

VLDAQ

Real-Time Data Acquisition From AvibiaLine AVLX Devices

Ersteller: Avibia GmbH

Datum: 09/03/2021

Version: 1.0.0.1

Table of Contents

Overview.....	3
What is Raw Data?.....	3
Streaming in Real-Time and in HD.....	3
Streaming.....	3
Real-Time.....	3
High Definition.....	3
VLDAQ Library.....	4
Basic Program Flow.....	4
Important Notes.....	5
Example Application: DataLogger.....	5
Data Types.....	6
Raw Data Format.....	6
VLD_Device.....	6
VLD_Error.....	6
VLD_SensorCallback.....	8
VLD_CallbackReturn.....	9
Function Reference.....	10
VLD_connectDevice.....	10
VLD_disconnectDevice.....	11
VLD_getSampleRate.....	11
VLD_getConversionFactorToVolt.....	12
VLD_getAvailableSensors.....	13
VLD_getAvailableTriggers.....	14
VLD_addSensor.....	14
VLD_removeSensor.....	16
VLD_removeAllSensors.....	17
VLD_enableTriggerData.....	17
VLD_disableTriggerData.....	18
VLD_enableAllTriggerData.....	19
VLD_disableAllTriggerData.....	20
VLD_enableCallback.....	21
VLD_disableCallback.....	22
VLD_enableCallbackForAll.....	22
VLD_disableCallbackForAll.....	23
VLD_startReadout.....	24
VLD_stopReadout.....	25
VLD_fillSensorBuffers.....	26
VLD_readSensor.....	27
VLD_releaseMemory.....	29

Overview

The AvibiaLine product series is a solution for stationary vibration analysis in the context condition monitoring. The AVLX types are additionally well suited for use in research and development as well as advanced analysis techniques like artificial intelligence.

To support these use cases the AVLX devices provide an interface to read raw data directly. This document details the VLDAQ application programming interface (API) that can be used by other applications to access that data.

What is Raw Data?

Raw data are the digitized but otherwise unprocessed measurements of the acceleration sensor. These signals reflect the actual behavior of the vibration in time. They can be used in digital signal processing algorithms for example to calculate process values or to transform the signal into the frequency domain via a Fourier transformation.

The sampling of the signal is done in accordance with the Nyquist–Shannon sampling theorem so there is no aliasing within the precision of the measurement.

The AVLX devices also provide a synchronous output of trigger edges which enables the analysis of rotary frequencies and phase relations.

Streaming in Real-Time and in HD

The AVLX-HD series together with the VLDAQ library allows for the most capable transfer of data.

Streaming

AvibiaLine provides **streaming technology** so there are no predefined time limits. The user is free to decide when the data is needed and how much data is necessary. The raw data can be received continuously and **without gaps**.

Real-Time

The raw data is provided in **real-time**. The internal buffers are designed to keep the latency between the measurement and the delivery to the user program under one second. Therefore the application has to collect and process the data in a similar time frame or provide enough memory to contain all the data that will be recorded. The amount of incoming data for a fully equipped AVLX device (AVLX8 HD with 8 channels) is:

8 channels x 96 000 samples per second x 4 bytes per sample = ca. **3 Mbyte / second**. This amount of data can easily be processed by current computers.

High Definition

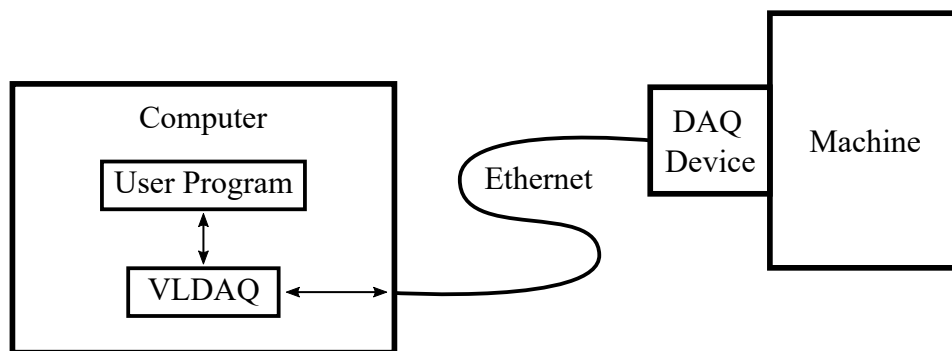
The outstanding sampling resolution of the AVLX-HD series

