr**  
The (low-level) transport error that may associated with the exception.
- **transportMessage**  
The (low-level) transport layer error message. This may be empty depending on the source of the error.

DexelaException::DexelaException ( const DexelaException & ex, 
const char * fun...
Copy constructor for the DexelaException Class.

**Parameters**

- `ex` The DexelaException object from which to copy the low-level information about the source of the exception.
- `function` The top-level name of the function in which the exception was thrown.

```cpp
DexelaException::~DexelaException ( void )
    throw ( )
```

DexelaException destructor.
**Member Function Documentation**

**Derr** `DexelaException::GetCode ( )`

Function for retrieving the exception's **Derr** code.

**Returns**
- A member of the **Derr** enumeration detailing the exception's error code.

**const char * DexelaException::GetFileName ( )**

Function for retrieving the name of the (low-level) source file from which the exception was thrown.

**Note:** This information (along with line-number, and transport error) can be sent to PerkinElmer support for further information about possible causes of the exception.

**Returns**
- A string containing the name of the (low-level) source file from which the exception was thrown.

**const char * DexelaException::GetFunctionName ( )**

Function for retrieving the name of the (top-level) function from which the error was thrown.

**Note:** This should help to find the function that is causing the exception.

**Returns**
- A string containing the name of the (top-level) function from which the exception was thrown.
int DexelaException::GetLineNumber ( )

Function for retrieving line-number of the source of the (low-level) exception throw.
**Note:** This number (along with transport-error and file-name) can be sent to PerkinElmer support for further information about possible causes of the exception.

**Returns**
An integer detailing the exception's low-level line number.

int DexelaException::GetTransportError ( )

Function for retrieving the exception's low-level transport error code.
**Note:** This code (along with line-number, and file-name) can be sent to PerkinElmer support for further information about possible causes of the exception.

**Returns**
An integer detailing the exception's low-level transport error code.

const char * DexelaException::GetTransportMessage ( )

Function for retrieving the (low-level) message from the transport library.
**Note:** This message (along with transport-error, line number and filename) can be sent to PerkinElmer support for further information about possible causes of the exception.

**Returns**
A string containing the low-level transport layer error message.

const char * DexelaException::what ( ) const
    throw (
Function for retrieving the exception's error message.

**Returns**
A string containing the exception's error message.

The documentation for this class was generated from the following files:

- DexelaException.h
- DexelaException.cpp

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by doxygen 1.8.7
DexelaDetector API

DexImage Class Reference

This class is used to store and handle the images acquired from a detector. More...

#include <DexImage.h>
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DexImage (void)</strong></td>
<td>DexImage constructor. Creates a new (empty) image.</td>
<td>More...</td>
</tr>
<tr>
<td>*<em>DexImage (const char <em>filename)</em></em></td>
<td>DexImage constructor. Creates a new image by reading in from the specified file.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>DexImage (const DexImage &amp;input)</strong></td>
<td>DexImage copy constructor. Creates a new DexImage object by copying the input DexImage object.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>void operator= (const DexImage &amp;input)</strong></td>
<td>DexImage assignment operator. Creates a new DexImage object by copying the input DexImage object.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>~DexImage (void)</strong></td>
<td>DexImage destructor.</td>
<td>More...</td>
</tr>
<tr>
<td>*<em>void ReadImage (const char <em>filename)</em></em></td>
<td>Reads an image in from the specified file</td>
<td>More...</td>
</tr>
</tbody>
</table>
| **void WritImage (const char *filename)** | Writes the image data to the specified file (SMV, HIS or TIF)  
**Note**: This will write the entire image stack out.  
**Note2**: See the function **WritImage** for writing out a single (user specified) plane from the stack. | More... |
<p>| *<em>void WritImage (const char <em>filename, int iZ)</em></em> | Writes out a single image plane (user specified) | |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Build</strong> (int iWidth, int iHeight, int iDepth, <strong>pType IpXType)</strong></td>
<td>Builds an image using the specified dimensions and pixel type</td>
</tr>
<tr>
<td><strong>Build</strong> (int model, <strong>bins binFmt</strong>, int iDepth)</td>
<td>Builds an image using the specified detector model and binning format</td>
</tr>
<tr>
<td><strong>GetDataPointerToPlane</strong> (int iZ=0)</td>
<td>Gets the pointer to the image data for the specified plane.</td>
</tr>
<tr>
<td><strong>GetImageXdim</strong> ()</td>
<td>Gets the image x dimension (width) in pixels.</td>
</tr>
<tr>
<td><strong>GetImageYdim</strong> ()</td>
<td>Gets the image y dimension (height) in pixels.</td>
</tr>
<tr>
<td><strong>GetImageDepth</strong> ()</td>
<td>Gets the image depth.</td>
</tr>
<tr>
<td><strong>GetImagePixelType</strong> ()</td>
<td>Gets the image pixel type.</td>
</tr>
<tr>
<td><strong>PlaneAvg</strong> (int iZ=0)</td>
<td>Calculates the average pixel value of the input image for the specified plane.</td>
</tr>
<tr>
<td><strong>FixFlood</strong> ()</td>
<td>This image fixes the input flood image. This means that the reciprocal of the image is taken and normalized about 1. This is done to speed up</td>
</tr>
</tbody>
</table>
the flood correction procedure (multiplication is faster than division).

**Note:** The input flood image should be a median image (i.e. should have a depth of 1). This can be obtained by using the `FindMedianofPlanes` method.

```plaintext
void FindMedianofPlanes ()
This calculates the median image from the input image stack.
**Note:** This function will replace the input stack of images with a single (median) image
```

```plaintext
void FindAverageofPlanes ()
This calculates the average image from the input image stack.
**Note:** This function will replace the input stack of images with a single (average) image of type float
```

```plaintext
void LinearizeData ()
Linearizes the pixel values of the image. This is done using a piece-wise linear approximation where the sections are defined by an array of integers (linearization starts). This allows the output of the detector to be made linear
**Note:** A default set of linearization starts will be used unless the user specifies their own using the `SetLinearizationStarts` method.
```

```plaintext
void SubtractDark ()
Subtracts a dark image from an input image.
**Note:** The dark images must first be loaded before calling this function (see `LoadDarkImage`).
**Note2:** A dark offset value will be added to the resulting image to prevent any negative numbers. This offset is set to 300 by default but can be changed using the `SetDarkOffset` method.
```
void **FloodCorrection** ()
Performs flood correction on the input image using the passed in fixed-flood image.
**Note:** The flood and dark images must first be loaded before calling this function (see **LoadFloodImage**, **LoadDarkImage**). More...

void **DefectCorrection** (int DefectFlags=31)
Function for performing defect corrections on the image.
**Note:** The flood, dark and defect map images must first be loaded before calling this function (see **LoadFloodImage**, **LoadDarkImage**, **LoadDefectMap**). More...

void **SubImageDefectCorrection** (int startCol, int startRow, int width, int height, int CorrectionsFlag=31)

void **FullCorrection** ()
This function performs the full correction (dark/offset, flood/gain and defect).
**Note:** The flood, dark and defect map images must first be loaded before calling this function (see **LoadFloodImage**, **LoadDarkImage**, **LoadDefectMap**). More...

void **UnscrambleImage** ()
This function unscrambles (sorts) a raw image acquired from a detector.
**Note:** The model number and the binning mode of the detector that the image was captured from must be specified (using **SetImageParameters**) before calling this method. More...

void **AddImage** ()
Adds another image (plane) to the stack. More...
**void** LoadDarkImage (const DexImage &dark)

Loads the dark image from the specified **DexImage** object.

**Note:** This dark image will then automatically be used for the various corrections.

**Note2:** This image should be a single plane (e.g. median) image. If it's not then the median image will be calculated (using **FindMedianofPlanes**) and stored.

**Note3:** This image should be of type **Offset**.

More...

**void** LoadDarkImage (const char *filename)

Loads the dark image from the specified file.

**Note:** This dark image will then automatically be used for the various corrections.

**Note2:** This image should be a single plane (e.g. median) image. If it's not then the median image will be calculated (using **FindMedianofPlanes**) and stored.

**Note3:** This image should be of type **Offset**.

More...

**void** LoadFloodImage (const DexImage &flood)

Loads the flood image from the specified **DexImage** object.

**Note:** This flood image will then automatically be used for offset corrections.

**Note2:** This image should be a single plane, fixed floating point image. If it's not then the median image will be calculated (using **FindMedianofPlanes**), then the image will be fixed (using **FixFlood**) and stored.

**Note3:** This image should be of type **Gain**.

More...

**void** LoadFloodImage (const char *filename)

Loads the flood image from the specified file.

**Note:** This flood image will then automatically be used for gain corrections.
**Note2:** This image should be a single plane, fixed floating point image. If it's not then the median image will be calculated (using `FindMedianofPlanes`), then the image will be fixed (using `FixFlood`) and stored.

**Note3:** This image should be of type `Gain`. More...

<table>
<thead>
<tr>
<th>void LoadDefectMap (const DexImage &amp;defect)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loads the defect-map image from the specified DexImage object.</td>
</tr>
<tr>
<td><strong>Note:</strong> This defect-map image will then automatically be used for defect corrections.</td>
</tr>
<tr>
<td><strong>Note2:</strong> This image should be of type <code>Defect</code>. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void LoadDefectMap (const char *filename)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loads the defect-map image from the specified file.</td>
</tr>
<tr>
<td><strong>Note:</strong> This defect-map image will then automatically be used for defect corrections.</td>
</tr>
<tr>
<td><strong>Note2:</strong> This image should be of type <code>Defect</code>. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DexImage GetDarkImage ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the dark image (which is used in offset/dark correction). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DexImage GetFloodImage ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the flood image (which is used in gain/flood correction). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DexImage GetDefectMap ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the defect-map image (which is used in defect correction). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DexImage GetImagePlane (int iZ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates a new DexImage object from the data at the specified plane. More...</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>DexImageTypes getImageType()</td>
</tr>
<tr>
<td>void SetImageType (DexImageTypes type)</td>
</tr>
<tr>
<td>void SetDarkOffset (int offset)</td>
</tr>
<tr>
<td>int GetDarkOffset ()</td>
</tr>
<tr>
<td>void SetLinearizationStarts (unsigned int *msArray, int msLength)</td>
</tr>
<tr>
<td>unsigned int * GetLinearizationStarts (int &amp;msLength)</td>
</tr>
<tr>
<td>void SetImageParameters (bins binningMode, int modelNumber)</td>
</tr>
<tr>
<td>int GetImageModel ()</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>---------------------------------------------</td>
</tr>
<tr>
<td><code>GetModelNumber()</code></td>
</tr>
<tr>
<td><code>GetImageBinning()</code></td>
</tr>
<tr>
<td><code>IsEmpty()</code></td>
</tr>
<tr>
<td><code>SetScrambledFlag(bool onOff)</code></td>
</tr>
<tr>
<td><code>SetROIParameters(unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight)</code></td>
</tr>
</tbody>
</table>
Detailed Description

This class is used to store and handle the images acquired from a detector.
Constructor & Destructor Documentation

DexImage::DexImage ( void )

DexImage constructor. Creates a new (empty) image.

Exceptions
DexelaException

DexImage::DexImage ( const char * filename )

DexImage constructor. Creates a new image by reading in from the specified file.

Parameters
filename Path to file to read image in from

Exceptions
DexelaException

DexImage::DexImage ( const DexImage & input )

DexImage copy constructor. Creates a new DexImage object by copying the input DexImage object.

Parameters
input DexImage object to copy from.

Exceptions
DexelaException
DexImage::~DexImage ( void )

DexImage destructor.
Member Function Documentation

```cpp
void DexImage::AddImage( )
```

Adds another image (plane) to the stack.

Exceptions

DexelaException

```cpp
void DexImage::Build( int iWidth,
                      int iHeight,
                      int iDepth,
                      pType iPxType )
```

Builds an image using the specified dimensions and pixel type

Parameters

- **iWidth**  Desired width (in pixels) for the image.
- **iHeight** Desired height (in pixels) for the image.
- **iDepth**  Desired depth for the image.
- **iPxType** A member of the `pType` enumeration representing the pixel type for the image.

Exceptions

DexelaException

```cpp
void DexImage::Build( int model,
                      bins binFmt,
                      int iDepth )
```
Builds an image using the specified detector model and binning format

**Parameters**
- **model** Detector model type corresponding to the image.
- **binFmt** A member of the `bins` enumeration representing the binning mode corresponding to the image.
- **iDepth** Desired depth for the image.

**Exceptions**
- `DexelaException`

```c
void DexImage::DefectCorrection ( int DefectFlags = 31 )
```

Function for performing defect corrections on the image. **Note:** The flood, dark and defect map images must first be loaded before calling this function (see `LoadFloodImage`, `LoadDarkImage`, `LoadDefectMap`).

**Exceptions**
- `DexelaException`

```c
void DexImage::FindAverageofPlanes ( )
```

This calculates the average image from the input image stack. **Note:** This function will replace the input stack of images with a single (average) image of type float

**Exceptions**
- `DexelaException`

```c
void DexImage::FindMedianofPlanes ( )
```

This calculates the median image from the input image stack.
**Note:** This function will replace the input stack of images with a single (median) image

**Exceptions**

DexelaException

```cpp
void DexImage::FixFlood() {
    // This image fixes the input flood image. 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).
    // Note: The input flood image should be a median image (i.e. should have a depth of 1). This can be obtained by using the FindMedianofPlanes method.
    // Exceptions
    DexelaException
}
```

```cpp
void DexImage::FloodCorrection() {
    // Performs flood correction on the input image using the passed in fixed-flood image.
    // Note: The flood and dark images must first be loaded before calling this function (see LoadFloodImage, LoadDarkImage).
    // Exceptions
    DexelaException
}
```

```cpp
void DexImage::FullCorrection() {
    // This function performs the full correction (dark/offset, flood/gain and defect).
    // Note: The flood, dark and defect map images must first be loaded before calling this function (see LoadFloodImage, LoadDarkImage,
```
**LoadDefectMap**).

**Exceptions**

*DexelaException*

---

**DexImage** **DexImage::GetDarkImage** ( )

Gets the dark image (which is used in offset/dark correction).

**Returns**

The **DexImage** object that is used for offset/dark corrections

**Exceptions**

*DexelaException*

---

**int DexImage::GetDarkOffset** ( )

Gets the current dark offset value. This value is used as an offset to prevent from the possibility of getting negative pixel values.

**Returns**

The offset value (in ADU).

**Exceptions**

*DexelaException*

---

**void * DexImage::GetDataPointerToPlane** ( int \iZ = e )

Gets the pointer to the image data for the specified plane.

**Parameters**

*iZ* The number of the desired image plane.

**Returns**

Pointer to the image data for the specified plane.
**Exceptions**

*DexelaException*

---

**DexImage** *DexImage::GetDefectMap ( )*

Gets the defect-map image (which is used in defect correction).

**Returns**

The *DexImage* object that is used for defect corrections.

**Exceptions**

*DexelaException*

---

**DexImage** *DexImage::GetFloodImage ( )*

Gets the flood image (which is used in gain/flood correction).

**Returns**

The *DexImage* object that is used for gain/flood corrections.

**Exceptions**

*DexelaException*

---

**bins** *DexImage::GetImageBinning ( )*

Gets the binning mode of the detector that was used to acquire the image.

**Returns**

A member of the *bins* enumeration representing the binning mode of the detector.

**Exceptions**

*DexelaException*
int DexImage::GetImageDepth ( )

Gets the image depth.

Returns
   Image depth.

Exceptions
   DexelaException

int DexImage::GetImageModel ( )

Gets the model number of the detector that was used to acquire the image.

Returns
   The detector model number.

Exceptions
   DexelaException

pType DexImage::GetImagePixelType ( )

Gets the image pixel type.

Returns
   A member of the pType enumeration specifying the image pixel type.

Exceptions
   DexelaException

DexImage DexImage::GetImagePlane ( int iZ )

Creates a new DexImage object from the data at the specified
Parameters
   *iZ* The index of the plane to get the data from.

Returns
   A new *DexImage* object that consists of the data from the desired plane.

Exceptions
   *DexelaException*

**DexImageTypes** *DexImage::GetImageType ( )* 

Gets the image type (e.g. offset, gain, data, defect map).

