MEMBER OF THE NYNOMIC GROUP

AVATECH LIBRARY

SOFTWARE

Operation Manual

NEED TECHNICAL SUPPORT?

Scan the **QR-code** or visit **www.avantes.com/support**

We are happy to help you!

ID: 036252

AvaTech Library

Interface Package for Windows Applications

Interface Package for Linux Applications

Interface Package for macOS Applications

Version 1.2.0.0

USER'S MANUAL

July 2024

Avantes B.V.

Oude Apeldoornseweg 28
NL-7333 NS Apeldoorn
The Netherlands
Tel: +31-313-670170
Fax: +31-313-670179

Web: www.avantes.com
Email: info@avantes.com

ID: 036252

Microsoft, Visual C++, Visual Basic, Visual C#, Windows and Microsoft Office are registered trademarks of the Microsoft Corporation. Windows 8, Windows 10 and Windows 11 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

LabVIEW is a trademark of the National Instruments Corporation.
MATLAB is a registered trademark of The MathWorks, Inc.
Qt is a trademark of the Qt Company Ltd in Finland and/or other countries worldwide.

Software License

THE INFORMATION AND CODE PROVIDED BELOW (COLLECTIVELY REFERRED TO AS “SOFTWARE”) IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL AVANTES BV OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUSINESS PROFITS OR SPECIAL DAMAGES, EVEN IF AVANTES BV OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE FOREGOING LIMITATION MAY NOT APPLY.

This Software gives you the ability to write applications to acquire and process data from Avantes equipment. You have the right to redistribute the libraries contained in the Software, subject to the following conditions:

1. You are developing applications to control Avantes equipment. If you use the code contained herein to develop applications for other purposes, you MUST obtain a separate software license.
2. You distribute only the drivers necessary to support your application.
3. You place all copyright notices and other protective disclaimers and notices contained on the Software on all copies of the Software and your software product.
4. You or your company provides technical support to the users of your application. Avantes B.V. will not provide software support to these customers.
5. You agree to indemnify, hold harmless, and defend Avantes B.V. and its suppliers from and against any claims or lawsuits, including attorneys’ fees that arise or result from the use or distribution of your software product and any modifications to the Software.

ID: 036252

1 Installation

1.1 Installation on Windows

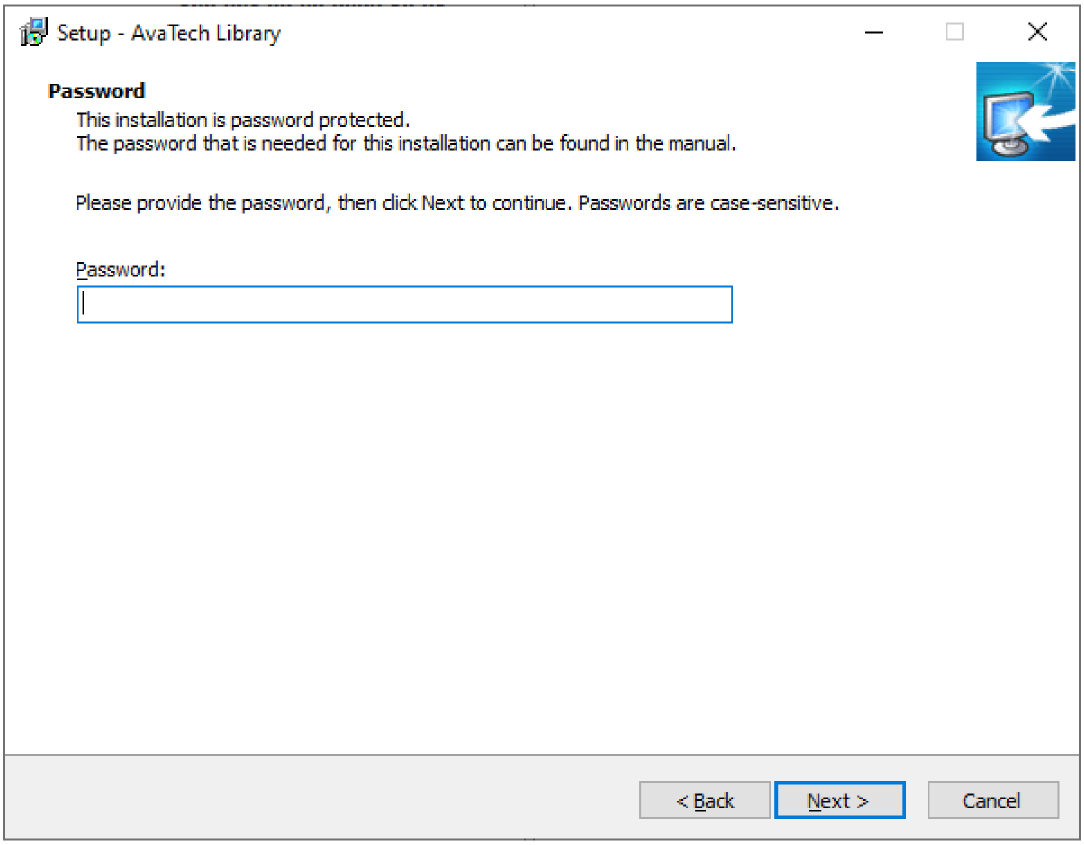
The AvaTech-DLL package is the 64-bit version of the AvaTech driver interface. It is needed when you want the Library to cooperate with 64-bit Windows programs, like self-written 64-bit programs. The AvaTech-DLL package version can be installed under 64-bit Windows 8/Windows 10 and Windows 11

The installation program for the AvaTech-DLL package can be started by running the file “AvaTech_Library_w.x.y.z.exe” that can be downloaded.

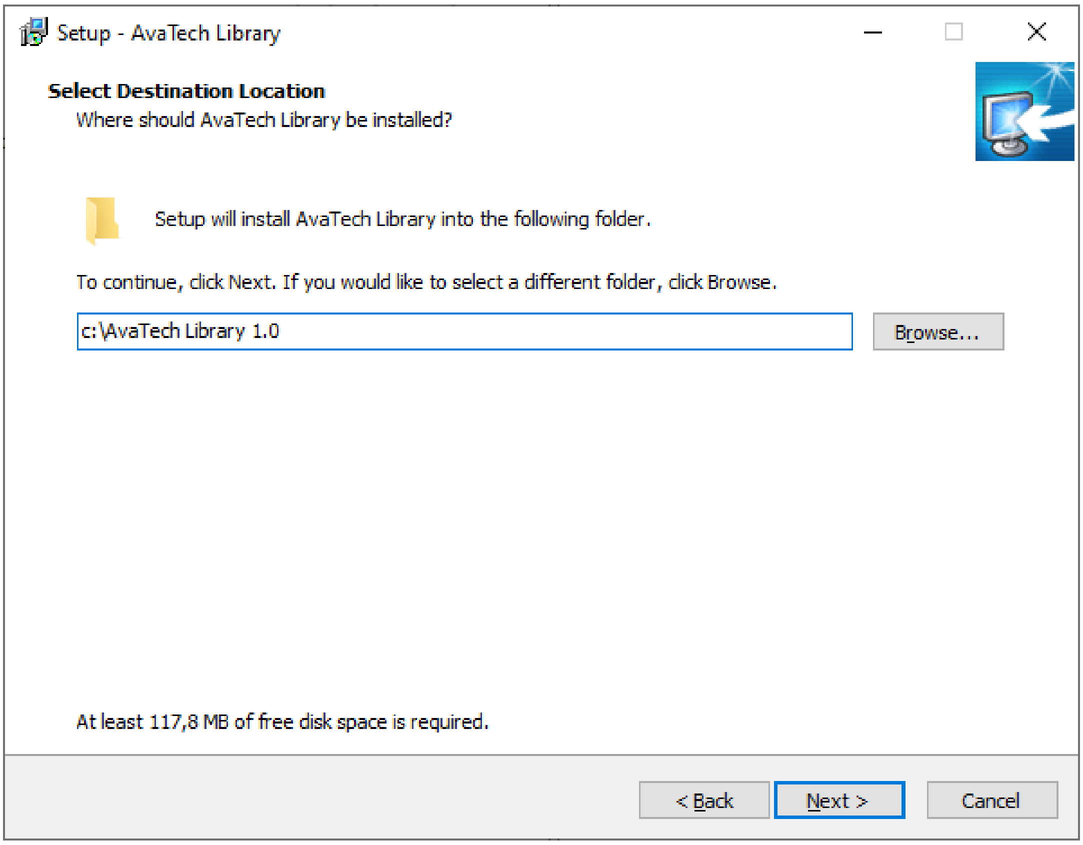
This manual describes the installation, functions and sample programs of AvaTech Library.

