Synertronic Designs

Glaz API 9.23

Application Programming Interface

- C-style interface
- 32-bit and 64-bit libraries

Introduction

The *Glaz API* provides a programming interface for integrating *Glaz*-based measurements into your own programming environment. This includes integration into MATLAB, Python and other applications written in C, C++, C#, Java and more.

Selecting the correct API

The *Glaz API* provides a C-style interface for:

- Windows 32-bit applications
- Windows 64-bit applications
- Linux 64-bit applications (gcc)

> ⚠ Use the 32-bit API for compiling and integrating into 32-bit applications. Use the 64-bit API for compiling and integrating into 64-bit applications.

Supported platforms

Platform	Versions/Distrobutions	Tested on
Windows	Windows 7 Windows 8 Windows 10	Windows 10
Linux	Ubuntu	Ubuntu 17.04

Installing the API

API archive

Download the API zip archive from: http://www.synertronic.co.za/products/glazapi.aspx. The archive contains the following directory structure:

`GlazLib`	API base directory with C interface API
`docs`	API documentation
`example`	example source code
`include`	C-style header file
`linux64`	64-bit C-style library files for Linux
`  rules.d`	UDEV rule files
`redist`	Visual Studio redistributables
`  VS2012`	Visual Studio 2012 redistributables
`  VS2017`	Visual Studio 2017 redistributables
`win32`	32-bit C-style LIB and DLLs for Windows (VS2012, dynamically linked Qt)
`win32-static`	32-bit C-style LIB and DLLs for Windows (VS2017, statically linked Qt)
`win64`	64-bit C-style LIB and DLLs for Windows (VS2012, dynamically linked Qt)
`win64-static`	64-bit C-style LIB and DLLs for Windows (VS2017, statically linked Qt)

Windows

Follow these steps to install the API:

1. Download the zip archive with the API from: http://www.synertronic.co.za/products/glazapi.aspx
2. Extract the zip archive.
3. Copy the `include` and `winXX` (`win32`, `win64` or `win64-static`) directory of the API into the target directory, from where you will integrate the API into your environment.
4. Install the relevant Visual Studio redistributable:

`win32`	Install `redist/VS2012/vcredist_x86.exe`
`win32-static`	Install `redist/VS2017/VC_redist.x86.exe`
`win64`	Install `redist/VS2012/vcredist_x64.exe`
`win64-static`	Install `redist/VS2017/VC_redist.x64.exe`

Linux

Follow these steps to install the API:

1. Download the zip archive with the API from: http://www.synertronic.co.za/products/glazapi.aspx
2. Extract the zip archive.
3. Copy the `include` and `linux64` directory of the API into the target directory, from where you will integrate the API into your environment.
4. If the API is used by an application compiled with gcc:
   a. Add the directory where the linux *.`so.0.0.X` is located to `LD_LIBRARY_PATH`.
   b. In the *.`so` directory add a symbolic link. For example:
   ```
   ln -s GlazLib.so GlazLib.so.0.0.7
   ```
5. The *Glaz LineScan* devices use an FTDI USB interface IC. Some Linux distributions have support for these USB interfaces by default (e.g. Ubuntu) and will automatically load the VCP (virtual com port) driver when the device is connected. Unload these drivers using one of the following methods:
   - Open a terminal and after connecting the devices call:
   ```
   sudo rmmod ftdi_sio
   sudo rmmod usbserial
   ```

- Copy the `synertronic.rules` file from `linux64 rules.d` to `/etc/udev/rules.d`. The `synertronic.rules` file contains a rule to automatically unload the VCP driver.

6. Provide access rights to the *Glaz LineScan* USB devices. Copy the `synertronic.rules` file from `linux64/rules.d` to `/etc/udev/rules.d`. The `synertronic.rules` file also contains a rule to provide access rights.

API with C interface

Overview

The API is defined in the single header file `GlazLib.h` and consists of several C-style functions. The API is used as follows:

1. Initialise the API
2. Apply settings with the setter functions.
3. Capture the background (optional, only used when background subtraction is required)
4. Run a measurement.
5. Retrieve and process results.
6. Repeat from either:
    - Step 1 and initialise with new script file.
    - Step 2 with new settings.
    - Step 4 with the same settings.
7. Close the API when finisihed.

Initialise the API

The API must be initialised to start a session. Applying settings and running measurement are only possible after initialisation. There are two methods to initialise the API:

- Initialise the API with a *Glaz* script by calling **`initialiseSession`**. This method is used for multi-camera measurements or measurements involving *Glaz-PD* devices.
- Initialise the API without a script by calling **`initialiseSingleDeviceSession`**. With this method you can only connect to a single device of a specified type.

> The C-style interface does not support multiple sessions. Calling **`initialiseSession`** or **`initialiseSingleDeviceSession`** will close the previous session and disconnect from all *Glaz* devices.

Apply settings

The API provides several setter functions to set the trigger mode, trigger delay, integration time and more. Apply the relevant settings before running the next measurement.

Run a measurement

Run a measurement by calling **`runMeasurement`**. During a measurement, the *Glaz* camera will capture the specified `scanCount` number of lines with the specified level of hardware averaging. The capture lines will be averaged a processed by the *Glaz* library as specified in the script file. In single-device mode the capture lines are simply averaged. See the *Glaz LineScan* manuals for more information.

> The **`runMeasurement`** function only returns execution when the measurement is completed. For very long measurement runs the application might seem to hang until the measurement is completed.

Retrieve and process results

The API provides several getter functions to retrieve results. The **`getResult`** function is most often used and returns the averaged result of a calculation with the given index. In single-device mode there is only one calculation result with index “0”. This result is simply the average of the captured lines.

Individual scanned lines are retrieved with the **`getScan`** function. All scanned lines are retrieved with the **`getAllScans`** function. This functionality is only available when `keepscans` is enabled.

Complex results are retrieved with the **`getComplex*`** functions. These functions are only relevant when the IFFT pre-processor is used.

See the *Glaz LineScan* manuals for more information.

Close the API

It is important to call the **`close`** function at the end of the application. This will close the session and disconnect from all *Glaz* devices.

Error handling

All functions, except **`getVersion`**, return an error code. The client of the API must check the returned error code and implement the relevant actions if an error was encountered.