Returns
   A member of the *DexImageTypes* enumeration specifying the image type.

Exceptions
   *DexelaException*

**int DexImage::GetImageXdim ( )**

Gets the image x dimension (width) in pixels.

Returns
   Image x dimension (width) in pixels.

Exceptions
   *DexelaException*

**int DexImage::GetImageYdim ( )**
Gets the image y dimension (height) in pixels.

**Returns**
Image y dimension (height) in pixels.

**Exceptions**
*DexelaException*

---

`unsigned int *`  
*DexImage::GetLinearizationStarts*  
`( int & msLength )`

Gets the linearization section numbers that are used for the linearization correction(*LinearizeData*).

**Parameters**
*msLength* An integer that will be set to the length of the linearization section numbers array.

**Returns**
An array of unsigned integers representing the section numbers used for linearization correction.

**Exceptions**
*DexelaException*

---

`bool DexImage::IsEmpty ( )`

This method returns whether the image is empty or not.

**Returns**
A boolean indicating whether the image is empty.

**Exceptions**
*DexelaException*
void DexImage::LinearizeData ( )

Linearizes the pixel values of the image. This is done using a piece-wise linear approximation where the sections are defined by an array of integers (linearization starts). This allows the output of the detector to be made linear.

**Note:** A default set of linearization starts will be used unless the user specifies their own using the `SetLinearizationStarts` method.

**Exceptions**
- DexelaException

void DexImage::LoadDarkImage ( const DexImage & dark )

Loads the dark image from the specified DexImage object.

**Note:** This dark image will then automatically be used for the various corrections.

**Note2:** This image should be a single plane (e.g. median) image. If it's not then the median image will be calculated (using `FindMedianofPlanes`) and stored.

**Note3:** This image should be of type Offset.

**Parameters**
- **dark** The DexImage object that should be used for dark corrections.

**Exceptions**
- DexelaException

void DexImage::LoadDarkImage ( const char * filename )

Loads the dark image from the specified file.

**Note:** This dark image will then automatically be used for the various corrections.

**Note2:** This image should be a single plane (e.g. median) image. If it's not then the median image will be calculated (using
FindMedianofPlanes) and stored.

**Note3:** This image should be of type Offset.

**Parameters**

- **filename** The path to the image file that should be read in and used for dark corrections.

**Exceptions**

- DexelaException

---

```cpp
void DexImage::LoadDefectMap ( const DexImage & defect )
```

Loads the defect-map image from the specified DexImage object.

**Note:** This defect-map image will then automatically be used for defect corrections.

**Note2:** This image should be of type Defect.

**Parameters**

- **defect** The DexImage object that should be used for defect corrections.

**Exceptions**

- DexelaException

---

```cpp
void DexImage::LoadDefectMap ( const char * filename )
```

Loads the defect-map image from the specified file.

**Note:** This defect-map image will then automatically be used for defect corrections.

**Note2:** This image should be of type Defect.

**Parameters**

- **filename** The path to the image file that should be read in and used for defect corrections.

**Exceptions**
void DexImage::LoadFloodImage ( const DexImage & flood )

Loads the flood image from the specified DexImage object.
Note: This flood image will then automatically be used for offset corrections.
Note2: This image should be a single plane, fixed floating point image. If it's not then the median image will be calculated (using FindMedianofPlanes), then the image will be fixed (using FixFlood) and stored.
Note3: This image should be of type Gain.

Parameters

flood The DexImage object that should be used for flood corrections.

Exceptions

DexelaException

void DexImage::LoadFloodImage ( const char * filename )

Loads the flood image from the specified file.
Note: This flood image will then automatically be used for gain corrections.
Note2: This image should be a single plane, fixed floating point image. If it's not then the median image will be calculated (using FindMedianofPlanes), then the image will be fixed (using FixFlood) and stored.
Note3: This image should be of type Gain.

Parameters

flood The path to the image file that should be read in and used for flood corrections.

Exceptions

DexelaException
void DexImage::operator= ( const DexImage & input )

DexImage assignment operator. Creates a new DexImage object by copying the input DexImage object.

Parameters
   input DexImage object to copy from.

Exceptions
   DexelaException

float DexImage::PlaneAvg ( int iZ = 0 )

Calculates the average pixel value of the input image for the specified plane.

Parameters
   iZ The plane of the image to work on

Returns
   A floating point number that is the average pixel value of the specified image plane.

Exceptions
   DexelaException

void DexImage::ReadImage ( const char * filename )

Reads an image in from the specified file.

Parameters
   filename The path to the file to read image from. Currently this is limited to SMV, HIS and TIF file types.
Exceptions
   DexelaException

void DexImage::SetDarkOffset ( int offset )

Sets the dark offset value to be used for various corrections (e.g. dark correction). This value is used as an offset to prevent from the possibility of getting negative pixel values.

Parameters
   offset The offset value (in ADU).

Exceptions
   DexelaException

void DexImage::SetImageParameters ( bins binningMode, int modelNumber )

Sets the model number and the binning mode of the detector that was used to acquire the image. This is used for the data-sorting (UnscrambleImage).

Parameters
   binningMode A member of the bins enumeration representing the binning mode of the detector.
   modelNumber The model number of the detector

Exceptions
   DexelaException

void DexImage::SetImageType ( DexImageTypes type )

Sets the image type to the desired type.
Parameters

**type** A member of the `DexImageTypes` enumeration that the image type should be set to.

Exceptions

**DexelaException**

```cpp
void DexImage::SetLinearizationStarts ( unsigned int * msArray, int msLength )
```

Sets the linearization section numbers that are used for the linearization correction (`LinearizeData`).

Parameters

- **msArray** An array of unsigned integers representing the section numbers used for linearization correction.
- **msLength** The length of the array

Exceptions

**DexelaException**

```cpp
void DexImage::SetScrambledFlag ( bool onOff )
```

This method can be used to manually set the scrambled flag of the image. This flag stores whether the data from the detector has already been unscrambled or not. This flag is automatically set when an image is unscrambled and consequently this method should not be required for most use-cases.

Parameters

- **onOff** A boolean representing the desired state of the flag.

Exceptions

**DexelaException**
**void DexImage::SubtractDark ( )**

Subtracts a dark image from an input image.
**Note:** The dark images must first be loaded before calling this function (see **LoadDarkImage**).
**Note2:** A dark offset value will be added to the resulting image to prevent any negative numbers. This offset is set to 300 by default but can be changed using the **SetDarkOffset** method.

**Exceptions**
- **DexelaException**

**void DexImage::UnscrambleImage ( )**

This function unscrambles (sorts) a raw image acquired from a detector.
**Note:** The model number and the binning mode of the detector that the image was captured from must be specified (using **SetImageParameters**) before calling this method.

**Exceptions**
- **DexelaException**

**void DexImage::WriteImage ( const char * filename )**

Writes the image data to the specified file (SMV, HIS or TIF)
**Note:** This will write the entire image stack out.
**Note2:** See the function **WriteImage** for writing out a single (user specified) plane from the stack.

**Parameters**
- **filename** The path to the file to write image to. Currently this is limited to SMV, HIS and TIF file types.
void DexImage::WriteImage ( const char * filename, int iZ )

Writes out a single image plane (user specified) from the stack (SMV, HIS or TIF)

**Parameters**
- **filename** The path to the file to write image to. Currently this is limited to SMV, HIS and TIF file types.
- **iZ** Image plane to write out

**Exceptions**
- **DexelaException**

The documentation for this class was generated from the following files:

- **DexImage.h**
- **DexImage.cpp**

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by [doxygen](http://www.doxygen.nl) 1.8.7
GeometryCorrectionParams Struct Reference

A structure used to specify the new image dimensions for geometry correction More...

#include <DexDefs.h>
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>iRefX</code></td>
<td>The new image width</td>
</tr>
<tr>
<td>int</td>
<td><code>iRefY</code></td>
<td>The new image height</td>
</tr>
</tbody>
</table>
Detailed Description

A structure used to specify the new image dimensions for geometry correction
Member Data Documentation

int GeometryCorrectionParams::iRefX

The new image width

int GeometryCorrectionParams::iRefY

The new image height

The documentation for this struct was generated from the following file:

- DexDefs.h
DexelaDetector API

Class Index

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by doxygen 1.8.7
## Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BusScanner</strong></td>
<td>This class is used to scan the different interfaces and give information about devices found.</td>
</tr>
<tr>
<td><strong>DetStatus</strong></td>
<td>Structure to hold the detector current status.</td>
</tr>
<tr>
<td><strong>DevInfo</strong></td>
<td>A structure to hold device information.</td>
</tr>
<tr>
<td><strong>DexelaDetector</strong></td>
<td>This class is used to control any interface-type Detector and acquire images from it. It will provide all the basic functionality required for all different Dexela detectors. For interface specific functionality please see the interface specific classes (e.g. <a href="#">DexelaDetectorGE</a>, <a href="#">DexelaDetectorCL</a>).</td>
</tr>
</tbody>
</table>
| **DexelaDetectorCL** | This class is used to control CameraLink Type Detectors. It will give access to functions that are not available to other interface-type detectors.  
**Note:** For all standard detector function calls please see the [DexelaDetector](#) class (these... |
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DexelaDetectorGE</strong></td>
<td>This class is used to control GigE Type Detectors. It will give access to functions that are not available to other interface-type detectors. &lt;br&gt;<strong>Note:</strong> For all standard detector function calls please see the <strong>DexelaDetector</strong> class (these functions are also available to <strong>DexelaDetectorGE</strong> objects)</td>
</tr>
<tr>
<td><strong>DexImage</strong></td>
<td>This class is used to store and handle the images acquired from a detector.</td>
</tr>
<tr>
<td><strong>DexelaException</strong></td>
<td>This class contains information about any possible error's in the API. In the event of a problem a <strong>DexelaException</strong> will be thrown. &lt;br&gt;<strong>Note:</strong> It is suggested that you wrap your code in a try-catch block to ensure that if any errors occur you can detect (and properly handle them) in your code.</td>
</tr>
<tr>
<td><strong>GeometryCorrectionParams</strong></td>
<td>A structure used to specify the new image dimensions for geometry correction</td>
</tr>
</tbody>
</table>
DexelaDetector API

Here is a list of all documented class members with links to the class documentation for each member:

- **a** -
  - AddImage() : DexImage

- **b** -
  - binLevel : DetStatus
  - Build() : DexImage
  - BusScanner() : BusScanner

- **c** -
  - CheckForCallbackError() : DexelaDetector
  - CheckForLiveError() : DexelaDetector
  - ClearBuffers() : DexelaDetector
  - ClearCameraBuffer() : DexelaDetector
  - CloseBoard() : DexelaDetector

- **d** -
  - DefectCorrection() : DexImage
  - DexelaDetector() : DexelaDetector
  - DexelaDetectorCL() : DexelaDetectorCL
  - DexelaDetectorGE() : DexelaDetectorGE
  - DexelaException() : DexelaException
  - DexImage() : DexImage
  - DisablePulseGenerator() : DexelaDetector
- e -

- EnableFrameCntWithinImage() : DexelaDetector
- EnablePulseGenerator() : DexelaDetector
- EnableROIMode() : DexelaDetector
- EnumerateCLDevices() : BusScanner
- EnumerateDevices() : BusScanner
- EnumerateGEDevices() : BusScanner
- exposureMode : DetStatus
- exposureTime : DetStatus

- f -

- FindAverageofPlanes() : DexImage
- FindMedianofPlanes() : DexImage
- FixFlood() : DexImage
- FloodCorrection() : DexImage
- FullCorrection() : DexImage
- fullWellMode : DetStatus

- g -

- GetBinningMode() : DexelaDetector
- GetBufferXdim() : DexelaDetector
- GetBufferYdim() : DexelaDetector
- GetCapturedBuffer() : DexelaDetector
- GetCode() : DexelaException
- GetDarkImage() : DexImage
- GetDarkOffset() : DexImage
- GetDataPointerToPlane() : DexImage
- GetDefectMap() : DexImage
- GetDetectorStatus() : DexelaDetector
- GetDevice() : BusScanner
- GetDeviceCL() : BusScanner
- GetDeviceGE() : BusScanner
- GetExposureMode() : DexelaDetector
- GetExposureTime() : DexelaDetector
- GetFieldCount() : DexelaDetector
- GetFileName() : DexelaException
- GetFirmwareBuild() : DexelaDetector
- i -

- iface : DexelaDetector
- iRefX : GeometryCorrectionParams
- iRefY : GeometryCorrectionParams
- IsCallbackActive() : DexelaDetector
- isConnected() : DexelaDetector

- GetFirmwareVersion() : DexelaDetector
- GetFloodImage() : DexImage
- GetFullWellMode() : DexelaDetector
- GetFunctionName() : DexelaException
- GetGapTime() : DexelaDetector
- GetImageBinning() : DexImage
- GetImageDepth() : DexImage
- GetImageModel() : DexImage
- GetImagePixelType() : DexImage
- GetImagePlane() : DexImage
- GetImageType() : DexImage
- GetImageXdim() : DexImage
- GetImageYdim() : DexImage
- GetLinearizationStarts() : DexImage
- GetLineNumber() : DexelaException
- GetModelNumber() : DexelaDetector
- GetNumBuffers() : DexelaDetector
- GetNumOfExposures() : DexelaDetector
- GetReadoutMode() : DexelaDetector
- GetReadOutTime() : DexelaDetector
- GetROICoordinates() : DexelaDetector
- GetROIState() : DexelaDetector
- GetSensorHeight() : DexelaDetector
- GetSensorWidth() : DexelaDetector
- GetSerialNumber() : DexelaDetector
- GetTestMode() : DexelaDetector
- GetTransportError() : DexelaException
- GetTransportMessage() : DexelaException
- GetTransportMethod() : DexelaDetector
- GetTriggerSource() : DexelaDetector
- GoLiveSeq() : DexelaDetector
- GoUnLive() : DexelaDetector
IsEmpty() : DexImage
IsFrameCntWithinImage() : DexelaDetector
IsLive() : DexelaDetector

- l -

LinearizeData() : DexImage
LoadDarkImage() : DexImage
LoadDefectMap() : DexImage
LoadFloodImage() : DexImage
LoadSensorConfigFile() : DexelaDetector

- m -

model : DevInfo

- o -

OpenBoard() : DexelaDetector, DexelaDetectorCL, DexelaDetectorGE
operator=() : DexImage

- p -

param : DevInfo
PlaneAvg() : DexImage
PowerCLInterface() : DexelaDetectorCL

- q -

QueryBinningMode() : DexelaDetector
QueryExposureMode() : DexelaDetector
QueryFullWellMode() : DexelaDetector
QueryReadoutMode() : DexelaDetector
QueryTriggerSource() : DexelaDetector

- r -

ReadBuffer() : DexelaDetector
- s -

- serialNum : DevInfo
- SetBinningMode() : DexelaDetector
- SetCallback() : DexelaDetector
- SetDarkOffset() : DexImage
- SetExposureMode() : DexelaDetector
- SetExposureTime() : DexelaDetector
- SetFullWellMode() : DexelaDetector
- SetGapTime() : DexelaDetector
- SetImageParameters() : DexImage
- SetImageType() : DexImage
- SetLinearizationStarts() : DexImage
- SetNumOfExposures() : DexelaDetector
- SetPersistentIPAddress() : DexelaDetectorGE
- SetPreProgrammedExposureTimes() : DexelaDetector
- SetReadoutMode() : DexelaDetector
- SetROICoordinates() : DexelaDetector
- SetScrambledFlag() : DexImage
- SetSlowed() : DexelaDetector
- SetTestMode() : DexelaDetector
- SetTriggerSource() : DexelaDetector
- Snap() : DexelaDetector
- SoftReset() : DexelaDetector
- SoftwareTrigger() : DexelaDetector
- StopCallback() : DexelaDetector
- SubtractDark() : DexImage

- t -

- testMode : DetStatus
- ToggleGenerator() : DexelaDetector
- transport : DevInfo
- triggerSource : DetStatus

- u -
• unit: **DevInfo**
  • UnscrambleImage(): **DexImage**

- **W** -

  • WaitImage(): **DexelaDetector**
  • what(): **DexelaException**
  • WriteBuffer(): **DexelaDetector**
  • WriteImage(): **DexImage**
  • WriteRegister(): **DexelaDetector**