This installation is password protected. Enter the following password to proceed with the installation:

Password: `Avantes6961LL4a`

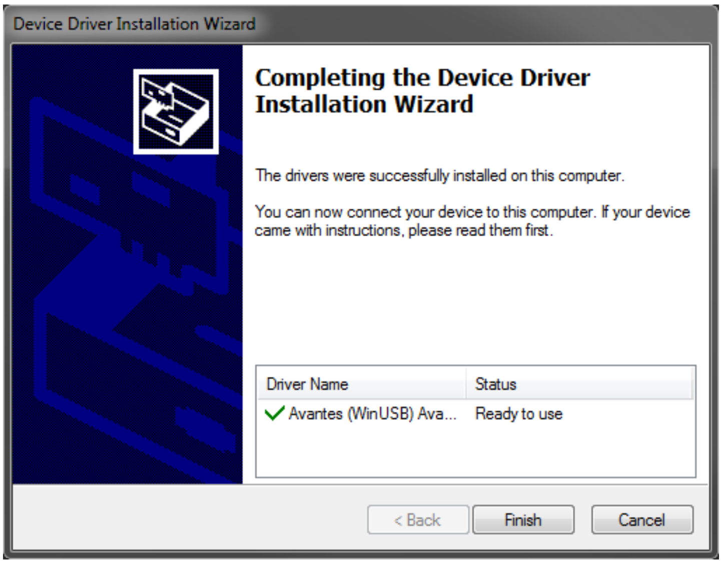


In the next dialog, the destination directory for the AvaTech Library package can be selected. The default destination directory is `C:\AvaTech Library <version_nr>`. If you want to install the software to a different directory, click the Browse button, select a new directory, and click OK.



If the specified directory does not exist, it will be created.

Next, the WinUSB device driver will be installed for the spectrometer. The Device Driver Installation Wizard will be launched automatically. The last dialog in the Device Driver Installation Wizard displays whether the WinUSB driver has been installed correctly.



After all files have been installed, the “Installation Complete” dialog shows up. Click Finish.

Connecting the hardware

Connect the USB connector to a USB port on your computer with the supplied USB cable. Modern versions of Windows will install the driver silently, without displaying dialogs.

Launching the software

This AvaTech Library manual, together with the source code of the example programs can be found in the installation directory. Basic examples are written for C, C++, C#, and Python. A more extensive example program is available in C++/QT. All example programs include source code.

1.2 Installation on Linux

The AvaTech Library is a dynamically linked shared library for the Linux operating system. It uses the *libusb* library for USB communications. The AvaTech Linux Library supports the AS7007 board. At present, the library is available in binary form only. Binary versions are available for several Linux versions. Please contact Avantes when a binary version is needed for a specific Linux version or distribution that is not available in the current Avantes portfolio.
For some versions of Linux, so-called .deb files are available. These can be used to install the library to the correct folders on your system.
The syntax to use is:
'sudo dpkg –i <name>.deb' (do not enter the single quotes)

If a .deb file does not work for you, a .tar.gz file can be used as an alternative. This can be unpacked and the files can be placed where the operating system expects them.

Sample programs using the library are also available. Basic examples are written for C, C++, C# and Python. A more extensive example program is available in C++/QT. All example programs include source code.

Connecting the hardware

Connect the USB connector to a USB port on your computer with the supplied USB cable.

Note: Depending on the Linux distribution, the standard user may not have permissions to open USB ports. One solution is to run the application as root, a better solution is to add permissions to specific USB devices. How this is done varies by Linux distribution, Appendix B describes the procedure for Ubuntu.

1.3 Installation on macOS

The AvaTech Library is a dynamically linked shared library for the macOS operating system. It uses the *libusb* library for USB communications. The AvaTech Library supports the AS7007 board.

Sample programs using the library are also available. Basic examples are written for C, C++, C# and Python. A more extensive example program is available in C++/QT. All example programs include source code.

Connecting the hardware
Connect the USB connector to a USB port on your computer with the supplied USB cable.

2 AvaTech Library Description

2.1 Core library versus Full library.

The AvaTech library is available in two editions:

- The core library with all functions needed for operation of the spectrometer. Source code (in C/C++) will be available for this edition.
- The full Library, which will contain all functions of the core DLL and some additional functions, for which only a binary version will be available. Reasons to not include source code can be things like the presence of intellectual property or security reasons and customer support considerations. E.g., features like stray light correction or dynamic dark correction are present in the full DLL only. If you insist on having source code here, you will have to develop your own version of these features.

Functions present in both versions:

- AVT_Get
- AVT_GetMultiple
- AVT_Set
- AVT_SetMultiple
- AVT_Activate
- AVT_Deactivate
- AVT_Init
- AVT_Done
- AVT_GetNrOfDevices
- AVT_GetList
- AVT_SetLogging
- AVT_GetVersion
- AVT_GetBuildInfo
- AVT_Trigger
- AVT_SetCallback
- AVT_SetDeviceMonitorCallback
- AVT_ExtractTriggerData

Functions present in the Full library only:

- AVT_SetSmoothing
- AVT_EnableDynamicDarkCorrection
- AVT_EnableStrayCorrection
- AVT_GetErrorMessage
- AVT_ProcessMeasurementData

2.2 Interface Overview

The AvaTech Library is a set of functions used to communicate with one or more spectrometer devices.

All values and data can be read or written using datapoints. In addition to reading (get) and writing (set), it is possible to trigger a datapoint. This is intended to perform an action on the spectrometer, for example, starting a measurement. The process for setting the measurement integration time, the number of averages or a digital output pin is identical, you just have to provide a different datapoint. This also applies to reading these values back, and to triggering of datapoints.

Datapoints are categorized. All datapoints have a unique combination of a category and a tag. For a number of programming languages, header files are available with structs or enumerations for all available categories and tags. An overview of available datapoints can be found in chapter

Datapoints.

All communication is initiated by the host PC, with one exception. Measurement data can be sent from spectrometer to host at the spectrometer's initiative. The user has to provide a callback function in order to handle this data. This is explained in more detail in Getting measurement data.

2.3 Usage of the AvaTech Library

The library uses a single pair of open and close functions (AVT_Init() and AVT_Done()) that have to be called by the application. As long as the open function has not yet been successfully called, all other functions will return an error code.

The open function (AVT_Init()) tries to open a communication port for all connected devices. Please note that you will need administrative rights to open a USB port under Linux.
The close function (AVT_Done()) closes the communication port(s) and releases all internal data storage.

The interface between the application and the library can be divided in four functional groups:

- Internal data read functions, which read device configuration data from the internal Library storage.
- Blocking control functions which send a request to the device and wait until an answer is received or a time-out occurs before returning control to the application.
- Non-blocking data read functions, which send a request to the device and then return control to the application. After the answer from the device is received or a timeout occurs, a notification is sent to the application.
- Data send functions which send device configuration data to the device.

After the application has initialized it should select the spectrometer(s) it wants to use. This procedure is explained in more detail in the following sections.

2.3.1 Activating a USB Device

For a USB connected device, the following steps have to be taken:

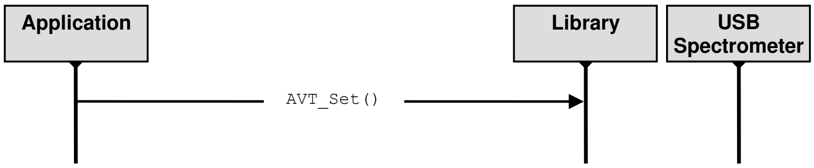
1. First call AVT_Init(0) to initialize the library to use with USB spectrometers.
2. Call AVT_GetNrOfDevices() to determine the number of attached devices.
3. Allocate a buffer to store identity info;
   ```
   RequiredSize = NrDevices * sizeof(AvtDeviceInfo)
   ```
   NrDevices is obtained in the previous step.
4. Call AVT_GetList() with the number of connected spectrometers. The resulting device list will be sorted on serial number.
5. Select the spectrometers you want to use with AVT_Activate()
6. Define a callback function that will be called by the library when spectral data are available, and pass the address of this function to the library with the AVT_SetCallback() function.

2.4 Interact with spectrometer

There are three main functions that make it possible to exchange data with a spectrometer.

2.4.1 Setting a datapoint value

Datapoints that are writable can be set using the AVT_Set function.



PARAMETERS

`uint32_t a_device_id [in]`

A device handle. The handle is the connection ID for an active device. The handle can be obtained by AVT_Activate()

`uint32_t a_category [in]`

The category number of the datapoint. Can be found in chapter 7 Datapoints.

`uint32_t a_tag [in]`

The tag number of the datapoint. Can be found in chapter 7 Datapoints.

`uint32_t a_length [in]`

The length of the data that is set. The length must match the size accepted by the datapoint. This can be found in chapter 7 Datapoints.

`uint8_t* a_data [in]`

The data that is set to the datapoint.