GlazLib.h header file

```
#define ERROR_NONE                              0
#define ERROR_NOT_INITIALISED                   1
#define ERROR_SCRIPT                            2
#define ERROR_CONNECTING_TO_CAMERAS             3
#define ERROR_DOWNLOADING_CALIBRATIONS          4
#define ERROR_INVALID_WAVELENGTHS               5
#define ERROR_INVALID_AVERAGING                 6
#define ERROR_INVALID_SCAN_COUNT                7
#define ERROR_INVALID_TRIGGER_MODE              8
#define ERROR_INVALID_TRIGGER_DELAY             9
#define ERROR_INVALID_INTEGRATION_TIME         10
#define ERROR_INVALID_SCAN_CLOCK_SPEED         11
#define ERROR_INVALID_SETTINGS                 12
#define ERROR_CAPTURING_BACKGROUNDS            13
#define ERROR_RUNNING_MEASUREMENT              14
#define ERROR_INVALID_CALCULATION_INDEX        15
#define ERROR_INVALID_RESULT_DATA_SIZE         16
#define ERROR_INVALID_PD_NUMBER                17
#define ERROR_INVALID_PD_CHANNEL               18
#define ERROR_INVALID_CAMERA_NUMBER            19
#define ERROR_INVALID_TRIGGER_FREQUENCY        20
#define ERROR_NO_MEASUREMENT_RUN               21
#define ERROR_INITIALISING_SINGEL_DEVICE       22
#define ERROR_INVALID_SINGLE_DEVICE_TYPE       23
#define ERROR_INVALID_SYNC_OUT_MODE            24
#define ERROR_INVALID_INTEGRATION_MODE         25
#define ERROR_CLOCK_SPEED_UNSUPPORTED          26
#define ERROR_INVALID_AUX_OUT_MODE             27
#define ERROR_CYCLE_COUNT_UNSUPPORTED          28
#define ERROR_INVALID_CYCLE_COUNT              29
#define ERROR_INVALID_TEST_MODE                30
#define ERROR_OUT_POLARITY_NOT_SUPPORTED       31
#define ERROR_INVALID_OUT_POLARITY             32
#define ERROR_RESOLUTION_OUT_OF_RANGE          33
#define ERROR_RESOLUTION_NOT_SUPPORTED         34
#define ERROR_RUNNING_USB_COMMS_TEST           35
#define ERROR_MEASUREMENT_STREAM               36
#define ERROR_AUX_STATES_NOT_SUPPORTED         37
#define ERROR_INTEGRATION_TIME_NOT_SUPPORTED   38
#define ERROR_INVALID_ADC_GAIN                 39
#define ERROR_AUX_CYCLE_COUNT_INVALID          40

#define GLAZ_LINESCAN_I_PULSESYNC_S10453_SINGLE_DEVICE_TYPE   1
#define GLAZ_LINESCAN_I_PULSESYNC_S11639_SINGLE_DEVICE_TYPE   2
#define GLAZ_LINESCAN_I_TIMEFILL_S11639_SINGLE_DEVICE_TYPE    3
#define GLAZ_LINESCAN_I_SPECTROCAM_S11639_SINGLE_DEVICE_TYPE  4
#define GLAZ_LINESCAN_II_SINGLE_DEVICE_TYPE                   5
#define GLAZ_LINESCAN_II_V2_SINGLE_DEVICE_TYPE                6
#define GLAZ_LINESCAN_LS_SINGLE_DEVICE_TYPE                   7
#define GLAZ_LINESCAN_EC_SINGLE_DEVICE_TYPE                   8

#define AVERAGING_X1      0
#define AVERAGING_X2      1
#define AVERAGING_X4      2
#define AVERAGING_X8      3
#define AVERAGING_X16     4
#define AVERAGING_X32     5
#define AVERAGING_X64     6
#define AVERAGING_X128    7
#define AVERAGING_X256    8
#define AVERAGING_X512    9
#define AVERAGING_X1024  10
#define AVERAGING_X2048  11
#define AVERAGING_X4096  12
```

```
#define RESOLUTION_16BIT    3
#define RESOLUTION_14BIT    2
#define RESOLUTION_12BIT    1
#define RESOLUTION_10BIT    0

#define TRIGGER_EXTERNAL    0
#define TRIGGER_INTERNAL    1
#define TRIGGER_BURST       2

#define INT_MODE_PULSESYNC  0
#define INT_MODE_TIMEFILL   1

#define OUT_INT_WINDOW              0
#define OUT_TRIGGER                 1
#define OUT_BUSY                    2
#define OUT_TRIGGER_CYCLE_START     3
#define OUT_TRIGGER_CYCLE_RUNNING   4
#define OUT_OFF                     5

#define OUT_POLARITY_ACTIVE_HI      1
#define OUT_POLARITY_ACTIVE_LO      0

#define SCAN_CLOCK_FULL_SPEED   0
#define SCAN_CLOCK_HALF_SPEED   1

#define TEST_OFF                0
#define TEST_DAC_ALTERNATING    1
#define TEST_DAC_ALL_ONES       2
#define TEST_DAC_ALL_ZEROS      3

#define ADC_GAIN_X1             0
#define ADC_GAIN_X2             1
#define ADC_GAIN_X4             2

void getVersion(int* majorVersion, int* minorVersion);

int enableDataStreamLog(bool enabled);

int initialiseSession(const char* scriptFileName);
int initialiseSingleDeviceSession(int singelDeviceType, bool keepScans, bool reverse);
int closeSession();

void resetAllDevices();
void resetAllPorts();

int setTestMode(int testMode);

int setWavelengths(double lambdaMin, double lambdaMax);
int setHardwareAveraging(int averaging);
int setResolution(int resolution);
int setScanCount(int scanCount);
int setScanClockSpeed(int speed);

int setADCGain(int gain);

int setTriggerDelay(int us);
int setTriggerMode(int mode);
int setInternalTriggerFrequency(double Hz);

int setIntegrationMode(int mode);
int setIntegrationTime(int us);

int setSyncOutMode(int mode);
int setSyncOutPolarity(int polarity);
int setAuxOutMode(int mode);
int setAuxOutPolarity(int polarity);
```

```
int setOutCycleCount(int cycleCount);

int setTimeout(int ms);

int captureBackground();
int runMeasurement();
int startMeasurement();
int isMeasurementDone(bool* isDone);
int getResult(int calculationIndex, int* count, double* values);
int getComplexResult(int calculationIndex, int* count, double* real, double* imag);

int getTimeStamp(int cameraNumber, int scanIndex, double* timestamp);
int getScan(int calculationIndex, int scanIndex, int* count, double* values);
int getComplexScan(int calculationIndex, int scanIndex, int* count, double* real, double* imag);

int getAllScansSizes(int calculationIndex, int* rowCount, int* coloumnCount);
int getAllScans(int calculationIndex, unsigned short* values);
int writeAllScansToFile(int calculationIndex, const char* filename, bool writeTimestamps);

int getPDValues(int pdNumber, int pdChannel, int* count, double* values);
int getPDReference(int pdNumber, int pdChannel, double* value);

int getAUXSTates(int cameraNumber, int* count, bool* values);

int getLastErrorMessage(char* errorMessage);
```

`void getVersion(int* majorVersion, int* minorVersion)`

Returns the version of the API.