- of time: 96 000 Samples per channel and second and
- of signal amplitude: 24 bit = 16.777.226 steps

is the cornerstone of the high definition signal acquisition. Especially for R&D or the development of AI algorithms it is **essential** to use **HD data capture** to make smaller and faster effects visible and to be able to analyze those effects. All this additional information would be lost with a more conventional industrial resolution.

VLDAQ Library

The VLDAQ library enables a user program to access and process the raw data interface and capture signals in real-time. In order to use that interface the AvibiaLine device has to be connected via ethernet to the computer running the application. The library handles the low level data transfer and provides a simplified interface for the control of the data acquisition.



Basic Program Flow

1. Before the functions of this library can be used it is necessary to establish a connection to a device using the function [VLD_connectDevice](#).
2. Once the Device is connected it is possible to query the available sensors ([VLD_getAvailableSensors](#)) and triggers ([VLD_getAvailableTriggers](#)). Sensor properties like the sample rate ([VLD_getSampleRate](#)), or the conversion factor to Volt ([VLD_getConversionFactorToVolt](#)) can also be determined.
3. To receive data from the device a sensor has to be added to the readout using [VLD_addSensor](#). Each sensor requires a buffer that is used as the main storage for the incoming data.
4. If a trigger signal should also be recorded for this sensor, it has to be added with [VLD_enableTriggerData](#) (or [VLD_enableAllTriggerData](#) if all available triggers should be recorded).
5. If the data should be read by a callback function, it has to be added with the function [VLD_enableCallback](#).
6. Repeat steps 3-5 until the desired readout configuration is reached. Using the appropriate remove/disable functions it is also able to disable options again or remove sensors.
7. Start the readout with [VLD_startReadout](#).
8. If no callback function was set, it is necessary to repeatedly call [VLD_readSensor](#) to fetch the data that has accumulated in the buffer and prevent it from filling up.

9. Once enough data has been recorded the readout can be stopped using [VLD_stopReadout](#). As an alternative to steps 7-9 it is also possible to use [VLD_fillSensorBuffers](#). This function fills the provided buffers once and blocks until all of them are full. They can then be read using [VLD_readSensor](#).
10. Change the readout configuration as needed start new readouts. Once all the necessary data has been captured the connection to the device has to be closed using [VLD_disconnectDevice](#).

Important Notes

- The configuration of the AvibiaLine device is done exclusively using the AvibiaLine Configurator. The VLDAQ library only controls which sensors and triggers will actually get read not which ones are available to be read in the first place. In particular it is impossible to add triggers with [VLD_enableTriggerData](#) that are disabled by the configuration software. Trigger thresholds, sample rates and the calibration of the sensors are also configured using AvibiaLine Configurator.
- Changing the configuration of the AvibiaLine device using the AvibiaLine Configurator also invalidates an existing readout configuration. The device should be disconnected first and reconnected after the new configuration is in place.
- To function properly the VLDAQ library and by extension the user application needs access to the network. A potentially installed firewall has to allow the sending and receiving of TCP and UDP packets on port 29410.
- It is advisable to keep the connection between the computer running the application and the AvibiaLine device as direct as possible. Each additional node (switch, router, etc.) represents another potential error source.