RETURNS

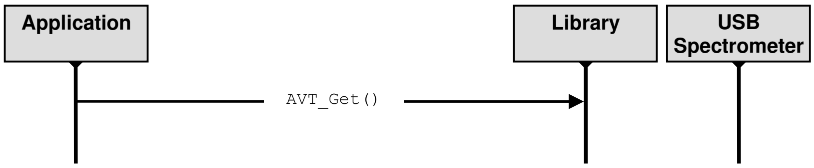
Return value is AVT_STATUS, AVT_SUCCESS when the set is succeeded, otherwise an error value containing a reason for failure.

EXAMPLE

```
char user_friendly_name[] spectrometer_name = "Spectrometer 1";
AVT_Set(m_device_id, AVTGenericTags::CATEGORY,
AVTGenericTags::USERFRIENDLYNAME::TAG,
      sizeof(spectrometer_name), spectrometer_name);
```

2.4.2 Getting a datapoint value

Datapoints that are readable can be get using the AVT_Get function.



PARAMETERS

`uint32_t a_device_id [in]`

A device handle. The handle is the connection ID for an active device. The handle can be obtained by AVT_Activate()

`uint32_t a_category [in]`

The category number of the datapoint. Can be found in chapter 7 Datapoints.

`uint32_t a_tag [in]`

The tag number of the datapoint. Can be found in chapter 7 Datapoints.

`uint32_t a_length [in]`

The length of the data that is get. The length must match the size accepted by the datapoint. This can be found in chapter 7 Datapoints.

`uint8_t* a_data [out]`

Pointer to data buffer where the retrieved data can be stored.

RETURNS

Return value is AVT_STATUS, AVT_SUCCESS when the set is succeeded, otherwise an error value containing a reason for failure.

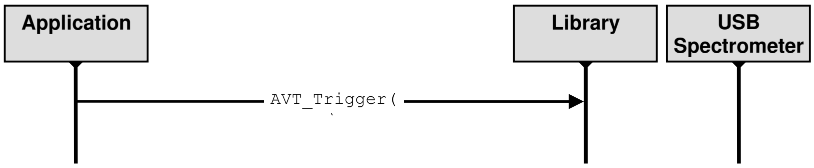
EXAMPLE

```
char user_friendly_name[64] spectrometer_name;

AVT_Get(m_device_id, AVTGenericTags::CATEGORY,
AVTGenericTags::USERFRIENDLYNAME::TAG,
      sizeof(spectrometer_name), spectrometer_name);
```

2.4.3 Triggering a datapoint

Datapoints that are of type `trigger' can be triggered using the AVT_Trigger function. By triggering a datapoint, an action is performed on the spectrometer. Some trigger datapoints require data sent with the trigger, other datapoints do not require data.



PARAMETERS

`uint32_t a_device_id [in]`

A device handle. The handle is the connection ID for an active device. The handle can be obtained by AVT_Activate()

`uint32_t a_category [in]`

The category number of the datapoint. Can be found in chapter 7 Datapoints.

`uint32_t a_tag [in]`

The tag number of the datapoint. Can be found in chapter 7 Datapoints.

`uint32_t a_length [in]`

The length of the data that is get. The length must match the size accepted by the datapoint. This can be found in chapter 7 Datapoints.

`uint8_t* a_data [in]`

Pointer to data buffer where the retrieved data can be stored.

RETURNS

Return value is AVT_STATUS, AVT_SUCCESS when the set is succeeded, otherwise an error value containing a reason for failure.

EXAMPLE

```
AVT_Trigger(m_device_id, AVTMeasurementTags::CATEGORY,
AVTMeasurementTags::Start::TAG,
      0, NULL);
```

2.4.4 Setting or getting multiple datapoints

For simplicity reasons, it is possible to set or get multiple datapoints at once using AVT_Set_multiple() or AVT_Get_multiple(). These functions use the AVTData structure.

```
typedef struct
{
      uint32_t category;
      uint32_t tag;
      uint32_t length;
      uint8_t* data;
      AVT_STATUS status;
} AvtData
```

EXAMPLE

```
vector<AvtData> avtData_list;
uint16_t start_pixel = 10;
uint16_t stop_pixel = 3786;
avtData_list.emplace_back(AvtData{
        AVTMeasurementTags::CATEGORY,
        AVTMeasurementTags::STOPPIXEL,
        sizeof(stop_pixel),
        reinterpret_cast<std::uint8_t*>(&m_stop_pixel),
        AVT_STATUS::AVT_ERROR
    });
    avtData_list.emplace_back(AvtData{
        AVTMeasurementTags::CATEGORY,
        AVTMeasurementTags::STARTPIXEL,
        sizeof(start_pixel),
        reinterpret_cast<std::uint8_t*>(&m_start_pixel),
        AVT_STATUS::AVT_ERROR
    });
//Send start and stop pixel together.
AVT_Set_Multiple(m_first_device_id, avtData_list.data(),
            static_cast<std::uint32_t>(avtData_list.size()))
```

If all set actions succeed, AVT_Set_Multiple will return AVT_SUCCESS.
If one of the Set actions fail AVT_Set_Multiple will return AVT_FAIL, the status field in the AvtData structure will contain an error code to indicate which set actions succeeded and which set actions failed.

EXAMPLE

```
std::uint8_t detectorType;
std::uint16_t number_of_pixels;
vector<AvtData> avtData_list;

std::vector<AvtData> detectorData = {
                AvtData {
                    AVTDetectorTags::CATEGORY,
                    AVTVersionTags::TYPE,
                    sizeof(detectorType),
                    reinterpret_cast<uint8_t*>(&detectorType),
                    AVT_STATUS::AVT_ERROR
                },
                AvtData {
                    AVTDetectorTags::CATEGORY,
                    AVTDetectorTags::NUMBER_OF_PIXELS,
                    sizeof(spectrOSVersion),
                    reinterpret_cast<uint8_t*>(&spectrOSVersion),
```

```
                        AVT_STATUS::AVT_ERROR
                    },
```

AVT_Get_Multiple(m_first_device_id, detectorData.data(),
static_cast<std::uint32_t>(versionData.size())

If all get actions succeed, AVT_Get_Multiple will return AVT_SUCCESS.
If one of the get actions fail AVT_Get_Multiple will return AVT_FAIL, the status field in the AvtData structure will contain an error code to indicate which set actions succeeded and which set actions failed.

2.4.5 Getting measurement data

There are two ways to get measurement data from the spectrometer. The first is by using a callback function, the second is by polling for measurement data.

2.4.5.1 Getting measurement data by using a callback function.

AVT_SetCallback can be used to provide a callback function that is invoked as soon as the spectrometer has data available for a specific trigger datapoint. By setting a callback for datapoint measurement.measurement_data, the callback is called when the spectrometer has measurement data available.

The callback function should have the following arguments:
void callback(std::uint32_t a_handle, AvtData* a_data, std::uint32_t a_num_items);

When measurement data is available the callback function is called, and a_data contains a pointer to an AvtData structure holding the measurement data. The measurement data can now be processed by the user.

In order to receive the next measurement a trigger should be sent back to the datapoint measurement.measurement_data. This trigger is used as an acknowledgement to inform the spectrometer that the host is ready to receive new data. After the trigger has been sent, the data on the location of the pointer a_data should no longer be used.

As soon as a new measurement is available, the callback will be re-invoked.

```
Example callback function

void measurement_data_calback(uint32_t device_id, AvtData *a_data, uint32_t
number_of_items)
{
      //Process a_data;
      your_function_processing_avt_data(a_data);
      //sent trigger to spectrometer so that next measurement can be sent
      AVT_Trigger(device_id, AVTMeasurementTags::CATEGORY,
             AVTMeasurementTags::MEASUREMENT_DATA::TAG, 0, NULL);
}

//set callback function for measurement data
AVT_SetCallback(m_first_device_id, AVTMeasurementTags::CATEGORY,
AVTMeasurementTags::MEASUREMENTDATA, &measurement_data_callback);
```

Paragraph 2.5.6.3 describes how to handle measurement data.

2.4.5.2 *Getting measurement data by polling*

The user can perform an AVT_Get on datapoint Measurement.measurementData at all times to get the data from the measurement. In case no measurement data is available, the get returns AVT_ERROR_IN_RETURN_DATA.

2.4.5.3 *Interpreting measurement data*

The callback function set by AVT_SetCallback is called with a pointer to an AvtData structure containing the measurement data. The contents of this data is a header followed by a set of datapoints.