- ~ ~ -

  • ~BusScanner(): **BusScanner**
  • ~DexelaDetector(): **DexelaDetector**
  • ~DexelaDetectorCL(): **DexelaDetectorCL**
  • ~DexelaDetectorGE(): **DexelaDetectorGE**
  • ~DexelaException(): **DexelaException**
  • ~DexImage(): **DexImage**
- a -
  - AddImage() : DexImage

- b -
  - Build() : DexImage
  - BusScanner() : BusScanner

- c -
  - CheckForCallbackError() : DexelaDetector
  - CheckForLiveError() : DexelaDetector
  - ClearBuffers() : DexelaDetector
  - ClearCameraBuffer() : DexelaDetector
  - CloseBoard() : DexelaDetector

- d -
  - DefectCorrection() : DexImage
  - DexelaDetector() : DexelaDetector
  - DexelaDetectorCL() : DexelaDetectorCL
  - DexelaDetectorGE() : DexelaDetectorGE
  - DexelaException() : DexelaException
  - DexImage() : DexImage
  - DisablePulseGenerator() : DexelaDetector
- e -

- EnableFrameCntWithinImage() : DexelaDetector
- EnablePulseGenerator() : DexelaDetector
- EnableROIMode() : DexelaDetector
- EnumerateCLDevices() : BusScanner
- EnumerateDevices() : BusScanner
- EnumerateGEDevices() : BusScanner

- f -

- FindAverageofPlanes() : DexImage
- FindMedianofPlanes() : DexImage
- FixFlood() : DexImage
- FloodCorrection() : DexImage
- FullCorrection() : DexImage

- g -

- GetBinningMode() : DexelaDetector
- GetBufferXdim() : DexelaDetector
- GetBufferYdim() : DexelaDetector
- GetCapturedBuffer() : DexelaDetector
- GetCode() : DexelaException
- GetDarkImage() : DexImage
- GetDarkOffset() : DexImage
- GetDataPointerToPlane() : DexImage
- GetDefectMap() : DexImage
- GetDetectorStatus() : DexelaDetector
- GetDevice() : BusScanner
- GetDeviceCL() : BusScanner
- GetDeviceGE() : BusScanner
- GetExposureMode() : DexelaDetector
- GetExposureTime() : DexelaDetector
- GetFieldCount() : DexelaDetector
- GetFileName() : DexelaException
- GetFirmwareBuild() : DexelaDetector
- GetFirmwareVersion() : DexelaDetector
- GetFloodImage() : DexImage
- GetFullWellMode() : DexelaDetector
- i -

- l -

- GetFunctionName() : DexelaException
- GetGapTime() : DexelaDetector
- GetImageBinning() : DexImage
- GetImageDepth() : DexImage
- GetImageModel() : DexImage
- GetImagePixelType() : DexImage
- GetImagePlane() : DexImage
- GetImageType() : DexImage
- GetImageXdim() : DexImage
- GetImageYdim() : DexImage
- GetLinearizationStarts() : DexImage
- GetLineNumber() : DexelaException
- GetModelNumber() : DexelaDetector
- GetNumBuffers() : DexelaDetector
- GetNumOfExposures() : DexelaDetector
- GetReadoutMode() : DexelaDetector
- GetReadOutTime() : DexelaDetector
- GetROICoordinates() : DexelaDetector
- GetROIState() : DexelaDetector
- GetSensorHeight() : DexelaDetector
- GetSensorWidth() : DexelaDetector
- GetSerialNumber() : DexelaDetector
- GetTestMode() : DexelaDetector
- GetTransportError() : DexelaException
- GetTransportMessage() : DexelaException
- GetTransportMethod() : DexelaDetector
- GetTriggerSource() : DexelaDetector
- GoLiveSeq() : DexelaDetector
- GoUnLive() : DexelaDetector

- IsCallbackActive() : DexelaDetector
- IsConnected() : DexelaDetector
- IsEmpty() : DexImage
- IsFrameCntWithinImage() : DexelaDetector
- IsLive() : DexelaDetector
- LinearizeData() : DexImage
  - LoadDarkImage() : DexImage
  - LoadDefectMap() : DexImage
  - LoadFloodImage() : DexImage
  - LoadSensorConfigFile() : DexelaDetector

- o -
  - OpenBoard() : DexelaDetector, DexelaDetectorCL, DexelaDetectorGE
  - operator=() : DexImage

- p -
  - PlaneAvg() : DexImage
  - PowerCLInterface() : DexelaDetectorCL

- q -
  - QueryBinningMode() : DexelaDetector
  - QueryExposureMode() : DexelaDetector
  - QueryFullWellMode() : DexelaDetector
  - QueryReadoutMode() : DexelaDetector
  - QueryTriggerSource() : DexelaDetector

- r -
  - ReadBuffer() : DexelaDetector
  - ReadImage() : DexImage
  - ReadRegister() : DexelaDetector

- s -
  - SetBinningMode() : DexelaDetector
  - SetCallback() : DexelaDetector
  - SetDarkOffset() : DexImage
  - SetExposureMode() : DexelaDetector
  - SetExposureTime() : DexelaDetector
  - SetFullWellMode() : DexelaDetector
- t -

  - ToggleGenerator() : DexelaDetector

- u -

  - UnscrambleImage() : DexImage

- w -

  - WaitImage() : DexelaDetector
  - what() : DexelaException
  - WriteBuffer() : DexelaDetector
  - WriteImage() : DexImage
  - WriteRegister() : DexelaDetector

- ~ -

  - ~BusScanner() : BusScanner
  - ~DexelaDetector() : DexelaDetector
- `~DexelaDetectorCL()`: DexelaDetectorCL
- `~DexelaDetectorGE()`: DexelaDetectorGE
- `~DexelaException()`: DexelaException
- `~DexImage()`: DexImage
• binLevel : DetStatus
• exposureMode : DetStatus
• exposureTime : DetStatus
• fullWellMode : DetStatus
• iface : DevInfo
• iRefX : GeometryCorrectionParams
• iRefY : GeometryCorrectionParams
• model : DevInfo
• param : DevInfo
• serialNum : DevInfo
• testMode : DetStatus
• transport : DevInfo
• triggerSource : DetStatus
• unit : DevInfo
Here is a list of all documented files with brief descriptions:

<table>
<thead>
<tr>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusScanner.h</td>
</tr>
<tr>
<td>DexDefines.h</td>
</tr>
<tr>
<td>DexDefs.h</td>
</tr>
<tr>
<td>DexelaDetector.h</td>
</tr>
<tr>
<td>DexelaDetectorCL.h</td>
</tr>
<tr>
<td>DexelaDetectorGE.h</td>
</tr>
<tr>
<td>DexelaException.h</td>
</tr>
<tr>
<td>DexImage.h</td>
</tr>
<tr>
<td>DexImage/resource.h</td>
</tr>
<tr>
<td>BusScanner/resource.h</td>
</tr>
<tr>
<td>DexelaException/resource.h</td>
</tr>
<tr>
<td>resource1.h</td>
</tr>
</tbody>
</table>
DexelaDetector API

BusScanner.h File Reference

```c
#include "DexDefs.h" #include "DexelaDetector.h"
#include <vector>
#include <boost/shared_ptr.hpp>
```

Go to the source code of this file.
Classes

class **BusScanner**
This class is used to scan the different interfaces and give information about devices found. *More...*
DexelaDetector API

DexDefines.h File
Reference

Go to the source code of this file.
### Macros

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TransMsgSize</code></td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td><code>DllExport</code></td>
<td></td>
<td><code>__declspec(dllimport)</code></td>
</tr>
<tr>
<td><code>DllExportC</code></td>
<td></td>
<td><code>__declspec(dllexport)</code></td>
</tr>
<tr>
<td><code>MAX_PIXEL_VAL</code></td>
<td>16383</td>
<td>Maximum value for any pixel in detector output (14 bit)</td>
</tr>
<tr>
<td><code>MIN_PIXEL_VAL</code></td>
<td>0</td>
<td>Minimum allowable pixel value</td>
</tr>
<tr>
<td><code>minTimeIncrement</code></td>
<td>0.01F</td>
<td>The minimum time increment in ms for expose and read mode</td>
</tr>
<tr>
<td><code>minTimeIncrement2</code></td>
<td>195.2F</td>
<td>The minimum time increment in ns for line delay mode</td>
</tr>
<tr>
<td><code>ExposureSleepTimems</code></td>
<td>10</td>
<td>Variable to hold a time in ms used for sleeping threads in streaming mode.</td>
</tr>
<tr>
<td><code>TimingResolution</code></td>
<td>100</td>
<td>The resolution of the exposure time settings 1 for ms 100 for 0.01 ms</td>
</tr>
<tr>
<td><code>RETURN_CHAR_LENGTH_CONST</code></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td><code>DarkPixelXOffset</code></td>
<td>2</td>
<td>The offset in pixels in x when reading dark pixel data</td>
</tr>
<tr>
<td>#define</td>
<td>Define Name</td>
<td>Value</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>DarkPixelYOffset</td>
<td>4</td>
<td>The offset in pixels in y when reading dark pixel data</td>
</tr>
<tr>
<td>AddrFPGANumber</td>
<td>126</td>
<td>Register address for the FPGA version number of the detector</td>
</tr>
<tr>
<td>AddrSerialNumber</td>
<td>125</td>
<td>Register address for the serial number of the detector</td>
</tr>
<tr>
<td>AddrModelNumber</td>
<td>124</td>
<td>Register address for the model number of the detector</td>
</tr>
<tr>
<td>AddrGapTime</td>
<td>18</td>
<td>Register address for the gap time used in Frame Rate mode</td>
</tr>
<tr>
<td>AddrNumberOfFrames</td>
<td>17</td>
<td>Register address of number of frames register for use with sequence modes</td>
</tr>
<tr>
<td>AddrFirmwareVersion</td>
<td>127</td>
<td>Register address of firmware version information</td>
</tr>
<tr>
<td>AddrTriggerSource</td>
<td>0</td>
<td>Register address of the Trigger Source bits</td>
</tr>
<tr>
<td>AddrExposureTimeLow</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>AddrExposureTimeHigh</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>AddrExposureTime</td>
<td>12</td>
<td>Register address of exposure time information in low res system</td>
</tr>
<tr>
<td>Define</td>
<td>Address</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>AddrExposureTime2</code></td>
<td>13</td>
<td>Register address of exposure time information in low res system for Line_Delay mode</td>
</tr>
<tr>
<td><code>AddrExposureTime2Low</code></td>
<td>13</td>
<td>Register address of low bytes of exposure time information for line delay mode</td>
</tr>
<tr>
<td><code>AddrExposureTime2High</code></td>
<td>14</td>
<td>Register address of high bytes of exposure time information for line delay mode</td>
</tr>
<tr>
<td><code>AddrHorizontalBinReg</code></td>
<td>10</td>
<td>Address of horizontal binning register</td>
</tr>
<tr>
<td><code>AddrVerticalBinReg</code></td>
<td>9</td>
<td>Address of vertical binning register</td>
</tr>
<tr>
<td><code>AddrControlReg</code></td>
<td>0</td>
<td>Address of control register</td>
</tr>
<tr>
<td><code>AddrPPExposreTime1Low</code></td>
<td>27</td>
<td>Register address of low bytes of pre-programmed exposure time 1 information</td>
</tr>
<tr>
<td><code>AddrPPExposreTime1High</code></td>
<td>28</td>
<td>Register address of high bytes of pre-programmed exposure time 1 information</td>
</tr>
<tr>
<td><code>AddrPPExposreTime2Low</code></td>
<td>29</td>
<td>Register address of low bytes of pre-programmed exposure time 2 information</td>
</tr>
<tr>
<td><code>AddrPPExposreTime2High</code></td>
<td>30</td>
<td>Register address of high bytes of pre-programmed exposure time 2 information</td>
</tr>
<tr>
<td>Define</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><code>AddrPPExposreTime3Low</code></td>
<td>Register address of low bytes of pre-programmed exposure time 3 information</td>
<td>More...</td>
</tr>
<tr>
<td><code>AddrPPExposreTime3High</code></td>
<td>Register address of high bytes of pre-programmed exposure time 3 information</td>
<td>More...</td>
</tr>
<tr>
<td><code>AddrPPExposreTime4Low</code></td>
<td>Register address of low bytes of pre-programmed exposure time 4 information</td>
<td>More...</td>
</tr>
<tr>
<td><code>AddrPPExposreTime4High</code></td>
<td>Register address of high bytes of pre-programmed exposure time 4 information</td>
<td>More...</td>
</tr>
<tr>
<td><code>SerialNumberReg1</code></td>
<td>Address of serial number register 1</td>
<td>More...</td>
</tr>
<tr>
<td><code>SerialNumberReg2</code></td>
<td>Address of serial number register 2</td>
<td>More...</td>
</tr>
<tr>
<td><code>SerialNumberReg3</code></td>
<td>Address of serial number register 3</td>
<td>More...</td>
</tr>
<tr>
<td><code>TemperatureReg</code></td>
<td>Address of temperature register</td>
<td>More...</td>
</tr>
<tr>
<td><code>AddrWellReg</code></td>
<td>Address of Well register</td>
<td>More...</td>
</tr>
<tr>
<td><code>AddrWellHigh</code></td>
<td>Address of High Fullwell register</td>
<td>More...</td>
</tr>
<tr>
<td><code>AddrWellLow</code></td>
<td>Address of Low Fullwell register</td>
<td>More...</td>
</tr>
<tr>
<td>#define</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>AddrSensorBinReg</strong></td>
<td>3</td>
<td>Address of sensor bin register More...</td>
</tr>
<tr>
<td><strong>AddrSensorBinReg2</strong></td>
<td>5</td>
<td>Address of sensor bin register 2 More...</td>
</tr>
<tr>
<td><strong>AddrNumLines</strong></td>
<td>7</td>
<td>Address of number of lines per sensor register More...</td>
</tr>
<tr>
<td><strong>AddrNumPixels</strong></td>
<td>8</td>
<td>Address of number of pixels (per line) per sensor register More...</td>
</tr>
<tr>
<td><strong>SensorBinClear</strong></td>
<td>65087</td>
<td>Constant for clearing binning command More...</td>
</tr>
<tr>
<td><strong>DigitalBinBit</strong></td>
<td>65533</td>
<td>Constant for digital binning bit More...</td>
</tr>
<tr>
<td><strong>Sensor1x1</strong></td>
<td>0</td>
<td>Constant for 1x1 binning command More...</td>
</tr>
<tr>
<td><strong>Sensor1x2</strong></td>
<td>0</td>
<td>Constant for 1x2 binning command More...</td>
</tr>
<tr>
<td><strong>Sensor1x4</strong></td>
<td>64</td>
<td>Constant for 1x4 binning command More...</td>
</tr>
<tr>
<td><strong>Sensor2x1</strong></td>
<td>128</td>
<td>Constant for 2x1 binning command More...</td>
</tr>
<tr>
<td><strong>Sensor2x2</strong></td>
<td>128</td>
<td>Constant for 2x2 binning command More...</td>
</tr>
<tr>
<td><strong>Sensor2x4</strong></td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>Sensor4x1</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Sensor4x2</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Sensor4x4</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>BinCommit</td>
<td>514</td>
<td></td>
</tr>
<tr>
<td>AddrReadOutTime</td>
<td>410</td>
<td></td>
</tr>
<tr>
<td>ReadoutTimeFactor1313</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>AddrROIStartColumn</td>
<td>404</td>
<td></td>
</tr>
<tr>
<td>AddrROIwidth</td>
<td>405</td>
<td></td>
</tr>
<tr>
<td>AddrROIStartRow</td>
<td>402</td>
<td></td>
</tr>
<tr>
<td>AddrROIheight</td>
<td>403</td>
<td></td>
</tr>
<tr>
<td>AddrFrameCounter</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>#define</td>
<td>Address Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td><code>AddrFramePackingMode_ImageCountPerBlock</code> 64</td>
<td>Register address for frame packing mode: number of images per block</td>
<td></td>
</tr>
<tr>
<td><code>AddrFramePackingMode_BlockHeightInRows</code> 65</td>
<td>Register address for frame packing mode: number of images per block</td>
<td></td>
</tr>
<tr>
<td><code>AddrBuildDayAndMonth</code> 38</td>
<td>Register address storing day and month of the current firmware build</td>
<td></td>
</tr>
<tr>
<td><code>AddrBuildYear</code> 39</td>
<td>Register address storing the year of the current firmware built</td>
<td></td>
</tr>
<tr>
<td><code>AddrBuildTime</code> 40</td>
<td>Register address storing the time of the current firmware built</td>
<td></td>
</tr>
<tr>
<td><code>AddrReadOutTimeLow</code> 55</td>
<td>Register address storing the first 16bit of detector read-out time will be retrieved in ticks (1 tick = ReadOutTimeBase1313 ns)</td>
<td></td>
</tr>
<tr>
<td><code>AddrReadOutTimeHigh</code> 56</td>
<td>Register address storing the second 16bit of detector read-out time will be retrieved in ticks (1 tick = ReadOutTimeBase1313 ns)</td>
<td></td>
</tr>
<tr>
<td><code>AddrControlReg1</code> 1</td>
<td>Address of control register 1</td>
<td></td>
</tr>
<tr>
<td><code>AddrFeaturesReg0</code> 36</td>
<td>Address of features register 0</td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td><code>AddrFeaturesReg1</code></td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Address of features register 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>AVGERAGED_FLAG</code></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><code>FIXED_FLAG</code></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><code>LINEARIZED_FLAG</code></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td><code>SORTED_FLAG</code></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td><code>CLEAR_SORTED_FLAG</code></td>
<td>0xFFF7</td>
<td></td>
</tr>
<tr>
<td><code>OPERATION_KNOWN_FLAG</code></td>
<td>0x8000</td>
<td></td>
</tr>
<tr>
<td><code>CLEAR_OPERATION_KNOWN_FLAG</code></td>
<td>0x7FFF</td>
<td></td>
</tr>
<tr>
<td><code>NOOP_FLAG</code></td>
<td>0x0</td>
<td></td>
</tr>
<tr>
<td><code>XIS_OFFSET_CORRECTED_FLAG</code></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><code>XIS_GAIN_CORRECTED_FLAG</code></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><code>XIS_DEFECT_CORRECTED_FLAG</code></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td><code>XIS_MULTIGAIN_CORRECTED</code></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>/<em>this is not currently used apart from in XIS</em>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DEX_OFFSET_CORRECTED_FLAG</code></td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>/<em>Dexela versions of the corrections</em>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DEX_GAIN_CORRECTED_FLAG</code></td>
<td>32</td>
<td></td>
</tr>
<tr>
<td><code>DEX_DEFECT_CORRECTED_FLAG</code></td>
<td>64</td>
<td></td>
</tr>
<tr>
<td><code>DEX_EXTRA_PARAMS_FLAG</code></td>
<td>0x4000</td>
<td></td>
</tr>
<tr>
<td>/*this flag will</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
#define indicate the presence of new parameters (e.g. model, binning, operations) in the HIS header