Example Application: DataLogger

An example program is provided with the VLDAQ library in the folder ExampleDataLogger. This command line application connects to a device and writes all the received raw data into a file. The program is used as follows:

```
DataLogger(.exe) [IP] [duration s] [file prefix] [file extension]
```

Example: `DataLogger.exe 192.168.0.5 10 sensor_ bin`

In the example the application connects to a device with the IP 192.168.0.5 and captures data for a duration of 10 seconds. The data is written to a separate file for each sensor. The filename has the format `[file prefix][sensor index].[file extension]`. In the example that would be `sensor_0.bin` for the first sensor.

The raw data is saved without further processing (aside from setting invalid data to 0) and illustrates the data format the VLDAQ library provides. The files use little-endian ordering however so the least significant byte is placed first and the most significant byte is placed last.

Data Types

Raw Data Format

The raw data that is provided by the VLDAQ library (for example by using [VLD_readSensor](#)) is an array with entries of type `int32_t`. Each entry corresponds to one sample. Internally Avibia-Line devices use integers that are 24 bit wide. This integer gets shifted 8 bits to left and ends up in three most significant bytes of the 32 bit wide integer that gets returned by VLDAQ. The least significant byte is used to transfer the trigger signals. If no triggers are enabled for a sensor, all 8 least significant bits are 0. If triggers are being recorded, the *n*th bit will be set if trigger *n* has registered an edge (in the example trigger 1 and 2). The trigger only reacts to either the rising or the falling edge not both. Which edge is used can be configured in the AvibiaLine Configurator.

Example:

00010101	10011110	11110110	00000011
Sensor Data			0 Trigger
			3, 2, 1

To convert the raw data to Volt an additional factor is needed that can be determined by the function [getConversionFactorToVolt](#). This factor already takes the shift by 8 bit into account and can just be multiplied with the 32-bit integer. If triggers are enabled the three least significant bits of the integer have to manually be set to 0.

VLD_Device

`VLD_Device` represents one specific device inside the program. Once a connection is established, the VLDAQ library provides a pointer to a `VLD_Device` which in turn is used by the application to call all the other function for that device.

VLD_Error

The `VLD_Error` enum is used as a return value by all functions in the VLDAQ library. It signals a successful function call or describes errors that occurred.

No.	Name	Description
0	<code>VLD_success</code>	The function call was successful.
1	<code>VLD_networkTimeout</code>	A network timeout occurred. This error can be caused by the device failing to respond in time or by a network malfunction interfering with the connection. A wrong IP address can also be the reason.
2	<code>VLD_hostNotFound</code>	The specified IP address was not found.

3 VLD_connectionRefused	The connection was refused by the device.
4 VLD_connectionReset	The connection was reset by the device. This error can occur when an application tries to establish multiple connections with the device.
5 VLD_deviceNotConnected	The device is not connected.
6 VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration or start a new readout until it is finished.
7 VLD_bufferOverflow	The buffer that was provided is full.
8 VLD_outOfSync	The readout was stopped because the data was out of sync. This error can happen if the connection was interrupted for a short time or if too many errors occurred.
9 VLD_receivedNoData	The readout was stopped but no data was received. This error can be caused by a firewall, that rejects UDP packets.
10 VLD_notEnoughData	There is not enough data to fill a complete block.
11 VLD_unreleasedMemory	A block of data was already provided for this sensor and the associated memory was not yet released. Until it is released using VLD_releaseMemory no new data can be provided.
12 VLD_noUnreleasedMemory	For this sensor exists no pending data that could be released.
13 VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14 VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
15 VLD_sensorAlreadyAdded	The sensor was already added.
16 VLD_sensorNotAdded	The sensor is available but was not yet added.
17 VLD_triggerInvalid	The trigger number is not in the valid range (0-2).
18 VLD_triggerNotAvailable	The trigger number is valid but the trigger is not available on the device.
19 VLD_callbackAlreadyAssigned	There is already a callback function assigned to this sensor.
20 VLD_pointerInvalid	A passed pointer is invalid.
21 VLD_bufferInvalid	The provided buffer has an invalid size. The block size has to be at least 128 and the total length of the buffer cannot be larger than 2,000,000,000.
22 VLD_noMemory	There is not enough system memory available to