(Example) Contents of measurement.measurementData.data

Message format version	uint8_t - 0			
Sequence number (unused)	uint8_t - 0			
Access type (trigger)	uint8_t - 3			
Number of CLTVs	uint8_t - 7 *			
Measurement.spectrum_data	uint16_t Cat: 3 *	uint16_t Tag: 28 *	uint16_t Status: ***	uint32_t Length: **
	Value: **			
Measurement.averages	uint16_t Cat: 3 **	uint16_t Tag: 6 **	uint16_t Status: ***	uint32_t Length: 4 **
	Value: uint32_t, actual number of averages			
Measurement.spectrum_data_timestamp	uint16_t Cat: 3 *	uint16_t Tag: 29 *	uint16_t Status: ***	uint32_t Length: 4 **
	Value uint32_t, measurement timestamp			
Digital_IO.digital_input_pins	uint16_t Cat: 6 *	uint16_t Tag: 0 *	uint16_t Status: ***	uint32_t Length: 4 **
	Value: uint32_t, Digital input status bitfield			
Digital_IO.detector_temperature	uint16_t Cat: 6 *	uint16_t Tag: 20 *	uint16_t Status: ***	uint32_t Length: 2 **
	Value: detector temperature in 1/100 degrees Kelvin			
Analog_IO.analog_in_pin_1	uint16_t Cat: 7 *	uint16_t Tag: 0 *	uint16_t Status: ***	uint32_t Length: 2 **
	Value: value of analog input port 1 in mV			
Digital_IO.board_temperature	uint16_t Cat: 6 *	uint16_t Tag: 19 *	uint16_t Status: ***	uint32_t Length: 2 **
	Value: board temperature in 1/100 degrees Kelvin			

*The number of ctlv' in measurement data, together with the order of the datapoints is subject to change in future firmware versions. Fields should always be interpreted.
** See chapter 7
*** Not used

There are 2 helper functions available to interpret the contents of Measurement.measurementData:
- AVT_ExtractTriggerData() is available in both the core and full library (See paragraph 2.1).
- AVT_ProcessMeasurementData() is only available in the full library (See paragraph 2.1).

Both functions can be used to convert the data in Measurement.measurementData to an array containing the individual datapoints.

Additionally AVT_ProcessMeasurementData() applies averaging, smoothing, dark pixel correction, dynamic dark correction, and straylight correction on the measurement data.

```
void measurement_data_calback(uint32_t device_id, AvtData *a_data, uint32_t
number_of_items)
```



```
{
      //Save a_data;
      std::uint32_t num_items;
      AvtData* items = nullptr;
      AVT_ProcessMeasurementData(device_id, a_data, &num_items,
            &items);
      //Items now pointers to array of AvtData items containing all
            measurement data

      //Process items
      your_function_processing_data_items(items);

      //sent trigger to spectrometer so that next measurement can be sent
      AVT_Trigger(device_id, AVTMeasurementTags::CATEGORY,
             AVTMeasurementTags::MEASUREMENT_DATA::TAG, 0, NULL);
}
```

See the provided demo apps for examples about how to do this for different programming languages. The C++ console demo shows examples for both full and core library.

2.5 Exported Functions

2.5.1 AVT_Get

Function:	**int AVT_Get**	
	(	
	uint32_t uint32_t uint32_t uint32_t uint8_t*	a_device_id, a_category, a_tag, a_length, a_value
	)	
Group:	Blocking data read function	
Description:	Receive datapoint data from spectrometer	
Parameters:	a_device_id: a_category: a_tag: a_length: a_value:	Device identifier returned by an AVT_Activate function Category of datapoint Tag of datapoint Length of datapoint in bytes Pointer to data of datapoint
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.2 AVT_GetMultiple

Function:	**AVT_STATUS AVT_GetMultiple**	
	(	
	uint32_t AvtData* uint32_t	a_device_id, a_avt_data_items, a_nr_of_items
	)	
Group:	Blocking data read function	
Description:	Receive datapoints from library	
Parameters:	a_device_id:	Device identifier

	a_avt_data_items: a_nr_of_items:	Datapoint data Number of datapoints
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.3 AVT_Set

Function:	**int AVT_Set**	
	(	
	uint32_t uint32_t uint32_t uint32_t uint8*	a_device_id, a_category, a_tag, a_length, a_value
	)	
Group:	Blocking data write function	
Description:	Send datapoint data to spectrometer	
Parameters:	a_device_id: a_category: a_tag: a_length: a_value:	Device identifier returned by an AVT_Activate function Category of datapoint Tag of datapoint Length of datapoint in bytes Pointer to data of datapoint
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.4 AVT_SetMultiple

Function:	**AVT_STATUS AVT_SetMultiple**	
	(	
	uint32_t AvtData* uint32_t	a_device_id, a_avt_data_items, a_nr_of_items
	)	
Group:	Blocking data write function	
Description:	Send datapoint data to spectrometer	
Parameters:	a_device_id: a_avt_data_items: a_nr_of_items:	Device identifier Datapoint data Number of datapoints
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.5 AVT_Activate

Function:	**AVT_STATUS AVT_Activate**	
	(	
	uint32_t	a_device_id
	)	
Group:	Blocking control function	
Description:	Activates selected spectrometer for communication.	
Parameters:	a_device_id	Device identifier as specified by Avt_GetList
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.6 AVT_Deactivate

Function:	AVT_STATUS AVT_Deactivate	
	(	
	uint32_t	a_device_id
	)	
Group:	Blocking control function	
Description:	Closes communication with selected spectrometer.	
Parameters:	a_device_id:	Device identifier as specified by Avt_GetList
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.7 AVT_Init

Function:	AVT_STATUS AVT_Init	
	(	
	Void	
	)	
Group:	Blocking control function	
Description:	Initializes the communication interface with the spectrometers and the internal data structures.	
Parameters:	None:	
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.8 AVT_Done

Function:	AVT_STATUS AVT_Done	
	(	
	Void	
	)	
Group:	Blocking control function	
Description:	Closes the communication and releases internal storage.	
Parameters:	None	
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.9 AVT_GetNrOfDevices

Function:	uint32_t AVT_GetNrOfDevices	
	(	
	void	
	)	
Group:	Blocking control function	
Description:	Returns the number of devices attached.	
Parameters:	None	
Return:	> 0: Number of Avatech spectrometers connected to the system 0: No devices found	

2.5.10 AVT_GetList

Function:	**AVT_STATUS AVT_GetList**	
	(	
	uint32_t AvtDeviceInfo*	number_of_items, device_info_list
	)	
Group:	Blocking control function	
Description:	Returns device information for each spectrometer connected to the ports indicated at AVT_Init().	
Parameters:	number_of_items: AvtDeviceInfo:	Number of devices to store in the device list Pointer to allocated buffer to store device information. The resulting device list will be sorted on serial number.
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.11 AVT_SetLogging

Function:	**AVT_STATUS AVT_SetLogging**	
	(	
	AVT_LOG_LEVEL	a_log_level
	)	
Group:	Blocking control function.	
Description:	Enables or disables writing debug information to a log file, allows levels of detail	

Parameters:	a_log_level:	Set to a desired level
		
		typedef enum AVT_AVT_LOG_LEVEL_
		{
		    AVT_LOG_LEVEL_OFF,
		    AVT_LOG_LEVEL_ERROR,
		    AVT_LOG_LEVEL_WARNING,
		    AVT_LOG_LEVEL_INFO,
		    AVT_LOG_LEVEL_DEBUG,
		    AVT_LOG_LEVEL_ALL
		} AVT_LOG_LEVEL;

Return:	AVT_STATUS	AVT_SUCCESS, or error value
Remark:	When logging is enabled, debug information will be written to a file called “avatech.log”. The log file is located in the user folder (c:\Users\[username] for windows, home/<username> for Linux and /Users/<username> for Mac).	

2.5.12 AVT_GetVersion

Function:	**AVT_STATUS AVT_GetVersion**	
	(	
	char*	a_version
	)	
Group:	Blocking control function.	
Description:	Returns version info consisting of major, minor and build version numbers	
Parameters:	a_version:	Array of char, define buffer size of ?? chars
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.13 AVT_GetBuildInfo

Function:	**AVT_STATUS AVT_GetBuildInfo**	
	(	
	char*	a_build_info
	)	
Group:	Blocking control function.	
Description:	Returns extended build info, including branch and hash info	
Parameters:	a_build_info:	Arrray of char, define buffer size of ?? chars
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.14 AVT_Trigger

Function:	**AVT_STATUS AVT_Trigger**	
	(	
	uint32_t uint32_t uint32_t uint32_t void*	a_device_id, a_category, a_tag, a_length, a_optional_value
	)	
Group:	Blocking control function	
Description:	Trigger a datapoint on the spectrometer	
Parameters:	a_device_id: a_category: a_tag: a_length: a_optional_value:	
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.15 AVT_SetCallback

Function:	**AVT_STATUS AVT_SetCallback**	
	(	
	uint32_t uint32_t uint32_t callback_function	a_device_id, a_category, a_tag, a_callback
	)	
Group:	Blocking control function	
Description:	Register a callback function that should be invoked when a datapoint is triggered by the spectrometer.	
Parameters:	a_device_id: a_category: a_tag: a_callback:	
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.16 AVT_SetDeviceMonitorCallback

Function:	**AVT_STATUS AVT_SetDeviceMonitorCallback**	
	(	
	callback_function	a_callback
	)	
Group:	Blocking control function	
Description:	Sets callback function to be called when device connection status changes	
Parameters:	a_callback:	
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.17 AVT_SetSmoothing