#define CORRECTION_KNOWN_FLAG 0x8000
#define UNCORRECTED_FLAG 0x0

#define TIFFTAG_DEX_CORRECTION_FLAGS 34595 /* New tiff-tag for storing correction flags parameter */

#define TIFFTAG_DEX_OPERATION_FLAGS 34596 /* New tiff-tag for storing operation flags parameter */

#define TIFFTAG_DEX_IMAGE_TYPE 34597 /* New tiff-tag for storing image-type parameter */

#define DEX_DATA_IMAGE 0 /* regular data image */
#define DEX_OFFSET_IMAGE 1 /* offset data image */
#define DEX_GAIN_IMAGE 2 /* gain data image */
#define DEX_DEFECT_MAP 3 /* defect map image */
#define DEX_UNKONWN_TYPE_IMAGE 0xFF /* type of image is unknown */

#define TIFFTAG_DEX_MODEL_NUM 34598 /* New tiff-tag for storing image-type parameter */

#define TIFFTAG_DEX_BIN_FMT 34599
#define TIFFTAG_ROI_START_COL 34600
#define TIFFTAG_ROI_START_ROW 34601
#define TIFFTAG_DEFECT_FLAGS 34602
#define MAX_REG_ADDR 999

#define MAX_REG_VALUE 0xFFFF
## Typedefs

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef unsigned short</td>
<td>ushort</td>
</tr>
<tr>
<td>typedef unsigned long</td>
<td>ulong</td>
</tr>
<tr>
<td>typedef unsigned char</td>
<td>byte</td>
</tr>
</tbody>
</table>
## Macro Definition Documentation

### #define AddrBuildDayAndMonth	38
Register address storing day and month of the current firmware build

### #define AddrBuildTime	40
Register address storing the time of the current firmware built

### #define AddrBuildYear	39
Register address storing the year of the current firmware built

### #define AddrControlReg	0
Address of control register

### #define AddrControlReg1	1
Address of control register 1

### #define AddrExposureTime	12
Register address of exposure time information in low res system

### #define AddrExposureTime2	13
Register address of exposure time information in low res system for Line_Delay mode

```c
#define AddrExposureTime2High   14
```

Register address of high bytes of exposure time information for line delay mode

```c
#define AddrExposureTime2Low    13
```

Register address of low bytes of exposure time information for line delay mode

```c
#define AddrFeaturesReg0        36
```

Address of features register 0

```c
#define AddrFeaturesReg1        37
```

Address of features register 1

```c
#define AddrFirmwareVersion     127
```

Register address of firmware version information

```c
#define AddrFPGANumber          126
```

Register address for the FPGA version number of the detector

REGISTER CONSTANTS/////////////////////////
<table>
<thead>
<tr>
<th>#define</th>
<th>Address/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddrFrameCounter</td>
<td>63 Register address for 16bit framecounter</td>
</tr>
<tr>
<td>AddrFramePackingMode_BlockHeightInRows</td>
<td>65 Register address for frame packing mode: number of images per block</td>
</tr>
<tr>
<td>AddrFramePackingMode_ImageCountPerBlock</td>
<td>64 Register address for frame packing mode: number of images per block</td>
</tr>
<tr>
<td>AddrGapTime</td>
<td>18 Register address for the gap time used in Frame Rate mode</td>
</tr>
<tr>
<td>AddrHorizontalBinReg</td>
<td>10 Address of horizontal binning register</td>
</tr>
<tr>
<td>AddrModelNumber</td>
<td>124 Register address for the model number of the detector</td>
</tr>
<tr>
<td>AddrNumberOfFrames</td>
<td>17 Register address of number of frames register for use with sequence modes</td>
</tr>
</tbody>
</table>
#define AddrNumLines  7
Address of Number of lines per sensor register

#define AddrNumPixels  8
Address of Number of pixels (per line) per sensor register

#define AddrPPExposreTime1High  28
Register address of high bytes of pre-programmed exposure time 1 information

#define AddrPPExposreTime1Low  27
Register address of low bytes of pre-programmed exposure time 1 information

#define AddrPPExposreTime2High  30
Register address of high bytes of pre-programmed exposure time 2 information

#define AddrPPExposreTime2Low  29
Register address of low bytes of pre-programmed exposure time 2 information

#define AddrPPExposreTime3High  32
Register address of high bytes of pre-programmed exposure time 3 information

#define AddrPPExposreTime3Low 31

Register address of low bytes of pre-programmed exposure time 3 information

#define AddrPPExposreTime4High 34

Register address of high bytes of pre-programmed exposure time 4 information

#define AddrPPExposreTime4Low 33

Register address of low bytes of pre-programmed exposure time 4 information

#define AddrReadOutTime 410

Readout time for the sensor, dynamically adapted to the read out mode (binning, ROI). Dependent on the implementation the value has to be multiplied by the factor ReadoutTimeFactor

#define AddrReadOutTimeHigh 56

Register address storing the second 16bit of detector read-out time will be retrieved in ticks (1 tick = ReadOutTimeBase1313 ns)

#define AddrReadOutTimeLow 55
Register address storing the first 16bit of detector read-out time will be retrieved in ticks (1 tick = ReadOutTimeBase1313 ns)

#define AddrROIheight 403

Register address for the height of the ROI stripe

#define AddrROIStartColumn 404

Register address for the ROI OFFSET / first column:

#define AddrROIStartRow 402

Register address for the ROI OFFSET / first row:

#define AddrROIwidth 405

Register address for the width of the ROI stripe

#define AddrSensorBinReg 3

Address of sensor bin register

#define AddrSensorBinReg2 5

Address of sensor bin register 2

#define AddrSerialNumber 125
Register address for the serial number of the detector

```
#define AddrTriggerSource  0
```

Register address of the Trigger Source bits

```
#define AddrVerticalBinReg  9
```

Address of vertical binning register

```
#define AddrWellHigh       4
```

Address of High Fullwell register

```
#define AddrWellLow        65531
```

Address of Low Fullwell register

```
#define AddrWellReg        3
```

Address of Well register

```
#define BinCommit          514
```

Constant for binning commit command

```
#define DarkPixelXOffset   2
```

The offset in pixels in x when reading dark pixel data
#define DarkPixelYOffset 4

The offset in pixels in y when reading dark pixel data

#define DigitalBinBit 65533

Constant for digital binning bit

#define ExposureSleepTimems 10

Variable to hold a time in ms used for sleeping threads in streaming mode.

#define MAX_PIXEL_VAL 16383

Maximum value for any pixel in detector output (14 bit)

#define MIN_PIXEL_VAL 0

Minimum allowable pixel value

#define minTimeIncrement 0.01F

The minimum time increment in ms for expose and read mode

#define minTimeIncrement2 195.2F

The minimum time increment in ns for line delay mode
#define ReadoutTimeFactor1313  2

ReadoutTimeFactor for Dexela 1313

#define Sensor1x1    0

Constant for 1x1 binning command

#define Sensor1x2    0

Constant for 1x2 binning command

#define Sensor1x4    64

Constant for 1x4 binning command

#define Sensor2x1    128

Constant for 2x1 binning command

#define Sensor2x2    128

Constant for 2x2 binning command

#define Sensor2x4    192

Constant for 2x4 binning command
```c
#define Sensor4x1  256
Constant for 4x1 binning command

#define Sensor4x2  256
Constant for 4x2 binning command

#define Sensor4x4  320
Constant for 4x4 binning command

#define SensorBinClear  65087
Constant for clearing binning command

#define SerialNumberReg1  0
Address of serial number register 1

#define SerialNumberReg2  0
Address of serial number register 2

#define SerialNumberReg3  0
Address of serial number register 3

#define TemperatureReg  0
```
Address of temperature register

#define TimingResolution 100

The resolution of the exposure time settings 1 for ms 100 for 0.01 ms
DexDef.s.h File Reference

#include "DexDefs.h" #include "windows.h"

Go to the source code of this file.
## Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>DevInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A structure to hold device information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>GeometryCorrectionParams</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A structure used to specify the new image dimensions for geometry correction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>struct</th>
<th>DetStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure to hold the detector current status.</td>
</tr>
</tbody>
</table>
### Enumerations

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DetectorInterface</code></td>
<td>{ CL, GIGE } An enumeration of detector interface types. More...</td>
</tr>
<tr>
<td><code>TransportLib</code></td>
<td>{ Pleora, Epix } An enumeration of detector interface types. More...</td>
</tr>
<tr>
<td><code>bins</code></td>
<td>{ ( x_{11} = 1, x_{12}, x_{14}, x_{21}, x_{22}, x_{24}, x_{41}, x_{42}, x_{44}, ix_{22}, ) binsError } An enumeration of the different bin levels available More...</td>
</tr>
<tr>
<td><code>FileType</code></td>
<td>{ SMV, TIF, HIS, UNKNOWN } An enumeration of file types More...</td>
</tr>
<tr>
<td><code>Derr</code></td>
<td>{ SUCCESS, NULL_IMAGE, WRONG_TYPE, WRONG_DIMS, BAD_PARAM, BAD_COMMS, BAD_TRIGGER, BAD_COMMS_OPEN, BAD_COMMS_WRITE, BAD_COMMS_READ, BAD_FILE_IO, BAD_BOARD, OUT_OF_MEMORY, EXPOSURE_FAILED, BAD_BIN_LEVEL } Enumeration for error codes returned from the API functions More...</td>
</tr>
<tr>
<td><code>FullWellModes</code></td>
<td>{ Low =0, High, FullWellModesError } An enumeration of the available full well modes More...</td>
</tr>
<tr>
<td><code>pType</code></td>
<td>{ u16 = 2, flt = 4, u32 = 6 } Enumeration of pixel types More...</td>
</tr>
<tr>
<td>Enum Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ExposureModes</strong></td>
<td>{ Expose_and_read, Sequence_Exposure, Frame_Rate_exposure, Preprogrammed_exposure }</td>
</tr>
<tr>
<td></td>
<td>An enumeration of exposure modes.</td>
</tr>
<tr>
<td><strong>ExposureTriggerSource</strong></td>
<td>{ Ext_neg_edge_trig, Internal_Software, Ext_Duration_Trig }</td>
</tr>
<tr>
<td></td>
<td>An enumeration of exposure trigger sources.</td>
</tr>
<tr>
<td><strong>ReadoutModes</strong></td>
<td>{ ContinuousReadout, IdleMode, ReadoutModeError }</td>
</tr>
<tr>
<td></td>
<td>An enumeration of ReadOut modes.</td>
</tr>
<tr>
<td><strong>ResolutionModes</strong></td>
<td>{ pixelsize50micron = 1, pixelsize100micron = 0, ResolutionModesError }</td>
</tr>
<tr>
<td></td>
<td>An enumeration of the available resolution modes.</td>
</tr>
<tr>
<td><strong>DexImageTypes</strong></td>
<td>{ Data = 0, Offset = 1, Gain = 2, Defect = 3, UnknownType = 0xFF }</td>
</tr>
<tr>
<td></td>
<td>An enumeration of the different image types.</td>
</tr>
</tbody>
</table>
**Enumeration Type Documentation**

### enum bins

An enumeration of the different bin levels available

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x11</td>
<td>Unbinned</td>
</tr>
<tr>
<td>x12</td>
<td>Binned vertically by 2</td>
</tr>
<tr>
<td>x14</td>
<td>Binned vertically by 4</td>
</tr>
<tr>
<td>x21</td>
<td>Binned horizontally by 2</td>
</tr>
<tr>
<td>x22</td>
<td>Binned horizontally by 2 and vertically by 2</td>
</tr>
<tr>
<td>x24</td>
<td>Binned horizontally by 2 and vertically by 4</td>
</tr>
<tr>
<td>x41</td>
<td>Binned horizontally by 4</td>
</tr>
<tr>
<td>x42</td>
<td>Binned horizontally by 4 and vertically by 2</td>
</tr>
<tr>
<td>x44</td>
<td>Binned horizontally by 4 and vertically by 4</td>
</tr>
<tr>
<td>ix22</td>
<td>Digital 2x2 binning</td>
</tr>
<tr>
<td>binsError</td>
<td>Indicates an error</td>
</tr>
</tbody>
</table>

### enum Derr

Enumeration for error codes returned from the API functions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>The operation was successful</td>
</tr>
<tr>
<td>NULL_IMAGE</td>
<td>The image pointer was NULL</td>
</tr>
<tr>
<td>WRONG_TYPE</td>
<td>The image pixel type was wrong for the operation requested</td>
</tr>
<tr>
<td>WRONG_DIMS</td>
<td>The image dimensions were wrong for the</td>
</tr>
<tr>
<td>Operation Requested</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>BAD_PARAM One or more parameters were incorrect</td>
<td></td>
</tr>
<tr>
<td>BAD_COMMS The communications channel is not open or could not be opened</td>
<td></td>
</tr>
<tr>
<td>BAD_TRIGGER An invalid trigger source was requested</td>
<td></td>
</tr>
<tr>
<td>BAD_COMMS_OPEN The communications channel failed to open</td>
<td></td>
</tr>
<tr>
<td>BAD_COMMS_WRITE A failure in a detector write command occurred</td>
<td></td>
</tr>
<tr>
<td>BAD_COMMS_READ A failure in a detector read command occurred</td>
<td></td>
</tr>
<tr>
<td>BAD_FILE_IO An error occurred opening or reading from a file</td>
<td></td>
</tr>
<tr>
<td>BAD_BOARD The software failed to open the PC driver or frame grabber</td>
<td></td>
</tr>
<tr>
<td>OUT_OF_MEMORY A function call was not able to reserve the memory it required</td>
<td></td>
</tr>
<tr>
<td>EXPOSURE_FAILED Exposure Acquisition failed</td>
<td></td>
</tr>
<tr>
<td>BAD_BIN_LEVEL Incorrect bin level specified</td>
<td></td>
</tr>
</tbody>
</table>

**enum DetectorInterface**

An enumeration of detector interface types.