	complete the function.
23 VLD_tooManyOpenFiles	There are not enough file descriptors available to complete the function.
24 VLD_noSystemResources	There are not enough system resources to complete the function.
25 VLD_networkBusy	The network system is busy.
26 VLD_networkNotAvailable	The network system is not available.
27 VLD_winSocketNotAvailable	The Winsock API is not available (only on Microsoft Windows).

VLD_SensorCallback

The VLDAQ library allows the use of callback functions to process the data. The function is called once a block of data has been collected. The function that should be used as callback has to conform to the following function signature:

```
(VLD_CallbackReturn) ( void *context,
                        int sensorIndex,
                        uint32_t firstSample,
                        int32_t *data,
                        int dataLength,
                        bool dataContainsError)
```

Any such function can be passed to [VLD_enableCallback](#) or [VLD_enableCallback-ForAll](#) as a function pointer. VLD_SensorCallback represents that function pointer. The callback function is called by the VLDAQ library with the following arguments:

Type	Name	Description
void*	context	A user defined context that enables the function to use additional arguments. The context is passed together with the callback function when it is first registered and the library uses that context in each callback.
int	sensorIndex	Index of the sensor (0-based) for which the callback is called.
uint32_t	firstSample	Number of the first sample in the array.
int32_t*	data	Array of data.
int	dataLength	Number of samples in the data array. The callback function will only be called once a whole block of data is available. So the number of samples is always

Type	Name	Description
		equal to the block size that was provided to VLD_addSensor .
bool	dataContainsError	Indicator for errors in the data.

Once the callback function is done processing the data it has to report its status to the VLDAQ library via its return value [VLD_CallbackReturn](#).

VLD_CallbackReturn

VLD_CallbackReturn is an enum that is used as the return value of the callback function of a sensor. It signals to the VLDAQ library if the callback was successful (VLD_callbackDone) or if it failed (VLD_callbackFailed). In both cases the data that was provided in the callback will be discarded and once enough new data has accumulated another callback will be called. Additionally it is possible to signal to the library that the provided data is still needed (VLD_callbackDeferred). This can be necessary if the callback function itself calls another function asynchronously. In that case it is necessary to manually release the data using VLD_releaseMemory.

No.	Name	Description
0	VLD_callbackDone	The function was successful and the data can be discarded.
1	VLD_callbackFailed	The function failed and the data can be discarded.
2	VLD_callbackDeferred	The callback function is done but the data is still needed. The callback function for this sensor will only be called again once the pending data is released using VLD_releaseMemory .

Function Reference

Below is a list of all the functions that are provided by the VLDAQ library. The functions use [VLD_Error](#) as a return value to signal if they were successful or if an error occurred. All the possible return values are listed for each function. If the purpose of a function is to return other values it uses arguments with pointer type. The list of arguments for each function also contains a column titled IO that indicates if an argument is an input (in), output (out) or both (in, out, e.g. an input argument that gets modified).

VLD_connectDevice

Creates a connection to a device in the network. A successful call to this function is a prerequisite for the use of all the other functions in this library. The function creates a [VLD_Device](#) object that is used to address that device in subsequent function calls.

Syntax

```
VLD_Error VLD_connectDevice( const char *ip,  
                             VLD_Device **device)
```

Arguments

Type	Name	IO	Description
const char*	ip	in	IP address of the device that should be connected.
VLD_Device**	device	out	Device that was connected.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
1	VLD_networkTimeout	A network timeout occurred. This error can be caused by the device failing to respond in time or by a network malfunction interfering with the connection. A wrong IP address can also be the reason.
2	VLD_hostNotFound	The specified IP address was not found.
3	VLD_connectionRefused	The connection was refused by the device.
4	VLD_connectionReset	The connection was reset by the device. This error can occur when an application tries to establish multiple connections with the device.
20	VLD_pointerInvalid	A passed pointer is invalid.