Function:	**AVT_STATUS AVT_SetSmoothing**	
	(	
	uint32_t	a_device_id
	uint32_t	a_num_smoothing
	)	
Group:	Blocking control function	
Description:	Sets number of smoothing pixels, used on both left and right side of each pixel	
Parameters:	a_device_id	Device identifier as specified by AVT_GetList
	a_num_smoothing	Number of smoothing pixels
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.18 AVT_EnableDynamicDarkCorrection

Function:	**AVT_STATUS AVT_EnableDynamicDarkDetection**	
	(	
	uint32_t	a_device_id
	Bool	a_enable
	)	
Group:	Blocking control function	
Description:	Enables dynamic dark correction	
Parameters:	a_device_id	Device identifier as specified by AVT_GetList
	a_enable	Set to true when dynamic dark correction should be enabled, set to false otherwise
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.19 AVT_EnableStrayLightCorrection

Function:	**AVT_STATUS AVT_EnableStrayLightCorrection**	
	(	
	uint32_t	a_device_id
	Bool	a_enable
	Float	a_multiplication_factor
	)	
Group:	Blocking control function	
Description:	Enables stray light correction	
Parameters:	a_device_id	Device identifier as specified by AVT_GetList
	a_enable	Set to true when stray light should be corrected, set to false otherwise
	a_multiplication_factor	Factor used in correction algorithm
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.20 AVT_GetErrorMessage

Function:	**AVT_STATUS AVT_GetErrorMessage**	
	(	
	uint32_t	a_error_code
	char*	a_error_message
	uint32_t	a_buffer_size
	)	
Group:	Blocking control function	
Description:	Translates numeric error message to legible string, user needs to supply a buffer to write characters to and the size of this buffer.	
Parameters:	a_error_code	Error code returned by AVT library function
	a_error_message	Legible error code, returned by this function
	a_buffer_size	Max length of error message that is returned
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.21 AVT_ProcessMeasurementData

Function:	**AVT_STATUS AVT_ProcessMeasurementData**	
	(	
	uint32_t	a_device_id
	AvtData*	a_data_item
	uint32_t*	a_results_num_items
	AvtData**	a_measurement_results
	)	
Group:	Blocking control function	
Description:	This utility function applies averaging, smoothing, dark pixel correction, dynamic dark correction and straylight correction on the measurement data and will make the data available in the a_measurement_results parameter	
Parameters:	a_device_id	Device identifier as specified by AVT_GetList
	a_data_item	Parameter from callback function
	a_results_num_items	Number of measurement data blocks that are available (output)
	a_measurement_results	Pointer to pointer to measurement data (output)
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.5.22 AVT_ExtractTriggerData

Function:	**AVT_STATUS AVT_ExtractTriggerData**	
	(	
	uint32_t	a_device_id
	AvtData*	a_data_item
	uint32_t*	a_num_items
	AvtData**	a_extracted_items
	)	
Group:	Blocking control function	
Description:	Utility function that will make trigger data available in the a_extracted_items parameter	
Parameters:	a_device_id	Device identifier as specified by AVT_GetList
	a_data_item	Pointer to trigger data
	a_num_items	The number of datapoints in the trigger data (output)
	a_extracted_items	Pointer to array containing AvtData Structure (output)
Return:	AVT_STATUS	AVT_SUCCESS, or error value

2.6 Data Elements

Type	Format	Value/Range	Description
AVT_STATUS	See 2.8		

AvtData	struct
	{
	uint32_t category;
	uint32_t tag;
	uint32_t length;
	uint8_t* data;
	AVT_STATUS status;
	}
			
			Category of the datapoint to access
			Tag of the datapoint to access
			Length of the datapoint value
			Value of the datapoint
			Status of the request. Contains errors if setting/getting failed.

AvtDeviceInfo	struct
	{
	uint32_t id; //device_id
	char serial_number[AVT_SERIAL_LEN];
	AVT_DEVICE_STATUS status;
	}
			
			AVT_SERIAL_LEN = 15
			AVT_DEVICE_STATUS is a value of 4 bytes:
			
			typedef enum AVT_DEVICE_STATUS_
			{
			AVAILABLE,
			ACTIVATED,
			DISCONNECTED,
			ACTIVATED_DISCONNECTED,
			} AVT_DEVICE_STATUS;

2.7 Return Value Constants

The following table gives an overview of possible integer return codes:

Return code	Value	Description
AVT_SUCCESS	0x00	AvaTech protocol success, successfully executed function
AVT_ERROR	0x01	AvaTech protocol error, unable to successfully execute function
AVT_DEVICE_NOT_FOUND	0x02	Spectrometer device not found
AVT_NO_CONNECTION	0x03	Internal error: device has no connection
AVT_NOT_CONNECTED	0x04	Internal error: device is not connected
AVT_WRITE_FAILED	0x05	Internal error: write data failed
AVT_RESPONSE_TIMEOUT	0x06	Timeout waiting for response from spectrometer
AVT_ERROR_IN_RETURN_DATA	0x07	The spectrometer returned one or more datapoints with an error
AVT_DATA_COUNT_MISMATCH	0x08	Mismatch in data count: the number of returned datapoints doesn't match the number of requested datapoints
AVT_NO_RETURN_DATA	0x09	No datapoints returned from spectrometer
AVT_INVALID_DATA_POINTER	0x0A	The data pointer provided to the function is a null pointer
AVT_INVALID_ARGUMENT_POINTER	0x0B	The pointer provided to a function argument is a null pointer
AVT_INVALID_NUMBER_DATAITEMS	0x0C	Number of items in target list is invalid (i.e. is 0 or too small to hold all items)
AVT_INCOMPATIBLE_FIRMWARE_VERSION	0x0D	Library requires newer firmware version
AVT_INVALID_DATAPOINT	0x1001	Datapoint does not exist
AVT_INVALID_DATA_LENGTH	0x1002	Destination data size is too small
AVT_DATAPOINT_READONLY	0x1003	The datapoint is readonly
AVT_INVALID_NUMBER_OF_CTLVS	0x1004	The number of CTLVs is more than the spectrometer can handle
AVT_INVALID_VALUE_TOO_LOW	0x1100	Value set is lower than datapoint accepts
AVT_INVALID_VALUE_TOO_HIGH	0x1101	Value set is higher than datapoint accepts
AVT_INVALID_START_STOP_PIXEL	0x1200	Start pixel is larger than stop pixel
AVT_INVALID_TOTAL_INTEGRATIONTIME	0x1201	Number of averages * integration time is too large
AVT_INVALID_TRIGGER_SINGLESCAN_LEVEL	0x1202	Level triggering for single scan trigger mode is not supported
AVT_INVALID_STROBES_VS_INTEGRATIONTIME	0x1203	Number of strobe pulses does not fit current integration time
AVT_INVALID_SLC_RANGE	0x1204	The range used for straylight correction calculations (for calculating integral value) is outside the measurement data range
AVT_INVALID_LEVEL_TRIGGER_SOFTWARE_TRIGGER	0x1205	Level trigger mode is only possible with a hardware trigger
AVT_MEASUREMENT_ALREADY_RUNNING	0x1206	Start measurement was attempted while another measurement was already active

ID: 036252

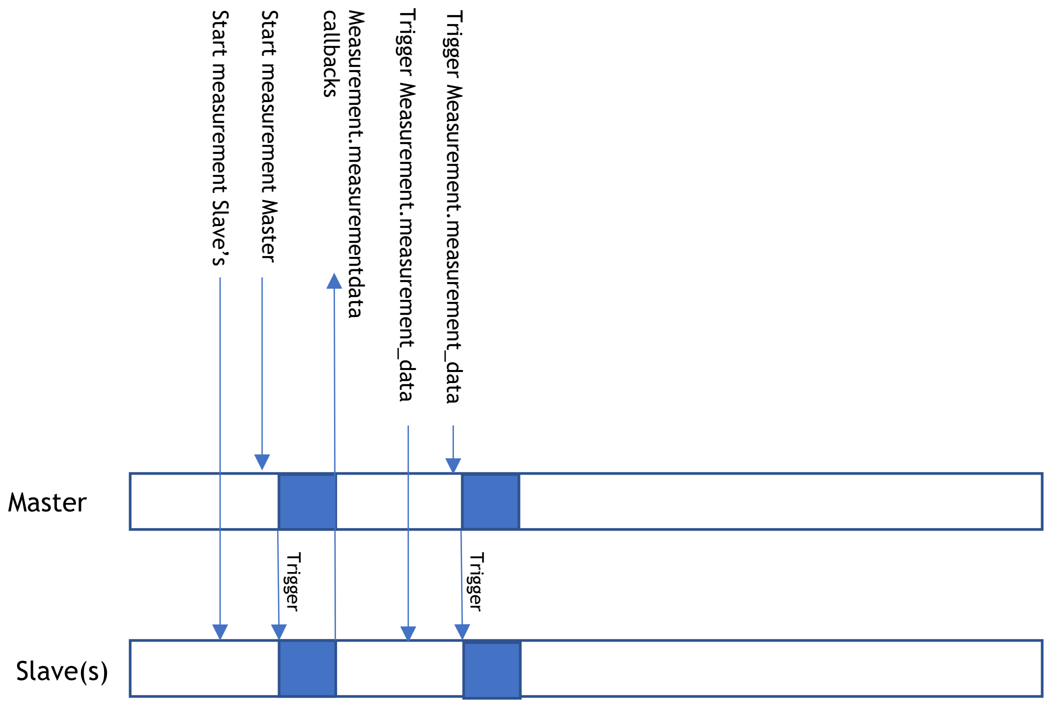
3 Synchronizing Spectrometers

The spectrometer can be connected with a sync cable. In synchronous mode, one spectrometer is configured as the “Master” and will trigger slave spectrometer(s) when a measurement starts. Master and slave spectrometers should be correctly configured by using the following datapoints