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL</td>
</tr>
<tr>
<td>GIGE</td>
</tr>
</tbody>
</table>

**enum DexImageTypes**

An enumeration of the different image types.

<p>| Enumerator |
|------------|------------|</p>
<table>
<thead>
<tr>
<th>Data</th>
<th>A data image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
<td>An offset (dark) image</td>
</tr>
<tr>
<td>Gain</td>
<td>An gain (flood) image</td>
</tr>
<tr>
<td>Defect</td>
<td>A defect-map image</td>
</tr>
<tr>
<td>UnknownType</td>
<td>The type of the image is not known</td>
</tr>
</tbody>
</table>

**enum ExposureModes**

An enumeration of exposure modes.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expose_and_read</td>
<td>The detector should clear the sensor and wait for exposure time to pass before reading the detector image.</td>
</tr>
<tr>
<td>Sequence_Exposure</td>
<td>The detector should take a sequence of images with no gaps.</td>
</tr>
<tr>
<td>Frame_Rate_exposure</td>
<td>The detector should take a sequence of images with a specified gap no less than the minimum exposure time for the bin level.</td>
</tr>
<tr>
<td>Preprogrammed_exposure</td>
<td>The detector should take a number of images with preset exposure times without a gap.</td>
</tr>
</tbody>
</table>

**enum ExposureTriggerSource**

An enumeration of exposure trigger sources.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ext_neg_edge_trig</td>
<td>Trigger on negative edge</td>
</tr>
<tr>
<td>Internal_Software</td>
<td>Trigger using software</td>
</tr>
<tr>
<td>Ext_Duration_Trig</td>
<td>Detector exposure duration and trigger controlled externally</td>
</tr>
</tbody>
</table>
enum **FileType**

An enumeration of file types

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMV</td>
<td>SMV</td>
</tr>
<tr>
<td>TIF</td>
<td>TIFF</td>
</tr>
<tr>
<td>HIS</td>
<td>HIS</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown/Unsupported</td>
</tr>
</tbody>
</table>

enum **FullWellModes**

An enumeration of the available full well modes

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>The low noise reduced dynamic range mode</td>
</tr>
<tr>
<td>High</td>
<td>The normal full well mode</td>
</tr>
<tr>
<td>FullWellModesError</td>
<td>Indicates an error</td>
</tr>
</tbody>
</table>

enum **pType**

Enumeration of pixel types

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16</td>
<td>A pixel type of 16-bit unsigned short</td>
<td></td>
</tr>
<tr>
<td>flt</td>
<td>A pixel type of 32-bit floating point</td>
<td></td>
</tr>
<tr>
<td>u32</td>
<td>A pixel type of 32-bit unsigned int</td>
<td>&lt;summary&gt;</td>
</tr>
</tbody>
</table>

enum **ReadoutModes**
An enumeration of ReadOut modes.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContinuousReadout</td>
<td>The sensor is continuously read-out using the minimum read-out time. On request an image will be transmitted. A frame request can be an external trigger pulse, internal trigger or software trigger</td>
</tr>
<tr>
<td>IdleMode</td>
<td>The sensor is only read out (using the minimum frame time) on request. The read-out will be followed by the transmission of the image. A frame request can be an external trigger pulse, internal trigger or software trigger</td>
</tr>
<tr>
<td>ReadoutModeError</td>
<td>Indicates an error</td>
</tr>
</tbody>
</table>

**enum** ResolutionModes

An enumeration of the available resolution modes

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixelsize50micron</td>
<td>The 100micron mode</td>
</tr>
<tr>
<td>pixelsize100micron</td>
<td>The 50micron mode</td>
</tr>
<tr>
<td>ResolutionModesError</td>
<td>Indicates an error</td>
</tr>
</tbody>
</table>

**enum** TransportLib

An enumeration of detector interface types.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pleora</td>
<td>Pleora type GigE transport library</td>
</tr>
<tr>
<td>Epix</td>
<td>Epix FG CameraLink transport library</td>
</tr>
</tbody>
</table>
DexelaDetector API

DexelaDetector.h File Reference

#include <vector>
#include "DexDefs.h"
#include "DexImage.h"
#include "DexelaException.h"
#include <boost/shared_ptr.hpp>

Go to the source code of this file.
Classes

| class  | DexelaDetector | This class is used to control any interface-type Detector and acquire images from it. It will provide all the basic functionality required for all different Dexela detectors. For interface specific functionality please see the interface specific classes (e.g. DexelaDetectorGE, DexelaDetectorCL). More... |
# Typedefs

<table>
<thead>
<tr>
<th>Type Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef void(* IMAGE_CALLBACK)(int fc, int buf, DexelaDetector *det)</td>
<td>Image callback function signature. This is the signature that any user-passed callback functions must adhere to. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
typedef void(* IMAGE_CALLBACK)(int fc, int buf, DexelaDetector *det)

Image callback function signature. This is the signature that any user-passed callback functions must adhere to.

**Parameters**

- **fc**  The field count associated with the image that just arrived
- **buf** The buffer number where the image was written to (this can be used by DexelaDetector::ReadBuffer() to read-out the image data).
- **det** The DexelaDetector object that sent the image.
DexelaDetector API

DexelaDetectorCL.h
File Reference

#include "dexeladetector.h"

Go to the source code of this file.
**Classes**

<table>
<thead>
<tr>
<th>class</th>
<th>DexelaDetectorCL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This class is used to control CameraLink Type Detectors. It will give access to functions that are not available to other interface-type detectors. <strong>Note:</strong> For all standard detector function calls please see the DexelaDetector class (these functions are also available to DexelaDetectorCL objects) More...</td>
</tr>
</tbody>
</table>

Generated on Tue Nov 25 2014 10:22:44 for DexelaDetector API by doxygen 1.8.7
DexelaDetector API

DexelaDetectorGE.h
File Reference

#include "DexelaDetector.h"

Go to the source code of this file.
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>DexelaDetectorGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This class is used to control GigE Type Detectors. It will give access to functions that are not available to other interface-type detectors. <strong>Note</strong>: For all standard detector function calls please see the DexelaDetector class (these functions are also available to DexelaDetectorGE objects)</td>
</tr>
</tbody>
</table>
DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
</tr>
<tr>
<td>DexelaDetector</td>
<td>DexelaException</td>
<td></td>
</tr>
</tbody>
</table>

DexelaException.h File Reference

```cpp
#include "dexdefs.h" #include <exception>

Go to the source code of this file.
### Classes

<table>
<thead>
<tr>
<th>class</th>
<th>DexelaException</th>
</tr>
</thead>
<tbody>
<tr>
<td>This class contains information about any possible errors in the API. In the event of a problem a DexelaException will be thrown.</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> It is suggested that you wrap your code in a try-catch block to ensure that if any errors occur you can detect (and properly handle them) in your code.</td>
<td></td>
</tr>
</tbody>
</table>
## Macros

```c
#define rethrowEr(EX) throw DexelaException(EX, __FUNCTION__);
#define throwNewEr(MSG, CODE, TRANSER, TRANSMSG) throw DexelaException(MSG, CODE, __LINE__, __FILE__, __FUNCTION__, TRANSER, TRANSMSG);
```
DexelaDetector API

DexImage.h File Reference

#include "DexDefs.h" #include "DexImage.h"
#include <vector>
#include <boost/shared_ptr.hpp>

Go to the source code of this file.
<table>
<thead>
<tr>
<th>class</th>
<th>DexImage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This class is used to store and handle the images acquired from a detector. More...</td>
</tr>
</tbody>
</table>
DexelaDetector API

DexImage/resource.h

```c
1  //{{NO_DEPENDENCIES}}
2  // Microsoft Visual C++ generated include file.
3  // Used by DexImageResources.rc
4  
5  // Next default values for new objects
6  //
7  #ifdef APSTUDIO_INVOKED
8  #ifndef APSTUDIO_READONLY_SYMBOLS
9  #define _APS_NEXT_RESOURCE_VALUE       101
10 #define _APS_NEXT_COMMAND_VALUE       40001
11 #define _APS_NEXT_CONTROL_VALUE       1001
12 #define _APS_NEXT_SYMED_VALUE         101
13 #endif
14 #endif
```
### DexelaDetector API

#### BusScanner/resource.h

```cpp
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.

// Used by BusScannerResources.rc

// Next default values for new objects

#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 101
#define _APS_NEXT_COMMAND_VALUE 40001
#define _APS_NEXT_CONTROL_VALUE 1001
#define _APS_NEXT_SYMED_VALUE 101
#endif
```

---

Generated on Tue Nov 25 2014 10:22:44 for DexelaDetector API by [doxygen](https://www.doxygen.org) 1.8.7
### DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
</tr>
<tr>
<td>DexelaDetector</td>
<td>DexelaException</td>
<td></td>
</tr>
</tbody>
</table>

#### DexelaException/resource.h

```cpp
1 //{{NO_DEPENDENCIES}}
2 // Microsoft Visual C++ generated include file.
3 // Used by DexelaExceptionResources.rc
4 //
5 // Next default values for new objects
6 //
7 #ifdef APSTUDIO_INVOKED
8 #ifndef APSTUDIO_READONLY_SYMBOLS
9 #define _APS_NEXT_RESOURCE_VALUE					101
10 #define _APS_NEXT_COMMAND_VALUE
11 #define _APS_NEXT_CONTROL_VALUE							1001
12 #define _APS_NEXT_SYMED_VALUE							101
13 #endif
14 #endif
```

Generated on Tue Nov 25 2014 10:22:44 for DexelaDetector API by [doxygen](https://doxygen.org) 1.8.7
### resource1.h

```c
1  //{{NO_DEPENDENCIES}}
2  // Microsoft Visual C++ generated include file.
3  // Used by DexelaDetectorResources.rc
4  //
5  // Next default values for new objects
6  //
7  #ifdef APSTUDIO_INVOKED
8  ifndef APSTUDIO_READONLY_SYMBOLS
9  #define _APS_NEXT_RESOURCE_VALUE 101
10 #define _APS_NEXT_COMMAND_VALUE 40001
11 #define _APS_NEXT_CONTROL_VALUE 1001
12 #define _APS_NEXT_SYMED_VALUE 101
13 #endif
14 #endif
```

Generated on Tue Nov 25 2014 10:22:44 for DexelaDetector API by [Doxygen](http://www.stackoverflow.com) 1.8.7
Here is a list of all documented file members with links to the documentation:

- a -

- AddrBuildDayAndMonth : DexDefines.h
- AddrBuildTime : DexDefines.h
- AddrBuildYear : DexDefines.h
- AddrControlReg : DexDefines.h
- AddrControlReg1 : DexDefines.h
- AddrExposureTime : DexDefines.h
- AddrExposureTime2 : DexDefines.h
- AddrExposureTime2High : DexDefines.h
- AddrExposureTime2Low : DexDefines.h
- AddrFeaturesReg0 : DexDefines.h
- AddrFeaturesReg1 : DexDefines.h
- AddrFirmwareVersion : DexDefines.h
- AddrFPGANumber : DexDefines.h
- AddrFrameCounter : DexDefines.h
- AddrFramePackingMode_BlockHeightInRows : DexDefines.h
- AddrFramePackingMode_ImageCountPerBlock : DexDefines.h
- AddrGapTime : DexDefines.h
- AddrHorizontalBinReg : DexDefines.h
- AddrModelNumber : DexDefines.h
- AddrNumberOfFrames : DexDefines.h
- AddrNumLines : DexDefines.h
- AddrNumPixels : DexDefines.h
- AddrPPExposureTime1High : DexDefines.h
- AddrPPExposureTime1Low : DexDefines.h
- AddrPPExposureTime2High : DexDefines.h
- b -

- BAD_BIN_LEVEL : DexDefs.h
  - BAD_BOARD : DexDefs.h
  - BAD_COMMS : DexDefs.h
  - BAD_COMMS_OPEN : DexDefs.h
  - BAD_COMMS_READ : DexDefs.h
  - BAD_COMMS_WRITE : DexDefs.h
  - BAD_FILE_IO : DexDefs.h
  - BAD_PARAM : DexDefs.h
  - BAD_TRIGGER : DexDefs.h
  - BinCommit : DexDefines.h
  - bins : DexDefs.h
  - binsError : DexDefs.h

- c -

- CL : DexDefs.h
  - ContinuousReadout : DexDefs.h
- d -

- DarkPixelXOffset : DexDefines.h
- DarkPixelYOffset : DexDefines.h
- Data : DexDefs.h
- Defect : DexDefs.h
- Derr : DexDefs.h
- DetectorInterface : DexDefs.h
- DexImageTypes : DexDefs.h
- DigitalBinBit : DexDefines.h

- e -

- Epix : DexDefs.h
- Expose_and_read : DexDefs.h
- EXPOSURE_FAILED : DexDefs.h
- ExposureModes : DexDefs.h
- ExposureSleepTimems : DexDefines.h
- ExposureTriggerSource : DexDefs.h
- Ext_Duration_Trig : DexDefs.h
- Ext_neg_edge_trig : DexDefs.h

- f -

- FileType : DexDefs.h
- flt : DexDefs.h
- Frame_Rate_exposure : DexDefs.h
- FullWellModes : DexDefs.h
- FullWellModesError : DexDefs.h

- g -

- Gain : DexDefs.h
- GIGE : DexDefs.h

- h -

- High : DexDefs.h
- HIS : DexDefs.h
- i -

- IdleMode : DexDefs.h
- IMAGE_CALLBACK : DexelaDetector.h
- Internal_Software : DexDefs.h
- ix22 : DexDefs.h

- l -

- Low : DexDefs.h

- m -

- MAX_PIXEL_VAL : DexDefines.h
- MIN_PIXEL_VAL : DexDefines.h
- minTimeIncrement : DexDefines.h
- minTimeIncrement2 : DexDefines.h

- n -

- NULL_IMAGE : DexDefs.h

- o -

- Offset : DexDefs.h
- OUT_OF_MEMORY : DexDefs.h

- p -

- pixelsize100micron : DexDefs.h
- pixelsize50micron : DexDefs.h
- Pleora : DexDefs.h
- Preprogrammed_exposure : DexDefs.h
- pType : DexDefs.h

- r -

- ReadoutModeError : DexDefs.h
- ReadoutModes : DexDefs.h
- R -

- S -

- T -

- U -

- W -
• WRONG_DIMS : DexDefs.h
• WRONG_TYPE : DexDefs.h

- X -

• x11 : DexDefs.h
• x12 : DexDefs.h
• x14 : DexDefs.h
• x21 : DexDefs.h
• x22 : DexDefs.h
• x24 : DexDefs.h
• x41 : DexDefs.h
• x42 : DexDefs.h
• x44 : DexDefs.h
DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>Typedefs</td>
<td>Enumerations</td>
</tr>
</tbody>
</table>

- IMAGE_CALLBACK: DexelaDetector.h

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by doxygen 1.8.7
DexelaDetector API

- bins: DexDefs.h
- Derr: DexDefs.h
- DetectorInterface: DexDefs.h
- DexImageTypes: DexDefs.h
- ExposureModes: DexDefs.h
- ExposureTriggerSource: DexDefs.h
- FileType: DexDefs.h
- FullWellModes: DexDefs.h
- pType: DexDefs.h
- ReadoutModes: DexDefs.h
- ResolutionModes: DexDefs.h
- TransportLib: DexDefs.h

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by doxygen 1.8.7
DexelaDetector API

- b -

- BAD_BIN_LEVEL : DexDefs.h
- BAD_BOARD : DexDefs.h
- BAD_COMMS : DexDefs.h
- BAD_COMMS_OPEN : DexDefs.h
- BAD_COMMS_READ : DexDefs.h
- BAD_COMMS_WRITE : DexDefs.h
- BAD_FILE_IO : DexDefs.h
- BAD_PARAM : DexDefs.h
- BAD_TRIGGER : DexDefs.h
- binsError : DexDefs.h

- c -

- CL : DexDefs.h
- ContinuousReadout : DexDefs.h

- d -

- Data : DexDefs.h
- Defect : DexDefs.h

- e -

- Epix : DexDefs.h
- Expose_and_read : DexDefs.h
- EXPOSURE_FAILED : DexDefs.h
- f -
  - flt : DexDefs.h
  - Frame_Rate_exposure : DexDefs.h
  - FullWellModesError : DexDefs.h

- g -
  - Gain : DexDefs.h
  - GIGE : DexDefs.h

- h -
  - High : DexDefs.h
  - HIS : DexDefs.h

- i -
  - IdleMode : DexDefs.h
  - Internal_Software : DexDefs.h
  - ix22 : DexDefs.h

- l -
  - Low : DexDefs.h

- n -
  - NULL_IMAGE : DexDefs.h

- o -
  - Offset : DexDefs.h
  - OUT_OF_MEMORY : DexDefs.h
- p -
  - pixelsize100micron: DexDefs.h
  - pixelsize50micron: DexDefs.h
  - Pleora: DexDefs.h
  - Preprogrammed_exposure: DexDefs.h

- r -
  - ReadoutModeError: DexDefs.h
  - ResolutionModesError: DexDefs.h

- s -
  - Sequence_Exposure: DexDefs.h
  - SMV: DexDefs.h
  - SUCCESS: DexDefs.h

- t -
  - TIF: DexDefs.h

- u -
  - u16: DexDefs.h
  - u32: DexDefs.h
  - UNKNOWN: DexDefs.h
  - UnknownType: DexDefs.h

- w -
  - WRONG_DIMS: DexDefs.h
  - WRONG_TYPE: DexDefs.h

- x -
  - x11: DexDefs.h
  - x12: DexDefs.h
  - x14: DexDefs.h
• x21 : DexDefs.h
• x22 : DexDefs.h
• x24 : DexDefs.h
• x41 : DexDefs.h
• x42 : DexDefs.h
• x44 : DexDefs.h
## DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>All</th>
<th>Typedefs</th>
<th>Enumerations</th>
<th>Enumerator</th>
<th>Macros</th>
</tr>
</thead>
</table>

### - a -

- `AddrBuildDayAndMonth` : `DexDefines.h`
- `AddrBuildTime` : `DexDefines.h`
- `AddrBuildYear` : `DexDefines.h`
- `AddrControlReg` : `DexDefines.h`
- `AddrControlReg1` : `DexDefines.h`
- `AddrExposureTime` : `DexDefines.h`
- `AddrExposureTime2` : `DexDefines.h`
- `AddrExposureTime2High` : `DexDefines.h`
- `AddrExposureTime2Low` : `DexDefines.h`
- `AddrFeaturesReg0` : `DexDefines.h`
- `AddrFeaturesReg1` : `DexDefines.h`
- `AddrFirmwareVersion` : `DexDefines.h`
- `AddrFPGANumber` : `DexDefines.h`
- `AddrFrameCounter` : `DexDefines.h`
- `AddrFramePackingMode_BlockHeightInRows` : `DexDefines.h`
- `AddrFramePackingMode_ImageCountPerBlock` : `DexDefines.h`
- `AddrGapTime` : `DexDefines.h`
- `AddrHorizontalBinReg` : `DexDefines.h`
- `AddrModelNumber` : `DexDefines.h`
- `AddrNumberOfFrames` : `DexDefines.h`
- `AddrNumLines` : `DexDefines.h`
- `AddrNumPixels` : `DexDefines.h`
- `AddrPPExposreTime1High` : `DexDefines.h`
- `AddrPPExposreTime1Low` : `DexDefines.h`
- `AddrPPExposreTime2High` : `DexDefines.h`
- `AddrPPExposreTime2Low` : `DexDefines.h`
- **b** -

  - BinCommit : **DexDefines.h**

- **d** -

  - DarkPixelXOffset : **DexDefines.h**
  - DarkPixelYOffset : **DexDefines.h**
  - DigitalBinBit : **DexDefines.h**

- **e** -

  - ExposureSleepTimems : **DexDefines.h**

- **m** -

  - MAX_PIXEL_VAL : **DexDefines.h**
  - MIN_PIXEL_VAL : **DexDefines.h**
  - minTimeIncrement : **DexDefines.h**
- r -

- s -

- t -

- Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by doxygen 1.8.7
This is the complete list of members for **BusScanner**, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusScanner(void)</td>
<td></td>
<td>BusScanner</td>
</tr>
<tr>
<td>EnumerateCLDevices()</td>
<td></td>
<td>BusScanner</td>
</tr>
<tr>
<td>EnumerateDevices()</td>
<td></td>
<td>BusScanner</td>
</tr>
<tr>
<td>EnumerateGEDevices()</td>
<td></td>
<td>BusScanner</td>
</tr>
<tr>
<td>GetDevice(int index)</td>
<td></td>
<td>BusScanner</td>
</tr>
<tr>
<td>GetDeviceCL(int index)</td>
<td></td>
<td>BusScanner</td>
</tr>
<tr>
<td>GetDeviceGE(int index)</td>
<td></td>
<td>BusScanner</td>
</tr>
<tr>
<td>ScanMockSetter (defined in BusScanner)</td>
<td></td>
<td>friend</td>
</tr>
<tr>
<td>~BusScanner(void)</td>
<td></td>
<td>BusScanner</td>
</tr>
</tbody>
</table>
```cpp
#pragma once