No.	Name	Description
22	VLD_noMemory	There is not enough system memory available to complete the function.
23	VLD_tooManyOpenFiles	There are not enough file descriptors available to complete the function.
24	VLD_noSystemResources	There are not enough system resources to complete the function.
25	VLD_networkBusy	The network system is busy.
26	VLD_networkNotAvailable	The network system is not available.
27	VLD_winSocketNotAvailable	The Winsock API is not available (only on Microsoft Windows).

VLD_disconnectDevice

Prerequisites: Device is connected.

Disconnects a device and destroys the corresponding [VLD_Device](#) object. The [VLD_Device](#) pointer that was passed becomes invalid after this function is called and does not have to be freed manually.

Syntax

```
VLD_Error VLD_disconnectDevice(VLD_Device *device)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to disconnect.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
20	VLD_pointerInvalid	A passed pointer is invalid.
26	VLD_networkNotAvailable	The network system is not available.

VLD_getSampleRate

Prerequisites: Device is connected.

Returns the sample rate of a sensor.

Syntax

```
VLD_Error VLD_getSampleRate( VLD_Device *device,  
                             int  sensorIndex,  
                             int  *sampleRate)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based).
int*	sampleRate	out	Sample rate of the sensor in samples per second.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_getConversionFactorToVolt

Prerequisites: Device is connected.

Returns the conversion factor to convert the sample values to Volt.

Syntax

```
VLD_Error VLD_getConversionFactorToVolt(  
    VLD_Device *device,  
    int  sensorIndex,  
    float *conversionFactor)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in	Device to be addressed.

Type	Name	IO	Description
int	sensorIndex	in	Index of the sensor (0-based).
float*	conversionFactor	out	Conversion factor for the sensor to Volt.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_getAvailableSensors

Prerequisites: Device is connected.

Returns a list of all available sensors on the device.

Syntax

```
VLD_Error VLD_getAvailableSensors( VLD_Device *device,  
                                   const int **sensorIndexList,  
                                   int *length)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in	Device to be addressed.
const int**	sensorIndexList	out	Array of available sensor indices (e.g. [0, 3] if the first and fourth sensor are available).
int*	length	out	Length of the array of available sensor indices.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success

No.	Name	Description
5	VLD_deviceNotConnected	The device is not connected.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_getAvailableTriggers

Prerequisites: Device is connected.

Returns a list of all available triggers on the device.

Syntax

```
VLD_Error VLD_getAvailableTriggers(  
                                VLD_Device *device,  
                                const int **triggerIndexList,  
                                int *length)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in	Device to be addressed.
const int**	triggerIndexList	out	Array of available sensor indices (e.g. [0, 1] if the first and second sensor are available).
int*	length	out	Length of the array of available trigger indices.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_addSensor

Prerequisites: Device is connected and there is no readout active.

Adds a sensor to the readout. A buffer has to be provided for each sensor to receive the incoming data. The buffer is an array with entries of type `int32_t`. Each entry is one sample. The buffer is conceptually divided into multiple blocks of equal size. Data can only be read as

one complete block. The length of the array has to be at least the product of the number of blocks and the block size. If the array is bigger the additional memory will not be utilized.

There are multiple things to consider when choosing block size and the number of blocks:

- An individual block has to be at least 128 samples long.
- The total length of the buffer (that is the product of the number of blocks and the block size) has to be less than 2,000,000,000.
- If the blocks are short the data has to be processed more often. On the one hand that allows for a faster response and more frequent updates. But on the other hand that might incur some overhead if the function that processes the data makes use of some slow operations that only have to be executed once per function call (e.g. allocating memory or opening and closing files). In general these operations should be moved outside the processing function where possible (e.g. allocating memory once and reusing it or opening a file once and working with open file). But if that is not an option increasing the block size can help alleviate the problem.
- If a block contains an error the whole block is marked as invalid even if the error only affects a small subsection. If there are a lot of errors then decreasing the block size can help to get at least some data.

Syntax