	Master	Slave
Measurement.trigger_mode	Single Scan (1)	Single scan (1)
Measurement.trigger_source	Software (0) or External (1)	Sync
Measurement.sync_master_enabled	True	False

When using multiple spectrometers, all settings that need to be performed for measuring (like start/stop pixel, integration time) should be set for each participating spectrometer.

The measurement callback should be registered per spectrometer, although, the same function can most likely by reused for each spectrometer. However, although the callback function should sent the trigger to Measurement.measurement_data to all spectrometers, it should sent the trigger to the master spectrometer as the last trigger.



4 Synchronizing with other equipment

4.1 Introduction

A spectrometer can be triggered externally, but also generate pulses to trigger other equipment during a measurement.

4.2 External triggering

The user should always trigger datapoint Measurement.start before the spectrometer will start a measurement. After triggering measurement.start a number of options are available to synchronize the measurement with external equipment.

A trigger source can be selected to postpone a measurement This can be enabled via datapoint Measurement.triggerSource

	Value	
Software	0	Measurement will start immediately on triggering measurement.start
External Trigger	1	Measurements will start after an rising edge has been detected on the external trigger pin.
Sync Trigger	2	Measurements will start after an rising edge has been detected on the sync pin, used for synchronization of spectrometers. See chapter 3.

The trigger mode can be set via datapoint Measurement.triggerMode

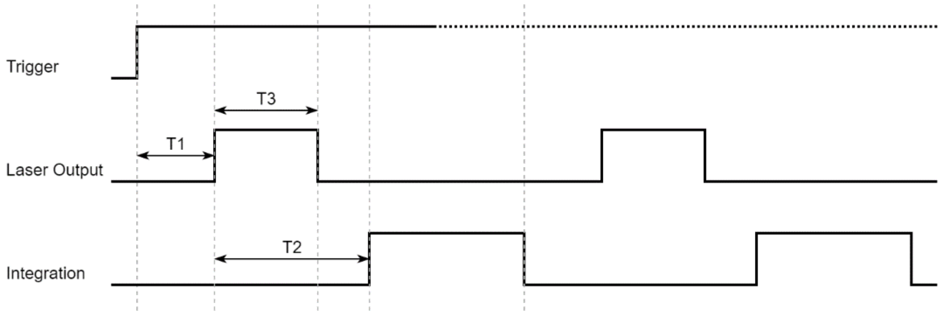
	Value	
Multi scan trigger	0	The spectrometer will start measuring after the first trigger/pulse.
Single scan Trigger	1	The spectrometer will perform one measurement after receiving a trigger/pulse. Afterwards it will wait for the next trigger/pulse.
Level Scan Trigger	2	The spectrometer will measure as long as the pulse is high. This function only works when Measurement.triggerSource is set to External Trigger.

4.3 Strobe pulse control

In order to synchronize the spectrometer with a pulsed light source like the AvaLight-XE, the spectrometer can generate strobe pulses during the integration time cycle. The datapoint measurement.strobeControl determines the number of pulses the spectrometer sends out on the strobe control pin during one integration time cycle. Strobe pulse duration is 1ms and strobe pulses are evenly distributed over the integration time minus a small delay at both the begin and and at the integration time. These delays are spectrometer dependent and can be found in datapoint measurement.strobe_delays.

4.4 Laser pulse control

In order to fire a laser, the spectrometer can generate laser pulses during the integration time cycle. The datapoint measurement.laserWidth determines the pulsewidth (T3) that is generated by the spectrometer. After the first (hardware or software) trigger, the laser output of the spectrometer will be driven continuously with every integration cycle in case multiple measurements are requested. The integration delay T2 (datapoint measurement.integration_delay) can be used to postpone the measurement until after the laser pulse.



The unit for T1, T2 and T3 is FPGA clock cycles. The FPGA clock runs at 48 MHz, so delays and pulse width can be set with 20.83 nanoseconds steps. If the integration delay T2 is set to 0 FPGA cycles, the rising edge of the integration signal will start one clock cycle (20.83ns) before the rising edge of the laser pulse. This will ensure that with this setting, the flash of the source that is triggered by the laser pulse entirely falls in the integration time cycle.

5 Buffering measurement data

When performing measurements with a short integration time, it can happen that the spectrometer is not able to send the measurement to the PC before the next measurement is available. The spectrometer has three options.

- Standard - Always send the last available measurement to the PC (default)
- Store to RAM - Buffer the measurements in a fifo-queue, sent the measurements after the last measurement has been performed.
- Dynamic store to RAM - Buffer the measurements in a fifo queue, sent the measurements while measuring.

The requested mode can be set via datapoint measurement.measurement_type.

6 Measurement data processing

6.1 Dynamic Dark correction

The pixels of the CCD detector are thermally sensitive, which causes a small current that is measured by the ADC as light, even without exposure to light. To get an approximation of this dark current, the signal of some optical black pixels of the detector can be taken and subtracted from the raw scope data. At these optical black pixels, the intensity and thermal behavior is the same as the active data pixels, if no light falls on the detector. Enabling dynamic dark correction will therefore result in a baseline fluctuating round zero, and measurement data will be less sensitive for temperature changes than with dynamic dark correction off. If the spectrometer does not include blackened datapixels, nor dedicated optical black pixels, enabling the dynamic dark correction results in an error message. Dynamic Dark correction can be enabled with AVT_EnableDynamicDarkCorrection().

6.2 Correct for defective pixels

Defective pixels can be used to store the pixel numbers that should be eliminated from the data transfer. The AvaTech library will calculate the data for a defective pixel by interpolating the data of the neighbor pixels. A defective pixel can be specified in the range from 0 to the value of datapoint detector.number_of_pixels. The AvaTech Library evaluates the pixels in the datapoint in an increasing order until a pixel is specified which is equal or larger than the number of pixels in the detector. The datapoint detector.defective_pixels can be used to generate a list of defective pixels.

6.3 Smoothing

The smoothing function averages the spectrum over a number of pixels on the detector array. For example, if datapoint Measurement.smoothPixels is set to 2, the spectral data for all pixels xn on the detector array will be averaged with their neighbor pixels xn-2, xn-1, xn+1 and xn+2.
The optimal smoothpix parameter depends on the distance between the pixels at the detector array and the width of the light beam that enters the spectrometer. If resolution is not an important issue, a higher smoothing parameter can be set to decrease noise at the price of less resolution.

6.4 Digital IO

The spectrometer has a number of programmable bidirectional input/output pins available. You must specify the direction of these ports by setting the Digital_IO.digital_IO_directions data point. This takes a 32-bit value, The first bit sets the direction of pin 1, the second bit sets the direction of bit 2, etcetera. 0 means input, 1 means output. E.g., setting the first five pins to output is done by using a value of 0x0000001F.

You then read digital inputs using the Digital_IO.digital_input_pins data point, also with a bitmapped 32-bit value.

You can set digital outputs using the Digital_IO.digital_output_pins data point, again with a bitmapped 32-bit value.

You can use PWM with all digital outputs, using e.g. for pin 1 the Digital_IO.Pwm_Pin_1_duty_cycle and Digital_IO.Pin_1_frequency data points. The duty cycle is an 8-bit value, coding a percentage (0 – 100), the frequency is a 32-bit value, coding a value of 500 – 300000 Hz.

Please note that pins 1 (trigger input), 2 (synchronization) and 5 (laser/strobe out) have double functions. Pins can only be used for one function.

6.5 Analog IO

The AS7007 spectrometer has 1 or more analog input and output pins (pin 8 and 9).

You set the first the analog output using the Analog_IO.Analog_out_pin_1 datapoint, using a 16-bit value that encodes millivolts.
You can read the first analog input using the Analog_IO.Analog_in_pin_1 datapoint, using a 16-bit value that encodes millivolts.