Parameters:

`majorVersion`	Major API version number.
`minorVersion`	Minor API version number.

`int initialiseSession(const char* scriptFileName)`

Initialise the API with the given `scriptFileName`. If the API was initialised before, the previous session is closed and the API disconnects from all previously connected *Glaz* devices. *Glaz* script files are described in more detail in the *Glaz LineScan* manuals.

Parameters:

`scriptFileName`	File path of the *Glaz* script file.

Return error codes:

`ERROR_NONE`	No error and initialisation was successful.
`ERROR_SCRIPT`	The specified script was not found or contains an error.
`ERROR_CONNECTING_TO_CAMERAS`	There was an error while connecting to the devices specified in the script file. This can be caused by an USB communication error or the specified device was not found or is busy. Use the **`getLastErrorMessage`** to obtain more information about the error.
`ERROR_INVALID_SETTINGS`	The script contains an invalid combination of settings and devices. Use the **`getLastErrorMessage`** to obtain more information about the error.
`ERROR_DOWNLOADING_CALIBRATIONS`	There was an error while downloading the camera calibration from one of the target devices. Use the **`getLastErrorMessage`** to obtain more information about the error.

`int initialiseSingleDeviceSession (int singelDeviceType, bool keepScans, bool reverse)`

Initialise the API in single-device mode. If the API was initialised before, the previous session is closed and the API disconnects from all previously connected *Glaz* devices.

During single-device initialisation the *Glaz* back-end is initialised with the following script:

```
<!DOCTYPE GlazScript>"
<config>"
      <camera serial=<SN> number="1" master="1" reverse=<R>/>
      <calculation name="Camera 1" keepscans=<KS>>
          <measurement camera="1"/>
      </calculation>
</config>
```

The serial number **<SN>**, reverse **<R>** and keep-scans **<KS>** attribute are determined from the `singelDeviceType`, `reverse` and `keepScans` parameters.

Parameters:

`singelDeviceType`	Specifies the type of *Glaz* LineScan camera. Must be one of the following values (as defined at the top of the header file): `GLAZ_LINESCAN_I_PULSESYNC_S10453_SINGLE_DEVICE_TYPE` 1 `GLAZ_LINESCAN_I_PULSESYNC_S11639_SINGLE_DEVICE_TYPE` 2 `GLAZ_LINESCAN_I_TIMEFILL_S11639_SINGLE_DEVICE_TYPE` 3 `GLAZ_LINESCAN_I_SPECTROCAM_S11639_SINGLE_DEVICE_TYPE` 4 `GLAZ_LINESCAN_II_SINGLE_DEVICE_TYPE` 5 `GLAZ_LINESCAN_II_V2_SINGLE_DEVICE_TYPE` 6 `GLAZ_LINESCAN_LS_SINGLE_DEVICE_TYPE` 7 `GLAZ_LINESCAN_EC_SINGLE_DEVICE_TYPE` 8 Note: LineScan-I PulseSync S10453 was previously called the *Glaz-I*. LineScan-I TimeFill S11639 was previously called the *Glaz-S*.
`keepScans`	When set to `true` all individual scans (lines) will be stored in memory and can be accessed via the **`getScan`** functions after **`runMeasurement`** was called.
`reverse`	When set to `true`, the line pixel data is reversed.

Return error codes:

`ERROR_NONE`	No error and initialisation was successful.
`ERROR_INVALID_SINGLE_DEVICE_TYPE`	An invalid value was passed for `singelDeviceType`.
`ERROR_INITIALISING_SINGEL_DEVICE`	Unknown error while initialising the internal session.
`ERROR_CONNECTING_TO_CAMERAS`	There was an error while connecting to the devices specified in the script file. This can be caused by an USB communication error or the specified device was not found or is busy. Use the **`getLastErrorMessage`** to obtain more information about the error.
`ERROR_INVALID_SETTINGS`	Unknown error while initialising the internal session. Use the **`getLastErrorMessage`** to obtain more information about the error.
`ERROR_DOWNLOADING_CALIBRATIONS`	There was an error while downloading the camera calibration from one of the target devices. Use the **`getLastErrorMessage`** to obtain more information about the error.

`int` **`closeSession`** `()`

Closes the current session and disconnects from all connected *Glaz* devices. It is highly recommended to call this function at the end of your application.

Return error codes:

`ERROR_NONE`	Session was closed successfully.

`void resetAllDevices()`

Resets all devices. This causes the devices to re-initialise. If a session was open, it will be automatically closed.

`void resetAllPorts()`

Resets all ports. This forces a power cycle on all ports and causes the devices to re-initialise. This function is recommended if a normal reset does not work. If a session was open, it will be automatically closed.

`int setTestMode(int testMode)`

This function is intended for debugging. Enabling one of the test modes will force a known pattern when calling **`runMeasurement`**. The options are: alternating pattern between 0x0000 and 0xFFFF, fixed value at 0xFFFF or fixed value at 0x000.

Supported by:

LineScan-I, LineScan-II, LineScan-I-Gen2

Parameters:

`testMode`	Must be one of the following values (as defined at the top of the header file): `TEST_OFF` 0 `TEST_DAC_ALTERNATING` 1 `TEST_DAC_ALL_ONES` 2 `TEST_DAC_ALL_ZEROS` 3

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_TEST_MODE`	The specified test mode is not one of the values listed above.

`int setWavelengths(double lambdaMin, double lambdaMax)`

Sets the minimum and maximum wavelengths when using the IFFT pre-processor. See the *Glaz LineScan* manuals for more information.