```
VLD_Error VLD_addSensor( VLD_Device *device,  
                        int sensorIndex,  
                        int32_t *buffer,  
                        int blockNumber,  
                        int blockSize)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) to be added.
int32_t*	buffer	in	Buffer that will receive the data.
int	blockNumber	in	Number of blocks in the buffer.
int	blockSize	in	Size of the blocks in the buffer.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.

6 VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
13 VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14 VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
15 VLD_sensorAlreadyAdded	The sensor was already added.
20 VLD_pointerInvalid	A passed pointer is invalid.
21 VLD_bufferInvalid	The provided buffer has an invalid size. The block size has to be at least 128 and the total length of the buffer cannot be larger than 2,000,000,000.
22 VLD_noMemory	There is not enough system memory available to complete the function.

VLD_removeSensor

Prerequisites: Device is connected and there is no readout active.

Removes a sensor from the readout.

Syntax

```
VLD_Error VLD_removeSensor( VLD_Device *device,  
                             int sensorIndex
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) to be removed.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.

No.	Name	Description
16	VLD_sensorNotAdded	The sensor is available but was not yet added.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_removeAllSensors

Prerequisites: Device is connected and there is no readout active.

Removes all added sensors from the readout.

Syntax

```
VLD_Error VLD_removeAllSensors(VLD_Device *device)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_enableTriggerData

Prerequisites: Device is connected and there is no readout active.

Enables the recording of one trigger for one sensor.

Syntax

```
VLD_Error VLD_enableTriggerData( VLD_Device *device,  
                                int sensorIndex,  
                                int triggerIndex)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.

Type	Name	IO	Description
int	sensorIndex	in	Index of the sensor (0-based) for which the trigger should be recorded.
int	triggerIndex	in	Index of the trigger (0-based) to be recorded.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
16	VLD_sensorNotAdded	The sensor is available but was not yet added.
17	VLD_triggerInvalid	The trigger number is not in the valid range (0-2).
18	VLD_triggerNotAvailable	The trigger number is valid but the trigger is not available on the device.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_disableTriggerData

Prerequisites: Device is connected and there is no readout active.

Disables the recording one trigger for one sensor.

Syntax

```
VLD_Error VLD_disableTriggerData( VLD_Device *device,  
                                   int sensorIndex,  
                                   int triggerIndex)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) for which the trigger should no longer be recorded.

Type	Name	IO	Description
int	triggerIndex	in	Index of the trigger (0-based) to no longer be recorded.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
16	VLD_sensorNotAdded	The sensor is available but was not yet added.
17	VLD_triggerInvalid	The trigger number is not in the valid range (0-2).
18	VLD_triggerNotAvailable	The trigger number is valid but the trigger is not available on the device.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_enableAllTriggerData

Prerequisites: Device is connected and there is no readout active.

Enables the recording of all available triggers for one sensor.

Syntax

```
VLD_Error VLD_enableAllTriggerData( VLD_Device *device,  
                                     int sensorIndex)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) for which all available triggers should be recorded.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
16	VLD_sensorNotAdded	The sensor is available but was not yet added.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_disableAllTriggerData

Prerequisites: Device is connected and there is no readout active.

Disables the recording of all available triggers for one sensor.

Syntax

```
VLD_Error VLD_disableAllTriggerData( VLD_Device *device,  
                                     int sensorIndex)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) for which all available triggers should no longer be recorded.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
16	VLD_sensorNotAdded	The sensor is available but was not yet added.

No.	Name	Description
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_enableCallback

Prerequisites: Device is connected and there is no readout active.

Sets the callback function for one sensor. If a callback function is already assigned it will be replaced.