7 Datapoints

7.1 Category 0: Generic

7.1.1 Datapoint 1: serial_number

Datapoint	Generic - serial_number	cat	tag
		0	1
Access	Read-only		
Type	char[9]		
Default value	{'8', '6', '7', '4', '5', '2', '3', '9', '1'}		
Description	Serial number of device		

7.1.2 Datapoint 2: user_friendly_name

Datapoint	Generic - user_friendly_name	cat	tag
		0	2
Access	Read/Write	Persistent Value	
Type	char[64]		
Default value	{'8', '6', '7', '4', '5', '2', '3', '9', '1'}		
Description	User settable name for spectrometer. A null terminated string.		

7.1.3 Datapoint 3: reset_reason

Datapoint	Generic - reset_reason	cat	tag
		0	3
Access	Read-only		
Type	std::uint8_t		
Description	Reset Reason: the last known reset reason of the Spectrometer; bit 0: Power on reset, bit 1: watchdog reset, bit 2 soft_reset, bit 3 debug_reset, bit4: hardware reset		

7.1.4 Datapoint 4: reset_device

Datapoint	Generic - reset_device	cat	tag
		0	4
Access	Trigger		
Type	trigger		
Description	Request to reset the device		

7.1.5 Datapoint 5: reset_device_config

Datapoint	Generic - reset_device_config	cat	tag
		0	5
Access	Trigger		
Type	trigger		
Description	Request to reset the Device Configuration parameters to their factory settings. This command will result in the loss of all user-specific device configuration settings		

7.1.6 Datapoint 12: device_type

Datapoint	Generic - device_type	cat	tag
		0	12
Access	Read-only		
Type	std::uint32_t		
Description	Integer indicating the device type. Value is set to 4 for AS7007		

7.1.7 Datapoint 14: max_number_of_ctlvs

Datapoint	Generic - max_number_of_ctlvs	cat	tag
		0	14
Access	Read-only		
Type	std::uint8_t		
Description	Contains the maximum number of ctlvs accepted by firmware. When a message is received with an amount of ctlvs that exceeds this number, return error		

7.1.8 Datapoint 15: serial_speed

Datapoint	Generic - serial_speed	cat	tag
		0	15
Access	Read/Write		
Type	std::uint32_t		
Description	Speed in baudrate for RS232 communication. Only for RS232-spectrometers		

7.2 Category 1: Detector

7.2.1 Datapoint 0: detector_type

Datapoint	Detector - detector_type	cat	tag
		1	0
Access	Read-only		
Type	std::uint8_t		
Description	Detector identification byte		

7.2.2 Datapoint 1: number_of_pixels

Datapoint	Detector - number_of_pixels	cat	tag
		1	1
Access	Read-only		
Type	std::uint16_t		
Description	Number of Pixels available on detector		

7.2.3 Datapoint 2: video_ADC_gain_channel_1

Datapoint	Detector - video_ADC_gain_channel_1	cat	tag
		1	2
Access	Read/Write	Persistent Value	
Type	float	Min	Max

		1.0	6.0
Default value	1.0		
Description	Video ADC AD9826 gain setting for channel1. Gain correction for spectrometer ADC (range 1.0V - 6.0V, divided in 64 steps)		

7.2.4 Datapoint 3: video_ADC_gain_channel_2

Datapoint	Detector - video_ADC_gain_channel_2	**cat**	**tag**
		1	3
Access	Read/Write	Persistent Value	
Type	float	**Min**	**Max**
		1.0	6.0
Default value	1.0		
Description	Video ADC AD9826 gain setting for channel 2. Gain correction for spectrometer ADC (range 1.0V - 6.0V, divided in 64 steps)		

7.2.5 Datapoint 4: video_ADC_offset_channel_1

Datapoint	Detector - video_ADC_offset_channel_1	**cat**	**tag**
		1	4
Access	Read/Write	Persistent Value	
Type	float	**Min**	**Max**
		-0.3	0.3
Default value	0.0		
Description	Video ADC AD9826 offset setting for channel 1 Offset correction for spectrometer ADC in Volt (range is -0.30V .. 0.30V divided in 512 steps)		

7.2.6 Datapoint 5: video_ADC_offset_channel_2

Datapoint	Detector - video_ADC_offset_channel_2	**cat**	**tag**
		1	5
Access	Read/Write	Persistent Value	
Type	float	**Min**	**Max**
		-0.3	0.3
Default value	0.0		
Description	Video ADC AD9826 offset setting for channel 2 Offset correction for spectrometer ADC in Volt (range is -0.30V .. 0.30V divided in 512 steps)		

7.2.7 Datapoint 6: external_video_offset

Datapoint	Detector - external_video_offset	**cat**	**tag**
		1	6
Access	Read/Write	Persistent Value	
Type	float	**Min**	**Max**
		0	2
Default value	0		
Description	External offset parameter is used to be able to match the detector output range with the ADC range		

7.2.8 Datapoint 7: defective_pixels

Datapoint	Detector - defective_pixels	cat	tag
		1	7
Access	Read/Write	Persistent Value	
Type	std::uint16_t[30]		
Default value	{4095, 4095}		
Description	A maximum of 30 defective pixels can be used to store the pixel numbers that should be eliminated from the data transfer. See chapter 6.2 Correct for defective pixels		

7.2.9 Datapoint 8: ADC_type

Datapoint	Detector - ADC_type	cat	tag
		1	8
Access	Read-only		
Type	std::uint8_t		
Description	indicates whether single ended (0) or differential ADC (1) is enabled		

7.2.10 Datapoint 9: DMA_error_status

Datapoint	Detector - DMA_error_status	cat	tag
		1	9
Access	Read-only		
Type	bool		
Description	indicates the error status of the DMA, is true when there is an error		

7.2.11 Datapoint 10: min_integration_time

Datapoint	Detector - min_integration_time	cat	tag
		1	10
Access	Read-only		
Type	std::uint32_t		
Description	minimum integration time, for verifying the requested integration time.		

7.2.12 Datapoint 11: max_integration_time

Datapoint	Detector - max_integration_time	cat	tag
		1	11
Access	Read-only		
Type	std::uint32_t		
Description	maximum integration time, for verifying the requested integration time.		

7.2.13 Datapoint 12: strobe_delays

Datapoint	Detector - strobe_delays	cat	tag
		1	12
Access	Read-only		
Type	std::uint16_t[2]		
Description	contains the strobe begin delay and the strobe end delay, used to calculate the amount of strobe pulses per scan		

7.2.14 Datapoint 13: detector_dark_pixels

Datapoint	Detector - detector_dark_pixels	cat	tag
		1	13
Access	Read-only		
Type	detector_dark_pixel_range[32]		
Description	contains the dark pixel information for the detector		

7.3 Category 3: Measurement

7.3.1 Datapoint 0: start

Datapoint	Measurement - start	cat	tag
		3	0
Access	Trigger		
Type	trigger		
Description	Triggering this datapoint results in a start of measurement.		

7.3.2 Datapoint 1: stop

Datapoint	Measurement - stop	cat	tag
		3	1
Access	Trigger		
Type	trigger		
Description	Triggering this datapoint results in stopping the active measurement		

7.3.3 Datapoint 2: start_pixel

Datapoint	Measurement - start_pixel	cat	tag
		3	2
Access	Read/Write		
Type	std::uint16_t		
Description	The start- and stoppixel are the first and last pixel to be sent to the host. The full range for a spectrometer is between 0 and the value specified in detector.number_of_pixels minus 1. Reducing the range of pixels helps to increase the data transfer speed and allows you to transfer only the data that is relevant to the application.		

7.3.4 Datapoint 3: stop_pixel

Datapoint	Measurement - stop_pixel	cat	tag
		3	3
Access	Read/Write		
Type	std::uint16_t		
Description	The start- and stoppixel are the first and last pixel to be sent to the host. The full range for a spectrometer is between 0 and the value specified in detector.number_of_pixels minus 1. Reducing the range of pixels helps to increase the data transfer speed and allows you to transfer only the data that is relevant to the application.		

7.3.5 Datapoint 4: integration_time

Datapoint	Measurement - integration_time	cat	tag
		3	4
Access	Read/Write		
Type	std::uint32_t		
Default value	1000		
Description	Integration time in microseconds. The integration time is the exposure time during one scan. The longer the integration time, the more light is exposed to the detector during a single scan, and therefore the higher the signal. The minimum and maximum integration time can be found in datapoints detector.min_integration time and detector.max_integration_time		

7.3.6 Datapoint 5: integration_delay

Datapoint	Measurement - integration_delay	cat	tag
		3	5
Access	Read/Write		
Type	std::uint32_t		
Default value	0		
Description	The integration delay parameter can be used to start the integration time not immediately after the measurement request (or on an external hardware trigger), but after a specified delay. The unit for this delay is FPGA clock cycles. The FPGA clock runs at 48 MHz, so the integration delay can be set with 20.83 nanoseconds steps.		