Supported by:

All

Parameters:

`lambdaMin`	The minimum wavelength. Default = 1.0. Validation: `lambdaMin > 0.0`

`lambdaMax`	The maximum wavelength. Default = 2.0. Validation:
	`lambdaMax > 0.0`
	`lambdaMax > lambdaMin`

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_WAVELENGTHS`	The `lambdaMin` and/or `lambdaMax` parameters failed validation.

`int setHardwareAveraging(int averaging)`

Sets the hardware averaging level. See the *Glaz LineScan* manuals for more information. The supported levels of hardware averaging are device-dependent.

Supported by:

LineScan-I, LineScan-II, LineScan-I-Gen2

Parameters:

`averaging` — The hardware averaging level. Default = `AVERAGING_X1`. Must be one of the following values (as defined at the top of the header file):

AVERAGING_X1	0
AVERAGING_X2	1
AVERAGING_X4	2
AVERAGING_X8	3
AVERAGING_X16	4
AVERAGING_X32	5
AVERAGING_X64	6
AVERAGING_X128	7
AVERAGING_X256	8
AVERAGING_X512	9
AVERAGING_X1024	10
AVERAGING_X2048	11
AVERAGING_X4096	12

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_AVERAGING`	The specified hardware averaging level is invalid or not supported. Use the **`getLastErrorMessage`** to obtain more information about valid hardware averaging levels.

`int setResolution(int resolution)`

Sets the resolution (number of bits) of the measurement data.

Supported by:

LineScan-II, LineScan-I-Gen2

Parameters:

`resolution` — The resolution. Default = `RESOLUTION_16BIT`. Must be one of the following values (as defined at the top of the header file):

RESOLUTION_16BIT	3
RESOLUTION_14BIT	2
RESOLUTION_12BIT	1
RESOLUTION_10BIT	0

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_RESOLUTION_OUT_OF_RANGE`	The specified resolution is invalid. Use the correct resolution as defined above.
`ERROR_RESOLUTION_NOT_SUPPORTED`	The device does not support the specified resolution.

`int setScanCount(int scanCount)`

Sets the number of scans (lines) that will be measured during one measurement run. This is also equal to the number of scans (lines) that will be used during software averaging. See the *Glaz LineScan* manuals for more information.

Supported by:

All

Parameters:

`scanCount`	The number of scans (lines) to be measured during one measurement run. Default = 1. Validation:
	`scanCount > 0`
	`scanCount <= 4000000 (LineScan-I-Gen2, version 4.0 or higher)`
	`or scancount <= 50000 (all other LineScan devices)`

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_SCAN_COUNT`	The `scanCount` parameter failed validation.

`int setScanClockSpeed(int speed)`

Sets pixel clock scan speed. See the *Glaz LineScan-I* manual for more information.

Supported by:

LineScan-I

Parameters:

`speed`	The clock speed. Default = `SCAN_CLOCK_FULL_SPEED`. Must be one of the following values (as defined at the top of the header file):
	`SCAN_CLOCK_FULL_SPEED 0`
	`SCAN_CLOCK_HALF_SPEED 1`

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_CLOCK_SPEED_UNSUPPORTED`	Variable clock speed is not supported. It is only supported by *LineScan-I* devices.
`ERROR_INVALID_SCAN_CLOCK_SPEED`	An invalid clock speed was specified.

`int setADCGain(int gain)`

Sets the ADC gain.

Supported by:

LineScan-EC (ADC gain x1, x2 and x4), *all other devices* (only ADC gain x1)

Parameters:

`gain`	The ADC gain. Default = `ADC_GAIN_X1`. Must be one of the following values (as defined at the top of the header file):
	`ADC_GAIN_X1 0`
	`ADC_GAIN_X2 1`
	`ADC_GAIN_X4 2`

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_ADC_GAIN`	An invalid or unsupported ADC gain was specified.

`int` **`setTriggerDelay`**`(int us)`

Sets the trigger delay in [µs].

Supported by:

All

Parameters:

`us`	The trigger delay. Default = 0 us. Validation:
	`us >= 0`
	`us <= 100000`

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_TRIGGER_DELAY`	The `us` parameter failed validation.

`int` **`setTriggerMode`**`(int mode)`

Sets the trigger mode.

Supported by:

All

Parameters:

`mode`	The hardware averaging level. Default = `TRIGGER_EXTERNAL`. Must be one of the following values (as defined at the top of the header file):
	`TRIGGER_EXTERNAL 0`
	`TRIGGER_INTERNAL 1`
	`TRIGGER_BURST 2`

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_TRIGGER_MODE`	The specified trigger mode is invalid or not supported.

`int` **`setInternalTriggerFrequency`**`(double Hz)`

Sets the internal trigger frequency in [Hz]. This value is only used when the trigger mode is set to “internal trigger”. The trigger frequency range is device-dependent.

Supported by:

LineScan-I (TimeFill), LineScan-II, LineScan-I-Gen2

Parameters:

`Hz`	The internal trigger frequency. Default = 1000 Hz.

Return error codes:

`ERROR_NONE`	No error and settings were applied.

ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_TRIGGER_FREQUENCY	The specified trigger frequency falls outside the valid range. Use the **getLastErrorMessage** to obtain more information about the valid range.

`int setIntegrationMode(int mode)`

Glaz LineScan-I devices are pre-programmed with a specific integration mode (PulseSync or TimeFill) and the integration mode cannot be changed at run-time. *Glaz LineScan-II* and *LineScan-I-Gen2* devices support dynamic integration modes and the integration mode can be changed at run-time. See the *Glaz LineScan* manuals for more information on integration modes.

Supported by:

LineScan-II, LineScan-I-Gen2

Parameters:

mode	The integration mode. Default = INT_MODE_TIMEFILL (*LineScan-II* only). Must be one of the following values (as defined at the top of the header file): INT_MODE_PULSESYNC 0 INT_MODE_TIMEFILL 1

Return error codes:

ERROR_NONE	No error and settings were applied.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_INTEGRATION_MODE	The specified integration mode is invalid or not supported.

`int setIntegrationTime(int us)`

Sets the camera integration time in [µs]. The range of supported integration times is device-dependent.

Supported by:

All

Parameters:

us	The integration time. Default = 10 us.

Return error codes:

ERROR_NONE	No error and settings were applied.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_INTEGRATION_TIME	The specified integration time falls outside the valid range. Use the **getLastErrorMessage** to obtain more information about the valid range.

`int setSyncOutMode (int mode)`

Sets the output mode of the *Sync* port. See the *Glaz LineScan-II* or *LineScan-I-Gen2* manual for more information. The supported modes are device-dependent. For devices in PulseSync mode, the *Sync* output mode is automatically forced to OUT_BUSY.