Syntax

```
VLD_Error VLD_enableCallback( VLD_Device *device,  
                             int sensorIndex,  
                             VLD_SensorCallback callback,  
                             void *context)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) to which a callback function should be assigned.
VLD_SensorCallback	callback	in	Callback function to be assigned.
void*	context	in	User defined context that allows the callback function to receive additional data.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
16	VLD_sensorNotAdded	The sensor is available but was not yet added.

No.	Name	Description
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_disableCallback

Prerequisites: Device is connected and there is no readout active.

Removes the callback function from one sensor.

Syntax

```
VLD_Error VLD_disableCallback( VLD_Device *device,  
                               int sensorIndex)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) from which the callback function should be removed.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
16	VLD_sensorNotAdded	The sensor is available but was not yet added.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_enableCallbackForAll

Prerequisites: Device is connected and there is no readout active.

Sets the callback function for all sensors that were added with [VLD_addSensor](#). If a callback function is already assigned it will be replaced.

Syntax

```
VLD_Error VLD_enableCallbackForAll(  
                                VLD_Device *device,  
                                VLD_SensorCallback callback,  
                                void *context)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
VLD_SensorCallback	callback	in	Callback function to be assigned.
void*	context	in	User defined context that allows the callback function to receive additional data.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_disableCallbackForAll

Prerequisites: Device is connected and there is no readout active.

Removes the callback function from all sensors that were added with [VLD_addSensor](#).

Syntax

```
VLD_Error VLD_disableCallbackForAll(VLD_Device *device)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration until it is finished.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_startReadout

Prerequisites: Device is connected and there is no readout active.

Starts the readout.

Syntax

```
VLD_Error VLD_startReadout(VLD_Device *device)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
1	VLD_networkTimeout	A network timeout occurred. This error can be caused by the device failing to respond in time or by a network malfunction interfering with the connection. A wrong IP address can also be the reason.
2	VLD_hostNotFound	The specified IP address was not found.
3	VLD_connectionRefused	The connection was refused by the device.
4	VLD_connectionReset	The connection was reset by the device. This error can occur when an application tries to establish multiple connections with the device.
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to start a new readout until it is finished.

No.	Name	Description
20	VLD_pointerInvalid	A passed pointer is invalid.
22	VLD_noMemory	There is not enough system memory available to complete the function.
23	VLD_tooManyOpenFiles	There are not enough file descriptors available to complete the function.
24	VLD_noSystemResources	There are not enough system resources to complete the function.
25	VLD_networkBusy	The network system is busy.
26	VLD_networkNotAvailable	The network system is not available.
27	VLD_winSocketNotAvailable	The Winsock API is not available (only on Microsoft Windows).

VLD_stopReadout

Prerequisites: Device is connected and a readout is in progress.

Stops the readout.

This function will not just return an error if the readout could not be stopped but also if there were any errors during the readout.

Syntax

```
VLD_Error VLD_stopReadout(VLD_Device *device)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
1	VLD_networkTimeout	A network timeout occurred. This error can be caused by the device failing to respond in time or by a network malfunction interfering with the connection. A wrong IP address can also be the reason.
2	VLD_hostNotFound	The specified IP address was not found.

No.	Name	Description
4	VLD_connectionReset	The connection was reset by the device. This error can occur when an application tries to establish multiple connections with the device.
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to change the configuration or start a new readout until it is finished.
7	VLD_bufferOverflow	The buffer that was provided is full.
8	VLD_outOfSync	The readout was stopped because the data was out of sync. This error can happen if the connection was interrupted for a short time or if too many errors occurred.
9	VLD_receivedNoData	The readout was stopped but no data was received. This error can be caused by a firewall, that rejects UDP packets.

VLD_fillSensorBuffers

Prerequisites: Device is connected.

Fills the buffers of all added Sensors once. This function blocks until all the buffers are filled.