#ifndef DEX_BUILD
#if defined _DEBUG
#pragma comment(lib,"BusScanner-d.lib")
#else
#pragma comment(lib,"BusScanner.lib")
#endif
#endif

#include "DexDefs.h"
#include "DexelaDetector.h"
#include <vector>
#include <boost/shared_ptr.hpp>

using namespace std;

class DllExport BusScanner
{
    public:
        BusScanner(void);
        ~BusScanner(void);

        int EnumerateDevices();
        int EnumerateGEDevices();
};
```
int EnumerateCLDevices();

DevInfo GetDevice(int index);
DevInfo GetDeviceGE(int index);
DevInfo GetDeviceCL(int index);

friend class ScanMockSetter;

#ifndef MOCK_TEST
private:
#endif
boost::shared_ptr<baseBusScanner> baseScanner;

};
# DetStatus Member List

This is the complete list of members for `DetStatus`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>binLevel</td>
<td>DetStatus</td>
</tr>
<tr>
<td>exposureMode</td>
<td>DetStatus</td>
</tr>
<tr>
<td>exposureTime</td>
<td>DetStatus</td>
</tr>
<tr>
<td>fullWellMode</td>
<td>DetStatus</td>
</tr>
<tr>
<td>testMode</td>
<td>DetStatus</td>
</tr>
<tr>
<td>triggerSource</td>
<td>DetStatus</td>
</tr>
</tbody>
</table>

Generated on Tue Nov 25 2014 10:22:44 for DexelaDetector API by [doxygen](https://www.doxygen.nl/) 1.8.7
DexelaDetector API

DexDefs.h

Go to the documentation of this file.

```c
#pragma once

#include "DexDefines.h"
#include "windows.h"

typedef enum //DetectorInterface
{
    CL,
    GIGE
} DetectorInterface;

typedef enum //TransportLib
{
    Pleora,
    Epix
} TransportLib;

typedef struct //DevInfo
{
    int model;
    int serialNum;
    DetectorInterface iface;
    char param[50];
    int unit;
    TransportLib transport;
} DevInfo;
```
typedef enum //bins
{
    x11 = 1,
    x12,
    x14,
    x21,
    x22,
    x24,
    x41,
    x42,
    x44,
    ix22,
    binsError
} bins;

typedef enum //FileType
{
    SMV,
    TIF,
    HIS,
    UNKNOWN
} FileType;

typedef enum //Derr
{
    SUCCESS,
    NULL_IMAGE,
    WRONG_TYPE,
    WRONG_DIMS,
    BAD_PARAM,
    BAD_COMMS,
    BAD_TRIGGER,
    BAD_COMMS_OPEN,
BAD_COMMS_WRITE,
BAD_COMMS_READ,
BAD_FILE_IO,
BAD_BOARD,
OUT_OF_MEMORY,
EXPOSURE_FAILED,
BAD_BIN_LEVEL
}Derr;

typedef enum //FullWellModes
{
    Low=0,
    High,
    FullWellModesError
}FullWellModes;

typedef enum //pType
{
    u16 = 2,
    flt = 4,
    u32 = 6
}pType;

typedef enum //ExposureModes
{
    Expose_and_read,
    Sequence_Exposure,
    Frame_Rate_exposure,
    Preprogrammed_exposure
}Exposure Modes;

typedef enum //ExposureTriggerSource
{
    Ext_neg_edge_trig,
typedef struct // GeometryCorrectionParams
{
  int iRefX; // 1536
  int iRefY; // 1944
} GeometryCorrectionParams;

typedef struct // DetStatus
{
  ExposureModes exposureMode;
  FullWellModes fullWellMode;
  float exposureTime;
  bins binLevel;
  ExposureTriggerSource triggerSource;
  BOOL testMode;
} DetStatus;

typedef enum // ReadoutModes
{
  ContinuousReadout,
  IdleMode,
  ReadoutModeError
} ReadoutModes;

typedef enum // ResolutionModes
{
  pixelsize50micron = 1,
  pixelsize100micron = 0,
typedef enum //DexImageTypes {
    Data = 0,
    Offset = 1,
    Gain = 2,
    Defect = 3,
    UnknownType = 0xFF
} DexImageTypes;
This is the complete list of members for DevInfo, including all inherited members.

<table>
<thead>
<tr>
<th>iface</th>
<th>DevInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>DevInfo</td>
</tr>
<tr>
<td>param</td>
<td>DevInfo</td>
</tr>
<tr>
<td>serialNum</td>
<td>DevInfo</td>
</tr>
<tr>
<td>transport</td>
<td>DevInfo</td>
</tr>
<tr>
<td>unit</td>
<td>DevInfo</td>
</tr>
</tbody>
</table>
DexelaDetector API

DexelaDetector Member List

This is the complete list of members for DexelaDetector, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>(defined in DexelaDetector)</td>
</tr>
<tr>
<td>baseBusScanner</td>
<td>(defined in DexelaDetector)</td>
</tr>
<tr>
<td>CheckForCallbackError()</td>
<td></td>
</tr>
<tr>
<td>CheckForLiveError()</td>
<td></td>
</tr>
<tr>
<td>clDet</td>
<td>(defined in DexelaDetector)</td>
</tr>
<tr>
<td>ClearBuffers()</td>
<td></td>
</tr>
<tr>
<td>ClearCameraBuffer(int i)</td>
<td></td>
</tr>
<tr>
<td>CloseBoard()</td>
<td></td>
</tr>
<tr>
<td>Dex_CL (defined in DexelaDetector)</td>
<td></td>
</tr>
<tr>
<td>DexelaDetector(DevInfo &amp;devInfo)</td>
<td></td>
</tr>
<tr>
<td>DexelaDetector(DetectorInterface transport, int unit, const char *params)</td>
<td></td>
</tr>
<tr>
<td>DexelaDetectorPy (defined in DexelaDetector)</td>
<td></td>
</tr>
<tr>
<td>DisablePulseGenerator()</td>
<td></td>
</tr>
<tr>
<td>EnableFrameCntWithinImage(ushort usEnable)</td>
<td>(unsigned short usEnable)</td>
</tr>
<tr>
<td>EnablePulseGenerator(float frequency)</td>
<td></td>
</tr>
<tr>
<td>EnablePulseGenerator()</td>
<td></td>
</tr>
<tr>
<td>EnableROIMode(bool bEnableROI)</td>
<td></td>
</tr>
<tr>
<td>GetBinningMode()</td>
<td></td>
</tr>
<tr>
<td>GetBufferXdim()</td>
<td></td>
</tr>
<tr>
<td>GetBufferYdim()</td>
<td></td>
</tr>
<tr>
<td>GetCapturedBuffer()</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td></td>
</tr>
<tr>
<td>GetDetectorStatus()</td>
<td></td>
</tr>
<tr>
<td>GetExposureMode()</td>
<td></td>
</tr>
<tr>
<td>GetExposureTime()</td>
<td></td>
</tr>
<tr>
<td>GetFieldCount(void)</td>
<td></td>
</tr>
<tr>
<td>GetFirmwareBuild(int &amp;iDayAndMonth, int &amp;iYear, int &amp;iTime)</td>
<td></td>
</tr>
<tr>
<td>GetFirmwareVersion()</td>
<td></td>
</tr>
<tr>
<td>GetFullWellMode()</td>
<td></td>
</tr>
<tr>
<td>GetGapTime()</td>
<td></td>
</tr>
<tr>
<td>GetModelNumber()</td>
<td></td>
</tr>
<tr>
<td>GetNumBuffers(void)</td>
<td></td>
</tr>
<tr>
<td>GetNumOfExposures()</td>
<td></td>
</tr>
<tr>
<td>GetReadoutMode()</td>
<td></td>
</tr>
<tr>
<td>GetReadOutTime()</td>
<td></td>
</tr>
<tr>
<td>GetROICoordinates(unsigned short &amp;usStartColumn, unsigned short &amp;usStartRow, unsigned short &amp;usROIWidth, unsigned short &amp;usROIHeight)</td>
<td></td>
</tr>
<tr>
<td>GetROIState()</td>
<td></td>
</tr>
<tr>
<td>GetSensorHeight(unsigned short uiSensorID=1)</td>
<td></td>
</tr>
<tr>
<td>GetSensorWidth(unsigned short uiSensorID=1)</td>
<td></td>
</tr>
<tr>
<td>GetSerialNumber()</td>
<td></td>
</tr>
<tr>
<td>GetTestMode()</td>
<td></td>
</tr>
<tr>
<td>GetTransportMethod()</td>
<td></td>
</tr>
<tr>
<td>GetTriggerSource()</td>
<td></td>
</tr>
<tr>
<td>gigeDet (defined in DexelaDetector)</td>
<td></td>
</tr>
<tr>
<td>GoLiveSeq(int start, int stop, int numBuf)</td>
<td></td>
</tr>
<tr>
<td>GoLiveSeq()</td>
<td></td>
</tr>
<tr>
<td>GoUnLive()</td>
<td></td>
</tr>
<tr>
<td>IsCallbackActive()</td>
<td></td>
</tr>
<tr>
<td>IsConnected()</td>
<td></td>
</tr>
<tr>
<td>IsFrameCntWithinImage()</td>
<td></td>
</tr>
<tr>
<td>IsLive()</td>
<td></td>
</tr>
<tr>
<td>LoadSensorConfigFile(char *filename)</td>
<td></td>
</tr>
<tr>
<td>MockSetter (defined in DexelaDetector)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>OpenBoard()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>OpenBoard(int NumBuifs)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>QueryBinningMode(bins flag)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>QueryExposureMode(ExposureModes mode)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>QueryFullWellMode(FullWellModes fwm)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>QueryReadoutMode(ReadoutModes mode)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>QueryTriggerSource(ExposureTriggerSource ets)</strong></td>
<td></td>
</tr>
<tr>
<td>*<em>ReadBuffer(int bufNum, byte <em>buffer)</em></em></td>
<td></td>
</tr>
<tr>
<td><strong>ReadBuffer(int bufNum, DexImage &amp;img, int iZ=0)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ReadRegister(int address, int sensorNum=1)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetBinningMode(bins flag)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetCallback(IMAGE_CALLBACK func)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetExposureMode(ExposureModes mode)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetExposureTime(float timems)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetFullWellMode(FullWellModes fwm)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetGapTime(float timems)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetNumOfExposures(int num)</strong></td>
<td></td>
</tr>
<tr>
<td>*<em>SetPreProgrammedExposureTimes(int numExposures, float <em>exposuretimes_ms)</em></em></td>
<td></td>
</tr>
<tr>
<td><strong>SetReadoutMode(ReadoutModes mode)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetROICoordinates(unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetSlowed(bool flag)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetTestMode(BOOL SetTestOn)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetTriggerSource(ExposureTriggerSource ets)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Snap(int buffer, int timeout)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SoftReset(void)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SoftwareTrigger()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>StopCallback()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ToggleGenerator(BOOL onOff)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>WaitImage(int timeout)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>WriteBuffer(int bufNum, byte *buffer)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>WriteRegister(int address, int value, int sensorNum=0)</strong></td>
<td></td>
</tr>
</tbody>
</table>
Go to the documentation of this file.