Supported by:

LineScan-II, LineScan-I-Gen2

Parameters:

mode	The output mode. Default = OUT_INT_WINDOW. Must be one of the following values (as defined at the top of the header file):

OUT_INT_WINDOW	0
OUT_TRIGGER	1
OUT_BUSY	2
OUT_TRIGGER_CYCLE_START	3
OUT_TRIGGER_CYCLE_RUNNING	4
OUT_OFF	5

Return error codes:

ERROR_NONE	No error and settings were applied.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_SYNC_OUT_MODE	The specified output mode is invalid or not supported.

`int` **setSyncOutPolarity** (`int` polarity)

Sets the output polarity of the *Sync* port. See the *Glaz LineScan-II* or *LineScan-I-Gen2* manual for more information. For devices in PulseSync mode, the *Sync* output polarity is automatically forced to OUT_POLARITY_ACTIVE_LO.

Supported by:

LineScan-II, LineScan-I-Gen2

Parameters:

polarity	The output polarity. Default = OUT_POLARITY_ACTIVE_LO. Must be one of the following values (as defined at the top of the header file):

OUT_POLARITY_ACTIVE_HI	1
OUT_POLARITY_ACTIVE_LO	0

Return error codes:

ERROR_NONE	No error and settings were applied.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_OUT_POLARITY_NOT_SUPPORTED	Polarity settings are not supported by the device.
ERROR_INVALID_OUT_POLARITY	The specified polarity is not one of the values listed above.

`int` **setAuxOutMode** (`int` mode)

Sets the output mode of the *Aux* port. See the *Glaz LineScan-II* or *LineScan-I-Gen2* manual for more information.

Supported by:

LineScan-II, LineScan-I-Gen2

Parameters:

mode	The output mode. Default = OUT_INT_WINDOW. Must be one of the following values (as defined at the top of the header file):

OUT_INT_WINDOW	0
OUT_TRIGGER	1
OUT_BUSY	2
OUT_TRIGGER_CYCLE_START	3
OUT_TRIGGER_CYCLE_RUNNING	4
OUT_OFF (input)	5

Return error codes:

ERROR_NONE	No error and settings were applied.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_AUX_OUT_MODE	The specified output mode is invalid or not supported.

`int setAuxOutPolarity (int polarity)`

Sets the output polarity of the *Aux* port. See the *Glaz LineScan-II* or *LineScan-I-Gen2* manual for more information.

Supported by:

LineScan-II, LineScan-I-Gen2

Parameters:

`polarity`	The output polarity. Default = `OUT_POLARITY_ACTIVE_LO`. Must be one of the following values (as defined at the top of the header file):
	`OUT_POLARITY_ACTIVE_HI 1`
	`OUT_POLARITY_ACTIVE_LO 0`

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_OUT_POLARITY_NOT_SUPPORTED`	Polarity settings are not supported by the device.
`ERROR_INVALID_OUT_POLARITY`	The specified polarity is not one of the values listed above.

`int setOutCycleCount(int cycleCount)`

Sets the cycle count when using `OUT_TRIGGER_CYCLE_START` and `OUT_TRIGGER_CYCLE_RUNNING` output modes. See the *Glaz LineScan-II* or *LineScan-I-Gen2* manual for more information.

Supported by:

LineScan-II, LineScan-I-Gen2

Parameters:

`cycleCount`	The output cycle count. Default = 2. Validation:
	`cycleCount >= 1`
	`cycleCount <= 31`

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_CYCLE_COUNT_UNSUPPORTED`	Cycle counting is not supported. It is only supported by *LineScan-II* devices.
`ERROR_INVALID_CYCLE_COUNT`	The `cycleCount` parameter failed validation.

`int setTimeout(int ms)`

Sets the communication timeout in [ms]. When running a measurement, it can happen that devices are not triggered or that that communication is interrupted. During **`runMeasurement`**, the API will wait for the specified time-out and if no data was received from the connected devices it will return with an error code.

Supported by:

All

Parameters:

`ms`	The time-out duration. Default = 4000 ms.

Return error codes:

`ERROR_NONE`	No error and settings were applied.

ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.

`int` **captureBackground**(`int scanCount`)

Captures the background for all connected *LineScan* devices. During the background capture, `ScanCount` number of scans (lines) are measured and averaged.

Parameters:

`scanCount`	The number of scans (lines) to measure and average to capture the background.

Return error codes:

ERROR_NONE	Backgrounds were captured successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_CAPTURING_BACKGROUNDS	There was a communication error while receiving data from the connected *LineScan* devices. Check the USB cable connections. Use the **getLastErrorMessage** to obtain more information about the error.

`int` **runMeasurement**()

Starts a measurement run. The connected devices will perform a measurement with the previously specified settings. If settings were not specified, the default values are used.

This function will only return, when the measurement run is completed. A measurement run is completed after:

- All `scanCount` number of scans (lines) were captured by the *Glaz LineScan* devices, the data was received via USB and processed by the API back-end
- *or* A time-out was encountered
- *or* An error was detected.

Return error codes:

ERROR_NONE	No error and measurement was run successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_SETTINGS	An invalid combination of settings were specified. Use the **getLastErrorMessage** to obtain more information about the error.
ERROR_MEASUREMENT_STREAM	An error was detected in the data stream from the camera. You can retry by calling **runMeasurement** again. If the problem persists, check the camera USB connection.
ERROR_RUNNING_MEASUREMENT	There was a communication error while receiving data from the connected *LineScan* devices. Check the USB cable connections. Use the **getLastErrorMessage** to obtain more information about the error.

`int` **startMeasurement**()

Starts a measurement run. The connected devices will perform a measurement with the previously specified settings. If settings were not specified, the default values are used.

This function returns immediately. Call **isMeasurementDone** to check when the measurement run is completed.

Return error codes:

ERROR_NONE	No error and measurement was run successfully.

ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_SETTINGS	An invalid combination of settings were specified. Use the **getLastErrorMessage** to obtain more information about the error.

`int isMeasurementDone(bool* isDone)`

Check if the measurement run is completed. This function is used in conjunction with **startMeasurement**. A measurement run is completed after:

- All `scanCount` number of scans (lines) were captured by the *Glaz LineScan* devices, the data was received via USB and processed by the API back-end
- *or* A time-out was encountered
- *or* An error was detected.