Syntax

```
VLD_Error VLD_fillSensorBuffers(VLD_Device *device)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
1	VLD_networkTimeout	A network timeout occurred. This error can be caused by the device failing to respond in time or by a network malfunction interfering with the connection. A wrong IP address can also be the reason.
2	VLD_hostNotFound	The specified IP address was not found.

No.	Name	Description
3	VLD_connectionRefused	The connection was refused by the device.
4	VLD_connectionReset	The connection was reset by the device. This error can occur when an application tries to establish multiple connections with the device.
5	VLD_deviceNotConnected	The device is not connected.
6	VLD_readoutRunning	A readout is already in progress. It is impossible to start a new readout until it is finished.
7	VLD_bufferOverflow	The buffer that was provided is full.
8	VLD_outOfSync	The readout was stopped because the data was out of sync. This error can happen if the connection was interrupted for a short time or if too many errors occurred.
9	VLD_receivedNoData	The readout was stopped but no data was received. This error can be caused by a firewall, that rejects UDP packets.
20	VLD_pointerInvalid	A passed pointer is invalid.
22	VLD_noMemory	There is not enough system memory available to complete the function.
23	VLD_tooManyOpenFiles	There are not enough file descriptors available to complete the function.
24	VLD_noSystemResources	There are not enough system resources to complete the function.
25	VLD_networkBusy	The network system is busy.
26	VLD_networkNotAvailable	The network system is not available.
27	VLD_winSocketNotAvailable	The Winsock API is not available (only on Microsoft Windows).

VLD_readSensor

Prerequisites: Device is connected and the buffer contains enough data.

Reads data from one sensor. The number of returned samples is equal to the block size of the buffer. Data can only be read if one whole block of data available. If there is not enough data an error will be returned. Once the application has finished processing the data it has to be released with [VLD_releaseMemory](#).

Syntax

```
VLD_Error VLD_readSensor( VLD_Device *device,  
                           int sensorIndex,  
                           int32_t **data,  
                           bool *dataContainsError)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) from which data should be read.
int32_t**	data	out	Array of data. The length of the array is equal to the block size of the buffer.
bool*	dataContainsError	out	Indicator for errors in the data.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
10	VLD_notEnoughData	There is not enough data to fill a complete block.
11	VLD_unreleasedMemory	A block of data was already provided for this sensor and the associated memory was not yet released. Until it is released using VLD_releaseMemory no new data can be provided.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
16	VLD_sensorNotAdded	The sensor is available but was not yet added.
19	VLD_callbackAlreadyAssigned	There is already a callback function assigned to this sensor.
20	VLD_pointerInvalid	A passed pointer is invalid.

VLD_releaseMemory

Prerequisites: Device is connected and there is pending data.

Releases pending data for one sensor.

The VLDAQ library gives the application access to the data inside the buffers. If the application uses the function [VLD_readSensor](#) or a [callback function](#) that returns `VLD_CallBackDeferred` there is no way for the library to detect when the corresponding memory can be reused. Therefore the application has to signal to the library that it has finished processing the data by calling this function.

Syntax

```
VLD_Error VLD_releaseMemory( VLD_Device *device,  
                             int sensorIndex)
```

Arguments

Type	Name	IO	Description
VLD_Device*	device	in, out	Device to be addressed.
int	sensorIndex	in	Index of the sensor (0-based) for which pending data should be released.

Return Values

Type: VLD_Error

No.	Name	Description
0	VLD_success	Success
12	VLD_noUnreleasedMemory	For this sensor exists no pending data that could be released.
13	VLD_sensorInvalid	The sensor number is not in the valid range (0-7).
14	VLD_sensorNotAvailable	The sensor number is valid but the sensor is not available on the device.
15	VLD_sensorAlreadyAdded	The sensor was already added.
16	VLD_sensorNotAdded	The sensor is available but was not yet added.
20	VLD_pointerInvalid	A passed pointer is invalid.