```cpp
// DexelaDetector.h : main header file for the DexelaDetector DLL

#pragma once
#ifdef _DEBUG
#pragma comment(lib,"DexelaDetector-d.lib")
#else
#pragma comment(lib,"DexelaDetector.lib")
#endif

#include <vector>
#include "DexDefs.h"
#include "DexImage.h"
#include "DexelaException.h"
#include <boost/shared_ptr.hpp>

#pragma warning(disable: 4251)

class DllExport DexelaDetector {
    typedef void (*IMAGE_CALLBACK)(int fc, int buf, DexelaDetector* det);
};
```
```cpp
private:

    void* pyData;
    char param[50];
    DexelaException* cbException;

    bool callbackActive;
    UINT CallbackCounterThread();
    IMAGE_CALLBACK callback;

    HANDLE* detHandle;

    void* callbackWorker;

    //struct containing relevant info for threads
    struct threadInfo
    {
        int FC;
        int bufNum;
        DexelaDetector* det;
        threadInfo(int fc, int BufNum, DexelaDetector* Det)
        {
            FC = fc;
            bufNum = BufNum;
            det = Det;
        }
    }

protected:

    boost::shared_ptr<baseDetector> base;
    boost::shared_ptr<gigEDetector> gigeDet;
    boost::shared_ptr<camLinkDetector> clDet;
```
public:

DexelaDetector(DevInfo &devInfo);
DexelaDetector(DetectorInterface transport, int unit, const char* params);

virtual ~DexelaDetector(void);

virtual void OpenBoard();
void OpenBoard(int NumBufs);
void CloseBoard();

int GetBufferXdim(void);
int GetBufferYdim(void);
int GetNumBuffers(void);
int GetCapturedBuffer(void);
int GetFieldCount(void);

void ReadBuffer(int bufNum, byte* buffer);
void ReadBuffer(int bufNum, DexImage &img, int iZ=0);
void WriteBuffer(int bufNum, byte* buffer);

void SetFullWellMode(FullWellModes fwm);
void SetExposureMode(ExposureModes mode);
void SetExposureTime(float timems);
void SetBinningMode(bins flag);
void SetTestMode(BOOL SetTestOn);
void SetTriggerSource(ExposureTriggerSource ets);

void SetNumOfExposures(int num);
int GetNumOfExposures();
void SetGapTime(float timems);
float GetGapTime();
bool IsConnected();
ExposureModes GetExposureMode();
float GetExposureTime();
DetStatus GetDetectorStatus();
ExposureTriggerSource GetTriggerSource();
BOOL GetTestMode();
FullWellModes GetFullWellMode();
bins GetBinningMode();
int GetSerialNumber();
int GetModelNumber();
int GetFirmwareVersion();
void GetFirmwareBuild(int& iDayAndMonth, int& iYear, int& iT ime);
DetectorInterface GetTransportMethod();
double GetReadOutTime();
//float GetReadOutTime();
bool IsCallbackActive();
bool IsLive();

void Snap(int buffer, int timeout);
int ReadRegister(int address, int sensorNum=1);
void WriteRegister(int address, int value, int sensorNum=0);
void ClearCameraBuffer(int i);
void ClearBuffers();
void LoadSensorConfigFile(char* filename);
void SoftReset(void);
void GoLiveSeq(int start, int stop, int numBuf);
void GoLiveSeq();
void GoUnLive();
void SoftwareTrigger();

void EnablePulseGenerator(float frequency);

void EnablePulseGenerator();
void DisablePulseGenerator();
void ToggleGenerator(BOOL onOff);

void WaitImage(int timeout);

void SetCallback(IMAGE_CALLBACK func);

void StopCallback();

void CheckForCallbackError();
void CheckForLiveError();

void SetPreProgrammedExposureTimes(int numExposures, float* exposuretimes_ms);

void SetROICoordinates(unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight);

void GetROICoordinates(unsigned short& usStartColumn, unsigned short& usStartRow, unsigned short& usROIWidth, unsigned short& usROIHeight);

void EnableROIMode(bool bEnableROI);

bool GetROIState();

unsigned short GetSensorHeight(unsigned short uiSensorID=1);
unsigned short GetSensorWidth(unsigned short uiSensorID=1);
bool IsFrameCntWithinImage();
void EnableFrameCntWithinImage(unsigned short usEnable);
void SetSlowed(bool flag);
void SetReadoutMode(ReadoutModes mode);
ReadoutModes GetReadoutMode();
int QueryReadoutMode(ReadoutModes mode);
int QueryExposureMode(ExposureModes mode);
int QueryTriggerSource(ExposureTriggerSource ets);
int QueryFullWellMode(FullWellModes fwm);
int QueryBinningMode(bins flag);
friend class baseBusScanner;
friend class MockSetter;
friend class DexelaDetectorPy;
friend class Dex_CL;
};

typedef void (*IMAGE_CALLBACK)(int fc, int buf, DexelaDetector* det);
DexelaDetector API

DexelaDetectorCL Member List

This is the complete list of members for DexelaDetectorCL, including all inherited members.

- **base** (defined in DexelaDetector)
  - CheckForCallbackError()
  - CheckForLiveError()
- **clDet** (defined in DexelaDetector)
  - ClearBuffers()
  - ClearCameraBuffer(int i)
  - CloseBoard()
  - DexelaDetector(DevInfo &devInfo)
  - DexelaDetector(DetectorInterface transport, int unit, const char *params)
  - DexelaDetectorCL(DetectorInterface transport, int unit, const char *params)
  - DexelaDetectorCL(DevInfo &devInfo)
  - DisablePulseGenerator()
  - EnableFrameCntWithinImage(unsigned short usEnable)
  - EnablePulseGenerator(float frequency)
  - EnablePulseGenerator()
  - EnableROIMode(bool bEnableROI)
  - GetBinningMode()
  - GetBufferXdim(void)
  - GetBufferYdim(void)
  - GetCapturedBuffer(void)
  - GetDetectorStatus()
GetExposureMode()
GetExposureTime()
GetFieldCount(void)
GetFirmwareBuild(int &iDayAndMonth, int &iYear, int &iTime)
GetFirmwareVersion()
GetFullWellMode()
GetGapTime()
GetModelNumber()
GetNumBuffers(void)
GetNumOfExposures()
GetReadoutMode()
GetReadOutTime()
GetROICoordinates(unsigned short &usStartColumn, unsigned short &usStartRow, unsigned short &usROIWidth, unsigned short &usROIHeight)
GetROIState()
GetSensorHeight(unsigned short uiSensorID=1)
GetSensorWidth(unsigned short uiSensorID=1)
GetSerialNumber()
GetTestMode()
GetTransportMethod()
GetTriggerSource()
gigeDet (defined in DexelaDetector)
GoLiveSeq(int start, int stop, int numBuf)
GoLiveSeq()
GoUnLive()
IsCallbackActive()
IsConnected()
IsFrameCntWithinImage()
IsLive()
LoadSensorConfigFile(char *filename)
OpenBoard()
OpenBoard(int NumBufs)
<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters/Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerCLIInterface</td>
<td>(bool flag)</td>
</tr>
<tr>
<td>QueryBinningMode</td>
<td>(bins flag)</td>
</tr>
<tr>
<td>QueryExposureMode</td>
<td>(ExposureModes mode)</td>
</tr>
<tr>
<td>QueryFullWellMode</td>
<td>(FullWellModes fwm)</td>
</tr>
<tr>
<td>QueryReadoutMode</td>
<td>(ReadoutModes mode)</td>
</tr>
<tr>
<td>QueryTriggerSource</td>
<td>(ExposureTriggerSource ets)</td>
</tr>
<tr>
<td>ReadBuffer</td>
<td>(int bufNum, byte *buffer)</td>
</tr>
<tr>
<td>ReadBuffer</td>
<td>(int bufNum, DexImage &amp;img, int iZ=0)</td>
</tr>
<tr>
<td>ReadRegister</td>
<td>(int address, int sensorNum=1)</td>
</tr>
<tr>
<td>SetBinningMode</td>
<td>(bins flag)</td>
</tr>
<tr>
<td>SetCallback</td>
<td>(IMAGE_CALLBACK func)</td>
</tr>
<tr>
<td>SetExposureMode</td>
<td>(ExposureModes mode)</td>
</tr>
<tr>
<td>SetExposureTime</td>
<td>(float timems)</td>
</tr>
<tr>
<td>SetFullWellMode</td>
<td>(FullWellModes fwm)</td>
</tr>
<tr>
<td>SetGapTime</td>
<td>(float timems)</td>
</tr>
<tr>
<td>SetNumOfExposures</td>
<td>(int num)</td>
</tr>
<tr>
<td>SetPreProgrammedExposureTimes</td>
<td>(int numExposures, float *exposuretimes_ms)</td>
</tr>
<tr>
<td>SetReadoutMode</td>
<td>(ReadoutModes mode)</td>
</tr>
<tr>
<td>SetROICoordinates</td>
<td>(unsigned short usStartColumn, unsigned short usStartRow, unsigned short roiWidth,</td>
</tr>
<tr>
<td></td>
<td>(unsigned short roiHeight)</td>
</tr>
<tr>
<td>SetSlowed</td>
<td>(bool flag)</td>
</tr>
<tr>
<td>SetTestMode</td>
<td>(BOOL SetTestOn)</td>
</tr>
<tr>
<td>SetTriggerSource</td>
<td>(ExposureTriggerSource ets)</td>
</tr>
<tr>
<td>Snap</td>
<td>(int buffer, int timeout)</td>
</tr>
<tr>
<td>SoftReset</td>
<td>(void)</td>
</tr>
<tr>
<td>SoftwareTrigger</td>
<td>()</td>
</tr>
<tr>
<td>StopCallback</td>
<td>()</td>
</tr>
<tr>
<td>ToggleGenerator</td>
<td>(BOOL onOff)</td>
</tr>
<tr>
<td>WaitImage</td>
<td>(int timeout)</td>
</tr>
<tr>
<td>WriteBuffer</td>
<td>(int bufNum, byte *buffer)</td>
</tr>
<tr>
<td>WriteRegister</td>
<td>(int address, int value, int sensorNum=0)</td>
</tr>
<tr>
<td>~DexelaDetector</td>
<td>(void)</td>
</tr>
</tbody>
</table>
DexelaDetector CL.h

Go to the documentation of this file.

```cpp
#pragma once

#include "dexeladetector.h"

class DllExport DexelaDetectorCL : public DexelaDetector
{
public:

    DexelaDetectorCL(DetectorInterface transport, int unit, const char* params);
    DexelaDetectorCL(DevInfo &devInfo);
    virtual ~DexelaDetectorCL(void);
    void PowerCLInterface(bool flag);

    void OpenBoard();
    void OpenBoard(int NumBufs);

};
```
This is the complete list of members for DexelaDetectorGE, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>base</strong> (defined in DexelaDetector)</td>
<td></td>
</tr>
<tr>
<td>CheckForCallbackError()</td>
<td></td>
</tr>
<tr>
<td>CheckForLiveError()</td>
<td></td>
</tr>
<tr>
<td><strong>clDet</strong> (defined in DexelaDetector)</td>
<td></td>
</tr>
<tr>
<td>ClearBuffers()</td>
<td></td>
</tr>
<tr>
<td>ClearCameraBuffer(int i)</td>
<td></td>
</tr>
<tr>
<td>CloseBoard()</td>
<td></td>
</tr>
<tr>
<td>DexelaDetector(DevInfo &amp;devInfo)</td>
<td></td>
</tr>
<tr>
<td>DexelaDetector(DetectorInterface transport, int unit, const char *params)</td>
<td></td>
</tr>
<tr>
<td>DexelaDetectorGE(DevInfo &amp;devInfo)</td>
<td></td>
</tr>
<tr>
<td>DexelaDetectorGE(DetectorInterface transport, int unit, const char *params)</td>
<td></td>
</tr>
<tr>
<td>DisablePulseGenerator()</td>
<td></td>
</tr>
<tr>
<td>EnableFrameCntWithinImage(unsigned short usEnable)</td>
<td></td>
</tr>
<tr>
<td>EnablePulseGenerator(float frequency)</td>
<td></td>
</tr>
<tr>
<td>EnablePulseGenerator()</td>
<td></td>
</tr>
<tr>
<td>EnableROIMode(bool bEnableROI)</td>
<td></td>
</tr>
<tr>
<td>GetBinningMode()</td>
<td></td>
</tr>
<tr>
<td>GetBufferXdim(void)</td>
<td></td>
</tr>
<tr>
<td>GetBufferYdim(void)</td>
<td></td>
</tr>
<tr>
<td>GetCapturedBuffer(void)</td>
<td></td>
</tr>
<tr>
<td>GetDetectorStatus()</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>GetExposureMode()</td>
<td></td>
</tr>
<tr>
<td>GetExposureTime()</td>
<td></td>
</tr>
<tr>
<td>GetFieldCount(void)</td>
<td></td>
</tr>
<tr>
<td>GetFirmwareBuild(int &amp;iDayAndMonth, int &amp;iYear, int &amp;iTime)</td>
<td></td>
</tr>
<tr>
<td>GetFirmwareVersion()</td>
<td></td>
</tr>
<tr>
<td>GetFullWellMode()</td>
<td></td>
</tr>
<tr>
<td>GetGapTime()</td>
<td></td>
</tr>
<tr>
<td>GetModelNumber()</td>
<td></td>
</tr>
<tr>
<td>GetNumBuffers(void)</td>
<td></td>
</tr>
<tr>
<td>GetNumOfExposures()</td>
<td></td>
</tr>
<tr>
<td>GetReadoutMode()</td>
<td></td>
</tr>
<tr>
<td>GetReadOutTime()</td>
<td></td>
</tr>
<tr>
<td>GetROICoordinates(unsigned short &amp;usStartColumn, unsigned short &amp;usStartRow, unsigned short &amp;usROIWidth, unsigned short &amp;usROIHeight)</td>
<td></td>
</tr>
<tr>
<td>GetROIState()</td>
<td></td>
</tr>
<tr>
<td>GetSensorHeight(unsignd short uiSensorID=1)</td>
<td></td>
</tr>
<tr>
<td>GetSensorWidth(unsignd short uiSensorID=1)</td>
<td></td>
</tr>
<tr>
<td>GetSerialNumber()</td>
<td></td>
</tr>
<tr>
<td>GetTestMode()</td>
<td></td>
</tr>
<tr>
<td>GetTransportMethod()</td>
<td></td>
</tr>
<tr>
<td>GetTriggerSource()</td>
<td></td>
</tr>
<tr>
<td>gigeDet (defined in DexelaDetector)</td>
<td></td>
</tr>
<tr>
<td>GoLiveSeq(int start, int stop, int numBuf)</td>
<td></td>
</tr>
<tr>
<td>GoLiveSeq()</td>
<td></td>
</tr>
<tr>
<td>GoUnLive()</td>
<td></td>
</tr>
<tr>
<td>IsCallbackActive()</td>
<td></td>
</tr>
<tr>
<td>IsConnected()</td>
<td></td>
</tr>
<tr>
<td>IsFrameCntWithinImage()</td>
<td></td>
</tr>
<tr>
<td>IsLive()</td>
<td></td>
</tr>
<tr>
<td>LoadSensorConfigFile(char *filename)</td>
<td></td>
</tr>
<tr>
<td>OpenBoard()</td>
<td></td>
</tr>
<tr>
<td>OpenBoard(int NumBufs)</td>
<td></td>
</tr>
</tbody>
</table>
QueryBinningMode(bins flag)
QueryExposureMode(ExposureModes mode)
QueryFullWellMode(FullWellModes fwm)
QueryReadoutMode(ReadoutModes mode)
QueryTriggerSource(ExposureTriggerSource ets)
ReadBuffer(int bufNum, byte *buffer)
ReadBuffer(int bufNum, DexImage &img, int iZ=0)
ReadRegister(int address, int sensorNum=1)
SetBinningMode(bins flag)
SetCallback(IMAGE_CALLBACK func)
SetExposureMode(ExposureModes mode)
SetExposureTime(float timems)
SetFullWellMode(FullWellModes fwm)
SetGapTime(float timems)
SetNumOfExposures(int num)
SetPersistentIPAddress(int firstByte, int secondByte, int thirdByte, int fourthByte)
SetPreProgrammedExposureTimes(int numExposures, float *exposuretimes_ms)
SetReadoutMode(ReadoutModes mode)
SetROICoordinates(unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight)
SetSlowed(bool flag)
SetTestMode(BOOL SetTestOn)
SetTriggerSource(ExposureTriggerSource ets)
Snap(int buffer, int timeout)
SoftReset(void)
SoftwareTrigger()
StopCallback()
ToggleGenerator(BOOL onOff)
WaitImage(int timeout)
WriteBuffer(int bufNum, byte *buffer)
WriteRegister(int address, int value, int sensorNum=0)
~DexelaDetector(void)
```cpp
#include "DexelaDetector.h"

class DllExport DexelaDetectorGE : public DexelaDetector
{
public:
  DexelaDetectorGE(DevInfo &devInfo);
  DexelaDetectorGE(DetectorInterface transport, int unit, const char* params);
  virtual ~DexelaDetectorGE(void);

  void SetPersistentIPAddress(int firstByte, int secondByte, int thirdByte, int fourthByte);
  void OpenBoard();
  void OpenBoard(int NumBufs);
};
```
DexelaException Member List

This is the complete list of members for DexelaException, including all inherited members.

DexelaException(const char *message, Derr code, int line, const char *filename, const char *function)
DexelaException(const DexelaException &ex, const char *function)

GetCode()
GetFileName()
GetFunctionName()
GetLineNumber()
GetTransportError()
GetTransportMessage()

LoadErrorStrings(const char *filename) (defined in DexelaException)

what() const

~DexelaException(void)
DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
</tr>
</tbody>
</table>

DexelaException.h

Go to the documentation of this file.