Parameters:

isDone	*This is an out-parameter*. Returns TRUE when the measurement run is completed.

Return error codes:

ERROR_NONE	No error and measurement was run successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_MEASUREMENT_STREAM	An error was detected in the data stream from the camera. You can retry by calling **startMeasurement** again. If the problem persists, check the camera USB connection.
ERROR_RUNNING_MEASUREMENT	There was a communication error while receiving data from the connected *LineScan* devices. Check the USB cable connections. Use the **getLastErrorMessage** to obtain more information about the error.

`int getResult(int calculationIndex, int* count, double* values)`

Returns the result of a calculation after **runMeasurement** was called.

Parameters:

calculationIndex	The index of the calculation specified in the *Glaz* script file. The index is zero-based and depends on the order of calculations defined in the script file. The first calculation in the script file will have `calculationIndex = 0`, the second calculation will have `calculationIndex = 1` and so on. When the API was initialised with the **initialiseSingleDeviceSession** function, the only valid value is `calculationIndex = 0`.
count	*This is an out-parameter*. Returns the number of values in the values array. The number will be equal to the number of pixels in the camera sensor array.
values	*This is an out-parameter (array)*. Returns the result of the calculation. The array must be created by the client with a sufficient size. It is recommended to pass an array with a size of 2048.

Return error codes:

ERROR_NONE	No error and values were returned successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_CALCULATION_INDEX	The `calculationIndex` is out of range. Check the script file and determine the correct index.

ERROR_INVALID_RESULT_DATA_SIZE	The array size of the result array `values` does not match the number of pixels. This is most likely caused if a gated calculation was defined, but the calculation was never triggered. Check the gating attributes in the script file and the trigger level of the *Glaz-PD* device.
ERROR_NO_MEASUREMENT_RUN	The **runMeasurement** was not called and there are no results available.

```
int getComplexResult(int calculationIndex, int* count, double* real, double* imag)
```

Returns the complex result of a calculation after **runMeasurement** was called. A result will only be complex if the IFFT pre-processor is used.

Parameters:

calculationIndex	The index of the calculation specified in the *Glaz* script file. The index is zero-based and depends on the order of calculations defined in the script file. The first calculation in the script file will have `calculationIndex = 0`, the second calculation will have `calculationIndex = 1` and so on. When the API was initialised with the **initialiseSingleDeviceSession** function, the only valid value is `calculationIndex = 0`.
count	*This is an out-parameter*. Returns the number of values in the values array. The number will be equal to the number of pixels in the camera sensor array.
real	*This is an out-parameter (array)*. Returns the real part of the result of the calculation. The array must be created by the client with a sufficient size. It is recommended to pass an array with a size of 2048.
imag	*This is an out-parameter (array)*. Returns the imaginary part of the result of the calculation. The array must be created by the client with a sufficient size. It is recommended to pass an array with a size of 2048.

Return error codes:

ERROR_NONE	No error and values were returned successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_CALCULATION_INDEX	The `calculationIndex` is out of range. Check the script file and determine the correct index.
ERROR_INVALID_RESULT_DATA_SIZE	The array size of the result array `values` does not match the number of pixels. This is most likely caused if a gated calculation was defined, but the calculation was never triggered. Check the gating attributes in the script file and the trigger level of the *Glaz-PD* device.
ERROR_NO_MEASUREMENT_RUN	The **runMeasurement** was not called and there are no results available.

```
int getTimeStamp(int cameraNumber, int scanIndex, double* timeStamp)
```

Returns the timestamp for a given camera number and scan index. The timestamp is given in [µs].

Parameters:

cameraNumber	The number specified for a LineScan device in the *Glaz* script file.

scanIndex	The index of the scan. Validation:
	`scanIndex >= 0`
	`scanIndex < scanCount`
	Where `scanCount` is the parameter that was passed to the **setScanCount** function.

timeStamp	*This is an out-parameter*. Returns the timestamp for the specified camera number and scan index. If no timestamp is available, -1.0 is returned.

Return error codes:

ERROR_NONE	No error and values were returned successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_CAMERA_NUMBER	The cameraNumber is out of range. Check the script file and determine the correct device number.

`int getScan(int calculationIndex, int scanIndex, int* count, double* values)`

Returns the data of a specific scan (line) for a specific calculation after **runMeasurement** was called. This function will only return data if the keepscans attribute for the calculation is enabled in the script file. When the API was initialised with the **initialiseSingleDeviceSession** function, the keepScans parameter must have been set to true.

Parameters:

calculationIndex	The index of the calculation specified in the *Glaz* script file. The index is zero-based and depends on the order of calculations defined in the script file. The first calculation in the script file will have calculationIndex = 0, the second calculation will have calculationIndex = 1 and so on. When the API was initialised with the **initialiseSingleDeviceSession** function, the only valid value is calculationIndex = 0.
scanIndex	The index of the scan. Validation: scanIndex >= 0 scanIndex < scanCount Where scanCount is the parameter that was passed to the **setScanCount** function.
count	*This is an out-parameter*. Returns the number of values in the values array. The number will be equal to the number of pixels in the camera sensor array.
values	*This is an out-parameter (array)*. Returns the data of the specified calculation and scan. The array must be created by the client with a sufficient size. It is recommended to pass an array with a size of 2048.

Return error codes:

ERROR_NONE	No error and values were returned successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_CALCULATION_INDEX	Either the calculationIndex or scanIndex is out of range. Check the script file and determine the correct calculation index. Also check the scanCount parameter that was passed to the **setScanCount** function.
ERROR_INVALID_RESULT_DATA_SIZE	The array size of the result array values does not match the number of pixels. This is most likely caused if a gated calculation was defined, but the calculation was never triggered. Check the gating attributes in the script file and the trigger level of the *Glaz-PD* device.
ERROR_NO_MEASUREMENT_RUN	The **runMeasurement** was not called and there are no results available.

`int getComplexScan(int calculationIndex, int scanIndex, int* count, double* real, double* imag)`

Returns the complex data of a specific scan (line) for a specific calculation after **runMeasurement** was called. This function will only return data if the keepscans attribute for the calculation is enabled in the script file. When the API was initialised with the **initialiseSingleDeviceSession** function, the keepScans parameter must have been set to true. A result will only be complex if the IFFT pre-processor is used.

Parameters:

`calculationIndex`	The index of the calculation specified in the *Glaz* script file. The index is zero-based and depends on the order of calculations defined in the script file. The first calculation in the script file will have `calculationIndex = 0`, the second calculation will have `calculationIndex = 1` and so on. When the API was initialised with the **`initialiseSingleDeviceSession`** function, the only valid value is `calculationIndex = 0`.

`scanIndex`	The index of the scan. Validation:
	`scanIndex >= 0`
	`scanIndex < scanCount`
	Where `scanCount` is the parameter that was passed to the **`setScanCount`** function.

`count`	*This is an out-parameter*. Returns the number of values in the values array. The number will be equal to the number of pixels in the camera sensor array.
`real`	*This is an out-parameter (array)*. Returns the real part of the data of the specified calculation and scan. The array must be created by the client with a sufficient size. It is recommended to pass an array with a size of 2048.
`imag`	*This is an out-parameter (array)*. Returns the imaginary part of the data of the specified calculation and scan. The array must be created by the client with a sufficient size. It is recommended to pass an array with a size of 2048.

Return error codes:

`ERROR_NONE`	No error and values were returned successfully.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_CALCULATION_INDEX`	Either the `calculationIndex` or `scanIndex` is out of range. Check the script file and determine the correct calculation index. Also check the `scanCount` parameter that was passed to the **`setScanCount`** function.
`ERROR_INVALID_RESULT_DATA_SIZE`	The array size of the result array `values` does not match the number of pixels. This is most likely caused if a gated calculation was defined, but the calculation was never triggered. Check the gating attributes in the script file and the trigger level of the *Glaz-PD* device.
`ERROR_NO_MEASUREMENT_RUN`	The **`runMeasurement`** was not called and there are no results available.

`int` **`getAllScansSizes`**`(int calculationIndex, int* rowCount, int* coloumnCount)`

Returns the total array sizes of a specific calculation, as required for an array to be passed to the **`getAllScans`** function.

Parameters:

`calculationIndex`	The index of the calculation specified in the *Glaz* script file. The index is zero-based and depends on the order of calculations defined in the script file. The first calculation in the script file will have `calculationIndex = 0`, the second calculation will have `calculationIndex = 1` and so on. When the API was initialised with the **`initialiseSingleDeviceSession`** function, the only valid value is `calculationIndex = 0`.
`rowCount`	*This is an out-parameter*. Returns the number of scans (lines) captured for the given calculation.
`coloumnCount`	*This is an out-parameter*. Returns the number of pixels per scan (line).

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**..
`ERROR_INVALID_CALCULATION_INDEX`	The `calculationIndex` is out of range. Check the script file and determine the correct index.

`int` **`getAllScans`**`(int calculationIndex, unsigned short* values)`

Returns all the scan (line) data of a specific calculation. This function will only return data if the `keepscans` attribute for the calculation is enabled in the script file. When the API was initialised with the **`initialiseSingleDevice`** function, the `keepScans` parameter must have been set to `true`.

Parameters:

`calculationIndex`	The index of the calculation specified in the *Glaz* script file. The index is zero-based and depends on the order of calculations defined in the script file. The first calculation in the script file will have `calculationIndex = 0`, the second calculation will have `calculationIndex = 1` and so on. When the API was initialised with the **`initialiseSingleDeviceSession`** function, the only valid value is `calculationIndex = 0`.
`values`	*This is an out-parameter (array)*. Returns the data for all scans for the specified calculation. The array must be created by the client with a sufficient size. The minimum size of the array is `rowCount* coloumnCount`, as returned by the **`getAllScansSizes`** function. The data is returned as an 1D-array an is indexed as follows: `i*coloumnCount+j` , where `i` is the scan index and `j` is the pixel index.

Return error codes:

`ERROR_NONE`	No error and values were returned successfully.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_CALCULATION_INDEX`	The `calculationIndex` is out of range. Check the script file and determine the correct index.
`ERROR_INVALID_RESULT_DATA_SIZE`	The sub-array size of the result array `values` does not match the number of pixels. This is most likely caused if a gated calculation was defined, but the calculation was never triggered. Check the gating attributes in the script file and the trigger level of the *Glaz-PD* device.
`ERROR_NO_MEASUREMENT_RUN`	The **`runMeasurement`** was not called and there are no results available.

`int` **`writeAllScansToFile`**`(int calculationIndex, const char* filename, bool writeTimestamps)`

Writes all scans for the calculation with the given index to a binary file.

This function must be called before running a measurement. The scans are written to the target file while the measurement is performed.

The binary file is written in big-endian format and has the following structure if `writeTimestamps` is *false*:

```
uint16          number of scans, Ns
uint16          number of pixels, Np
Np x uint16     1. scan
Np x uint16     2. scan
…
Np x uint16     Ns. scan
```

The binary file has the following structure if `writeTimestamps` is *true*:

`4 x uint8`	preamble consisting of 4 bytes: 0x00, 0x00, 0xA5, 0xC3
`uint8`	version: 0x01
`uint16`	number of scans, Ns
`uint16`	number of pixels, Np
`uint32`	timestamp for 1. scan
`Np x uint16`	1. scan
`uint32`	timestamp for 2. scan
`Np x uint16`	2. scan
…	
`uint32`	timestamp for Ns. scan
`Np x uint16`	Ns. scan

The timestamp value can be converted to [µs] by multiplying it with the following factor:

Model	Conversion factor to [µs]
LineScan-I	0.1 (half speed) 0.05 (full speed)
LineScan-I-Gen2	0.2
LineScan-II	0.2
LineScan-LS	0.25
LineScan-EC	0.25

Parameters:

`calculationIndex`	The index of the calculation specified in the *Glaz* script file. The index is zero-based and depends on the order of calculations defined in the script file. The first calculation in the script file will have `calculationIndex = 0`, the second calculation will have `calculationIndex = 1` and so on. When the API was initialised with the **`initialiseSingleDevice`** function, the only valid value is `calculationIndex = 0`.
`filename`	File path of the target data file.
`writeTimestamps`	When set to TRUE, timestamps are written to the binary file for each captured line.

Return error codes:

`ERROR_NONE`	No error and settings were applied.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_CALCULATION_INDEX`	The `calculationIndex` is out of range. Check the script file and determine the correct index.

```
int getPDValues(int pdNumber, int pdChannel, int* count, double* values)
```

Returns the measured data for a given *Glaz-PD* device and channel after **`runMeasurement`** was called.