```cpp
#pragma once

#ifndef DEX_BUILD
#ifdef _DEBUG
#pragma comment(lib,"DexelaException-d.lib")
#else
#pragma comment(lib,"DexelaException.lib")
#endif
#endif

#include "dexdefs.h"
#include <exception>

using namespace std;

class DllExport DexelaException : public exception
{
public:
    DexelaException(const char* message, Der err code, int line, const char* filename, const char* function, int transportEr, const char* transportMessage);
    DexelaException(const DexelaException& ex, const char* function);
```
~DexelaException(void) throw();
const char* what() const throw();
Derr GetCode();
int GetTransportError();
const char* GetFileName();
int GetLineNumber();
const char* GetFunctionName();
const char* GetTransportMessage();

static void LoadErrorStrings(const char* filename);

private:
const char* _msg;
Derr _code;
const char* _filename;
int _line;
const char* _func;
int _transEr;
const char* _transMsg;

#define rethrowEr(EX) 
    throw 
    DexelaException(EX, __FUNCTION__);\n
#define throwNewEr(MSG, CODE, TRANSER, TRANSMSG) 
    throw 
    DexelaException(MSG, CODE, __LINE__, __FILE__, __FUNCTION__, TRANSER, TRANSMSG);\n
DexelaDetector API

DexImage Member List

This is the complete list of members for DexImage, including all inherited members.

AddImage()
Build(int iWidth, int iHeight, int iDepth, pType iPxType)
Build(int model, bins binFmt, int iDepth)
DefectCorrection(int DefectFlags=31)
DexImage(void)
DexImage(const char *filename)
DexImage(const DexImage &input)
FindAverageofPlanes()
FindMedianofPlanes()
FixFlood()
FloodCorrection()
FullCorrection()
GetDarkImage()
GetDarkOffset()
GetDataPointerToPlane(int iZ=0)
GetDefectMap()
GetFloodImage()
GetImageBinning()
GetImageDepth()
GetImageModel()
GetImagePixelType()
GetImagePlane(int iZ)
GetImageType()
GetImageXdim()
GetImageYdim()
GetLinearizationStarts(int &msLength)
IsEmpty()
LinearizeData()
LoadDarkImage(const DexImage &dark)
LoadDarkImage(const char *filename)
LoadDefectMap(const DexImage &defect)
LoadDefectMap(const char *filename)
LoadFloodImage(const DexImage &flood)
LoadFloodImage(const char *filename)
operator=(const DexImage &input)
PlaneAvg(int iZ=0)
ReadImage(const char *filename)
SetDarkOffset(int offset)
SetImageParameters(bins binningMode, int modelNumber)
SetImageType(DexImageTypes type)
SetLinearizationStarts(unsigned int *msArray, int msLength)
SetROIParameters(unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight)
SetScrambledFlag(bool onOff)
SubImageDefectCorrection(int startCol, int startRow, int width, int height, int CorrectionsFlag=31)
SubtractDark()
UnscrambleImage()
WriteImage(const char *filename)
WriteImage(const char *filename, int iZ)
~DexImage(void)
DexelaDetector API

DexImage.h

Go to the documentation of this file.

```cpp
// DexImage.h: Main header file for the DexImage object

#pragma once

#ifndef DEX_BUILD
#ifdef _DEBUG
#pragma comment(lib,"DexImage-d.lib")
#else
#pragma comment(lib,"DexImage.lib")
#endif
#endif

#include "DexDefs.h"
#include "DexImage.h"
#include <vector>
#include <boost/shared_ptr.hpp>

using namespace std;

#pragma warning(disable: 4251)

class DllExport DexImage
{

private:
```
boost::shared_ptr<BaseImage> baseIm;

public:

DexImage(void);
DexImage(const char* filename);
DexImage(const DexImage &input);
void operator =(const DexImage &input);
~DexImage(void);

void ReadImage(const char* filename);
void WriteImage(const char* filename);
void WriteImage(const char* filename, int iZ);
void Build(int iWidth, int iHeight, int iDepth, pType iPxType);
void Build(int model, bins binFmt, int iDepth);

void*
GetDataPointerToPlane(int iZ=0);

int GetImageXdim();
int GetImageYdim();
int GetImageDepth();
pType GetImagePixelType();
float PlaneAvg(int iZ=0);
void FixFlood();
void FindMedianofPlanes();
void FindAverageofPlanes();
void LinearizeData();
void SubtractDark();
void FloodCorrection();
void DefectCorrection(int DefectFlags=31);
void SubImageDefectCorrection(int startCol, int startRow, int width, int height, int CorrectionsFlag=31);
void FullCorrection();
void UnscrambleImage();
void AddImage();
void LoadDarkImage(const DexImage &dark);
void LoadDarkImage(const char* filename);
void LoadFloodImage(const DexImage &flood);
void LoadFloodImage(const char* filename);
void LoadDefectMap(const DexImage &defect);
void LoadDefectMap(const char* filename);
DexImage GetDarkImage();
DexImage GetFloodImage();
DexImage GetDefectMap();
DexImage GetImagePlane(int iZ);
DexImageTypes GetImageType();
void SetImageType(DexImageTypes type);
void SetDarkOffset(int offset);
int GetDarkOffset();
void SetLinearizationStarts(unsigned int* msArray, int msLength);

unsigned int* GetLinearizationStarts(int& msLength);

void SetImageParameters(bins binningMode, int modelNumber);

int GetImageModel();

bins GetImageBinning();

bool IsEmpty();

void SetScrambledFlag(bool onOff);

void SetROIParameters(unsigned short usStartColumn, unsigned short usStartRow, unsigned short usROIWidth, unsigned short usROIHeight);
GeometryCorrectionParams Member List

This is the complete list of members for GeometryCorrectionParams, including all inherited members.

iRefX  GeometryCorrectionParams
iRefY  GeometryCorrectionParams
DexelaDetector API

DexDefines.h

Go to the documentation of this file.

```c
#pragma once

typedef unsigned short ushort;
typedef unsigned long ulong;
typedef unsigned char byte;

#if _WIN32 || _WIN64
#if _WIN64
#define _X64
#else
#define _X86
#endif
#endif

#define TransMsgSize 1024

//define import for applications using the natveapi library.
#ifdef DEX_BUILD
#define DllExport __declspec( dllexport )
#else
#define DllExport __declspec( dllimport )
#endif

#ifdef DEX_BUILD_C
```
#define DllExportC __declspec( dllexport )
#else
#define DllExportC __declspec( dllimport )
#endif

#define MAX_PIXEL_VAL 16383
#define MIN_PIXEL_VAL 0
#define minTimeIncrement 0.01F
#define minTimeIncrement2 195.2F
#define ExposureSleepTimems 10
#define TimingResolution 100
#define RETURN_CHAR_LENGTH_CONST 50
#define DarkPixelXOffset 2
#define DarkPixelYOffset 4
#define AddrFPGANumber 126
#define AddrSerialNumber 125
#define AddrModelNumber 124
#define AddrGapTime 18
#define AddrNumberOfFrames 17
#define AddrFirmwareVersion 127
#define AddrTriggerSource 0
#define AddrExposureTimeLow 11
#define AddrExposureTimeHigh 12
#define AddrExposureTime 12
#define AddrExposureTime2 13
#define AddrExposureTime2Low 13
#define AddrExposureTime2High 14
#define AddrHorizontalBinReg 10
#define AddrVerticalBinReg 9
#define AddrControlReg 0
#define AddrPPExposreTime1Low 27
#define AddrPPExposreTime1High 28
#define AddrPPExposreTime2Low 29
#define AddrPPExposreTime2High 30
#define AddrPPExposreTime3Low 31
#define AddrPPExposreTime3High 32
#define AddrPPExposreTime4Low 33
#define AddrPPExposreTime4High 34
#define SerialNumberReg1 0
#define SerialNumberReg2 0
#define SerialNumberReg3 0
#define TemperatureReg 0
#define AddrWellReg 3
#define AddrWellHigh 4
#define AddrWellLow 65531
#define AddrSensorBinReg 3
#define AddrSensorBinReg2 5
#define AddrNumLines 7
#define AddrNumPixels 8
#define SensorBinClear 65087
#define DigitalBinBit 65533
#define Sensor1x1 0 //000 65087
#define Sensor1x2 0 //000 65087
#define Sensor1x4 64 //reset 65087 or 64
#define Sensor2x1 128 //128
#define Sensor2x2 128 //128
#define Sensor2x4 192 //192
#define Sensor4x1 256 //256
#define Sensor4x2 256 //256
#define Sensor4x4 320 //320
#define BinCommit 514
#define AddrReadOutTime
#define ReadoutTimeFactor1313 2
#define AddrROIStartColumn 404
#define AddrROIwidth
#define AddrROIStartRow 402
#define AddrROIheight 403
#define AddrFrameCounter 63
   //0x3F
#define AddrFramePackingMode_ImageCountPerBlock 64
   //0x40
#define AddrFramePackingMode_BlockHeightInRows 65
   //0x41
#define AddrBuildDayAndMonth 38
#define AddrBuildYear 39
#define AddrBuildTime 40
#define AddrReadOutTimeLow 55
#define AddrReadOutTimeHigh 56
#define AddrControlReg1 1
#define AddrFeaturesReg0 36
#define AddrFeaturesReg1 37
#define AVGERAGED_FLAG 1
#define FIXED_FLAG 2
#define LINEARIZED_FLAG 4
#define SORTED_FLAG 8
#define CLEAR_SORTED_FLAG 0xFFF7
#define OPERATION_KNOWN_FLAG 0x8000
#define CLEAR_OPERATION_KNOWN_FLAG 0x7FFF
#define NOOP_FLAG 0x0
#define XIS_OFFSET_CORRECTED_FLAG 1
#define XIS_GAIN_CORRECTED_FLAG 2
#define XIS_DEFECT_CORRECTED_FLAG 4
#define XIS_MULTIGAIN_CORRECTED 8
  /* this is not currently used appart from in XIS*/
#define DEX_OFFSET_CORRECTED_FLAG 16
  /* Dexela versions of the corrections*/
#define DEX_GAIN_CORRECTED_FLAG 32
#define DEX_DEFECT_CORRECTED_FLAG 64
#define DEX_EXTRA_PARAMS_FLAG 0x4000
  /* this flag will indicate the presence of new parameters (e.g. model, binning, operations) in the HIS header*/
#define CORRECTION_KNOWN_FLAG 0x8000
#define UNCORRECTED_FLAG 0x0
#define TIFFTAG_DEX_CORRECTION_FLAGS 34595 /* New tiff-tag for storing...*/
correction flags parameter */
#define TIFFTAG_DEX_OPERATION_FLAGS /* New tiff-tag for storing operation flags parameter */
#define TIFFTAG_DEX_IMAGE_TYPE /* New tiff-tag for storing image-type parameter */
#define DEX_DATA_IMAGE /* regular data image */
#define DEX_OFFSET_IMAGE /* offset data image */
#define DEX_GAIN_IMAGE /* gain data image */
#define DEX_DEFECT_MAP /* defect map image */
#define DEX_UNKONWN_TYPE_IMAGE 0xFF /* type of image is unknown */
#define TIFFTAG_DEX_MODEL_NUM /* New tiff-tag for storing image-type parameter */
#define TIFFTAG_DEX_BIN_FMT
#define TIFFTAG_ROI_START_COL
#define TIFFTAG_ROI_START_ROW
#define TIFFTAG_DEFECT_FLAGS
#define MAX_REG_ADDR 999
#define MAX_REG_VALUE 0xFFFF
#ifdef __cplusplus
#endif
```cpp
371 class BaseImage;
372 class baseDetector;
373 class gigEDetector;
374 class camLinkDetector;
375 class PleoraLib;
376 class xclib;
377 class baseBusScanner;
378
379 #endif
```
# DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>DexelaDetector</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DexelaDetector Directory Reference**
<table>
<thead>
<tr>
<th>directory</th>
<th>BusScanner</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>DexelaDetector</td>
</tr>
<tr>
<td>directory</td>
<td>DexelaException</td>
</tr>
<tr>
<td>directory</td>
<td>DexImage</td>
</tr>
</tbody>
</table>

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by doxygen 1.8.7
DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>DexelaDetector</td>
<td>BusScanner</td>
<td></td>
</tr>
</tbody>
</table>

BusScanner Directory Reference
## Files

<table>
<thead>
<tr>
<th>file</th>
<th>BusScanner.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>BusScanner.h [code]</td>
</tr>
<tr>
<td>file</td>
<td>BusScanner/resource.h [code]</td>
</tr>
</tbody>
</table>
# DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>DexelaDetector</td>
<td>DexelaException</td>
<td></td>
</tr>
</tbody>
</table>

**DexelaException Directory Reference**
## Files

<table>
<thead>
<tr>
<th>File</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>DexDefines.h</td>
<td>[code]</td>
</tr>
<tr>
<td>DexDefs.h</td>
<td>[code]</td>
</tr>
<tr>
<td>DexelaException.cpp</td>
<td></td>
</tr>
<tr>
<td>DexelaException.h</td>
<td>[code]</td>
</tr>
<tr>
<td>DexelaException/resource.h</td>
<td>[code]</td>
</tr>
</tbody>
</table>

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by doxygen 1.8.7
DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>DexelaDetector</td>
<td>DexelaDetector</td>
<td></td>
</tr>
</tbody>
</table>

DexelaDetector Directory Reference
### Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DexelaDetector.cpp</td>
<td></td>
</tr>
<tr>
<td>DexelaDetector.h</td>
<td>[code]</td>
</tr>
<tr>
<td>DexelaDetectorCL.cpp</td>
<td></td>
</tr>
<tr>
<td>DexelaDetectorCL.h</td>
<td>[code]</td>
</tr>
<tr>
<td>DexelaDetectorGE.cpp</td>
<td></td>
</tr>
<tr>
<td>DexelaDetectorGE.h</td>
<td>[code]</td>
</tr>
<tr>
<td>resource1.h</td>
<td>[code]</td>
</tr>
</tbody>
</table>

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by [doxygen](https://www.doxygen.org) 1.8.7
## DexelaDetector API

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>DexelaDetector</td>
<td>DexImage</td>
<td></td>
</tr>
</tbody>
</table>

### DexImage Directory Reference
# Files

<table>
<thead>
<tr>
<th>file</th>
<th>DexImage.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>DexImage.h [code]</td>
</tr>
<tr>
<td>file</td>
<td>DexImage/resource.h [code]</td>
</tr>
</tbody>
</table>

Generated on Tue Nov 25 2014 10:22:45 for DexelaDetector API by [doxygen](http://www.stackoverflow.com) 1.8.7