Parameters:

`pdNumber`	The number of the *Glaz-PD* device as specified in the *Glaz* script file.
`pdChannel`	The *Glaz-PD* channel number (either 1 or 2).
`count`	*This is an out-parameter*. Returns the number of values in the values array. The number will be equal to `scanCount` as passed to the **`setScanCount`** function.
`values`	*This is an out-parameter (array)*. Returns the measured data. The array must be created by the client with a sufficient size. The minimum size of the array is `scanCount` as passed to the **`setScanCount`** function.

Return error codes:

ERROR_NONE	No error and values were returned successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_PD_NUMBER	The pdNumber is out of range. Check the script file and determine the correct device number.
ERROR_INVALID_PD_CHANNEL	The pdChannel is out of range or the specified channel is not enabled in the script file.

`int getPDReference (int pdNumber, int pdChannel, double* value)`

Returns measured *Glaz-PD* value used for normalisation after **runMeasurement** was called. This is also the first measured value during a measurement run.

Parameters:

pdNumber	The number of the *Glaz-PD* device as specified in the *Glaz* script file.
pdChannel	The *Glaz-PD* channel number (either 1 or 2).
value	*This is an out-parameter*. Returns the reference value used for normalisation.

Return error codes:

ERROR_NONE	No error and values were returned successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**..
ERROR_INVALID_PD_NUMBER	The pdNumber is out of range. Check the script file and determine the correct device number.
ERROR_INVALID_PD_CHANNEL	The pdChannel is out of range or the specified channel is not enabled in the script file.

`int getAUXStates(int cameraNumber, int* count, double* values)`

Returns the measured *Aux* states for a given *LineScan* device after **runMeasurement** was called.

Parameters:

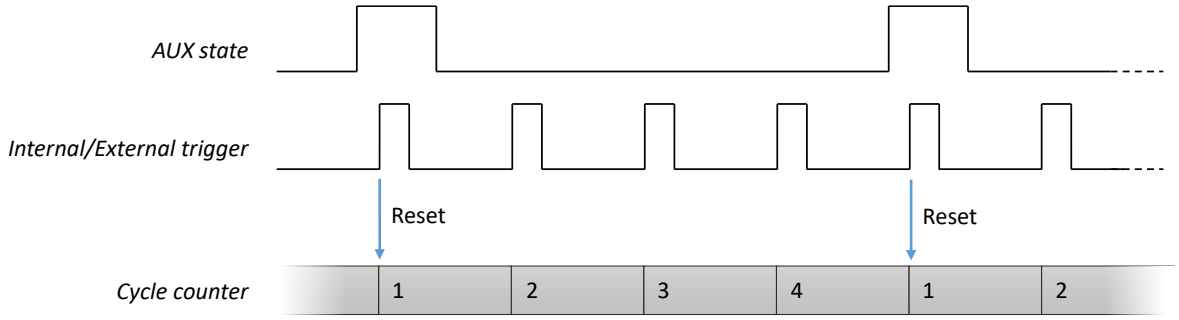
cameraNumber	The number of the *Glaz LineScan* device as specified in the *Glaz* script file.
count	*This is an out-parameter*. Returns the number of values in the values array. The number will be equal to scanCount as passed to the **setScanCount** function.
values	*This is an out-parameter (array)*. Returns the measured *Aux* states. The array must be created by the client with a sufficient size. The minimum size of the array is scanCount as passed to the **setScanCount** function. A “1” corresponds to a digital high state. A “0” corresponds to a digital low state.

Return error codes:

ERROR_NONE	No error and values were returned successfully.
ERROR_NOT_INITIALISED	The session was not initialised. First call **initialiseSession** or **initialiseSingleDeviceSession**.
ERROR_INVALID_CAMERA_NUMBER	The cameraNumber is out of range. Check the script file and determine the correct device number.
ERROR_AUX_STATES_NOT_SUPPORTED	The connected *LineScan* device does not have an *Aux* port and *Aux* states cannot be measured.

```
int getAUXCycleCounts(int cameraNumber, int maxCount, int* count, int* values)
```

Uses the *Aux* port states for a given *LineScan* device to perform a cycle count. With each trigger the cycle count is incremented. If the *Aux* state is high when triggered, the cycle count is reset to 1.



> The *Aux* port must be configured as an input when measuring external *Aux* port signals. See “setAuxOut Mode”.

Parameters:

`cameraNumber`	The number of the *Glaz LineScan* device as specified in the *Glaz* script file.
`maxCount`	The maximum expected cycle count.
`count`	*This is an out-parameter*. Returns the number of values in the values array. The number will be equal to `scanCount` as passed to the **`setScanCount`** function.
`values`	*This is an out-parameter (array)*. Returns the measured *Aux* cycle counts. The array must be created by the client with a sufficient size. The minimum size of the array is `scanCount` as passed to the **`setScanCount`** function.

Return error codes:

`ERROR_NONE`	No error and values were returned successfully.
`ERROR_NOT_INITIALISED`	The session was not initialised. First call **`initialiseSession`** or **`initialiseSingleDeviceSession`**.
`ERROR_INVALID_CAMERA_NUMBER`	The `cameraNumber` is out of range. Check the script file and determine the correct device number.
`ERROR_AUX_STATES_NOT_SUPPORTED`	The connected *LineScan* device does not have an *Aux* port and *Aux* states cannot be measured.
`ERROR_AUX_CYCLE_COUNT_INVALID`	The cycle count exceeds the specified `maxCount`.

```
int getLastErrorMessage(char* errorMessage)
```

Returns a description of the previously encountered error message. Call this function to obtain more information about the error.

Parameters:

`errorMessage`	*This is an out-parameter*. A string array containing a description of the error. The array must be created by the client with a sufficient size. The minimum recommended size is 1024. If no error was encountered previously, the string will be empty.

Return error codes:

`ERROR_NONE`	No error was encountered.

IMPORTANT NOTICE

Synertronic Designs reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to Synertronic Designs' terms and conditions of sale supplied at the time of order acknowledgment.

Synertronic Designs assumes no liability for applications assistance or customer product design. Customers are responsible for their applications using Synertronic Designs products. To minimize the risks associated with customer applications, customers should provide adequate operating safeguards.

Reproduction of information in Synertronic Designs data sheets, summary notes and brochures is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. Synertronic Designs is not responsible or liable for such altered documentation.

Synertronic Designs on the web:	www.synertronic.co.za
E-mail:	info@synertronic.co.za
Postal address:	Kaneel Cr 34 Stellenbosch 7600 South Africa