7.3.7 Datapoint 6: averages

Datapoint	Measurement - averages	cat	tag
		3	6
Access	Read/Write		
Type	std::uint32_t		
Default value	1		
Description	Number of averaged spectra for a single measurement		

7.3.8 Datapoint 9: smoothing

Datapoint	Measurement - smoothing	cat	tag
		3	9
Access	Read/Write		
Type	std::uint16_t		
Default value	0		
Description	Number of pixels to the left and to the right of a pixel that are used for smoothing (averaging) the spectrum in the wavelength dimension. Range is 0 - detector.number_of_pixels / 2. See chapter 6 Measurement data processing		

7.3.9 Datapoint 11: saturation_detection

Datapoint	Measurement - saturation_detection	cat	tag
		3	11
Access	Read/Write		
Type	std::uint8_t	Min	Max
		0	2
Default value	0		

Description	Enables (1) or disables (0) saturation detection.

7.3.10 Datapoint 12: trigger_mode

Datapoint	Measurement - trigger_mode	**cat**	**tag**
		3	12
Access	Read/Write		
Type	std::uint8_t	**Min**	**Max**
		0	2
Default value	0		
Description	trigger mode; 0: Multi scan trigger, 1: Single scan trigger, 2: Level scan trigger		

7.3.11 Datapoint 13: trigger_source

Datapoint	Measurement - trigger_source	**cat**	**tag**
		3	13
Access	Read/Write		
Type	std::uint8_t	**Min**	**Max**
		0	2
Default value	0		
Description	Trigger source; 0: Software, 1: External trigger, 2: Sync input		

7.3.12 Datapoint 15: strobe_control

Datapoint	Measurement - strobe_control	**cat**	**tag**
		3	15
Access	Read/Write		
Type	std::uint16_t		
Default value	0		
Description	Number of strobe pulses during integration period (high time of pulse is 1 ms);		

7.3.13 Datapoint 16: laser_delay

Datapoint	Measurement - laser_delay	**cat**	**tag**
		3	16
Access	Read/Write		
Type	std::uint32_t		
Default value	0		
Description	Laser delay since trigger, unit is internal FPGA clock cycle, 0 = one unit before laser start, range 0 - 0xFFFFFFFF		

7.3.14 Datapoint 17: laser_width

Datapoint	Measurement - laser_width	**cat**	**tag**
		3	17
Access	Read/Write		
Type	std::uint32_t	**Min**	**Max**
		0	65535
Default value	0		
Description	Laser pulse width		

7.3.15 Datapoint 19: str_status

Datapoint	Measurement - str_status	**cat** 3	**tag** 19
Access	Trigger		
Type	trigger		
Description	Dependent of type of measurement. In case of store to ram measurement, the contents is AVT_SUCCESS if store to ram measurement is finished. In case of dynamic store to ram, the contents is a combination of Measurement.scan_buffer_overflow_status, Measurement.measurement_fifo_size and Measurement.number_of_measurements_available		

7.3.16 Datapoint 20: measurement_fifo_size

Datapoint	Measurement - measurement_fifo_size	**cat** 3	**tag** 20
Access	Read-only		
Type	std::uint32_t		
Description	Measurement data fifo size, indicates the number of measurements that can be stored in ram		

7.3.17 Datapoint 21: measurement_type

Datapoint	Measurement - measurement_type	**cat** 3	**tag** 21
Access	Read/Write		
Type	std::uint8_t		
Default value	0		
Description	Type of measurement, 0 = standard measurement, 1 = store to ram, 2 = dynamic store to ram		

7.3.18 Datapoint 22: wavelength_polynomials

Datapoint	Measurement - wavelength_polynomials	**cat** 3	**tag** 22
Access	Read/Write	Persistent Value	
Type	float[5]		
Default value	{0.0f, 1.0f, 0.0f, 0.0f, 0.0f}		
Description	Polynomial coefficients needed to determine center wavelength per pixel		

7.3.19 Datapoint 23: nonlinearity_enable

Datapoint	Measurement - nonlinearity_enable	**cat** 3	**tag** 23
Access	Read/Write	Persistent Value	
Type	std::uint8_t	**Min** 0	**Max** 1
Default value	0		
Description	Can be used to indicate if user wants to apply nonlinearity correction for the spectrometer. Not used by spectrometer or library.		

7.3.20 Datapoint 24: nonlinearity_correction

Datapoint	Measurement - nonlinearity_correction	**cat**	**tag**
		3	24
Access	Read/Write	Persistent Value	
Type	nonlin_correction		
Default value	{{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}, 0.0, 0.0}		
Description	polynomials, low- and high count values		

7.3.21 Datapoint 25: number_of_measurements_available

Datapoint	Measurement - number_of_measurements_available	**cat**	**tag**
		3	25
Access	Read-only		
Type	std::uint32_t		
Description	Number of store to ram measurements available		

7.3.22 Datapoint 26: measurement_in_progress

Datapoint	Measurement - measurement_in_progress	**cat**	**tag**
		3	26
Access	Read-only		
Type	std::uint32_t		
Description	Measurement in progress		

7.3.23 Datapoint 27: number_of_measurements_requested

Datapoint	Measurement - number_of_measurements_requested	**cat**	**tag**
		3	27
Access	Read/Write		
Type	std::uint32_t		
Description	Number of requested measurements, to be set prior to a measurement start, 0 for continuous measurements		

7.3.24 Datapoint 28: spectrum_data

Datapoint	Measurement - spectrum_data	**cat**	**tag**
		3	28
Access	Read-only		
Type	trigger		
Description	contains an array of measurement data. The size depends on the number of pixels requested. The type is uint16_t if measurement.averages = 1. Otherwise the type is uint32_t.		

7.3.25 Datapoint 29: spectrum_data_timestamp

Datapoint	Measurement - spectrum_data_timestamp	**cat**	**tag**
		3	29
Access	Read-only		
Type	std::uint32_t		

Description	Last measurements time stamp [10us] as provided by the FPGA, time stamp is relative to power-on moment

7.3.26 Datapoint 30: sync_master_enabled

Datapoint	Measurement - sync_master_enabled	**cat** 3	**tag** 30
Access	Read/Write		
Type	std::uint8_t	**Min** 0	**Max** 1
Description	Enable or disable generation of sync pulses on the synchronization output of the Spectrometer		

7.3.27 Datapoint 31: simulated_measurement_enabled

Datapoint	Measurement - simulated_measurement_enabled	**cat** 3	**tag** 31
Access	Read/Write		
Type	std::uint8_t	**Min** 0	**Max** 1
Description	For testing purposes, the spectrometer can simulate a measurement.		

7.3.28 Datapoint 32: scan_buffer_overflow_status

Datapoint	Measurement - scan_buffer_overflow_status	**cat** 3	**tag** 32
Access	Read-only		
Type	bool		
Description	Scan buffer overflow status, returns true when one or more scans are missed due to internal scan buffer overflow		

7.3.29 Datapoint 33: spurious_trigger_idle_error_status

Datapoint	Measurement - spurious_trigger_idle_error_status	**cat** 3	**tag** 33
Access	Read-only		
Type	bool		
Description	returns true if external trigger detected when no measurement sequence was started or measurement was running		

7.3.30 Datapoint 34: spurious_trigger_overflow_error_status

Datapoint	Measurement - spurious_trigger_overflow_error_status	**cat** 3	**tag** 34
Access	Read-only		
Type	bool		
Description	returns true if external trigger was detected during a running measurement. This type of error is relevant only with trigger type Single Scan Trigger (SST)		

7.3.31 Datapoint 35: measurement_data

Datapoint	Measurement - measurement_data	cat	tag
		3	35
Access	Read/Write		
Type	trigger		
Description	measurement data is the combination of spectrumdata, timestamp, number of averages, temperatures and digital- and analog io datapoints		

7.3.32 Datapoint 36: measurement_data_max_size

Datapoint	Measurement - measurement_data_max_size	cat	tag
		3	36
Access	Read-only		
Type	std::uint32_t		
Description	Indicates the maximum size that the measurement data can take up in bytes, required when calling a GET on the measurement_data datapoint		

7.3.33 Datapoint 37: pixel_wavelength

Datapoint	Measurement - pixel_wavelength	cat	tag
		3	37
Access	Read-only		
Type	float[4096]		
Description	Wavelength for each pixel. Derived from Measurement.wavelength_polynomials at Spectrometer startup		

7.4 Category 4: Calibration

7.4.1 Datapoint 0: spectrum_correction

Datapoint	Calibration - spectrum_correction	cat	tag
		4	0
Access	Read/Write	Persistent Value	
Type	float[4096]		
Default value	{}		
Description	Value to be subtracted from each pixel, for Straylight correction		

7.4.2 Datapoint 1: irrad_smoothing

Datapoint	Calibration - irrad_smoothing	cat	tag
		4	1
Access	Read/Write	Persistent Value	
Type	std::uint16_t		
Default value	0		
Description	Number of pixels used for smoothing algorithm during irradiance calibration		

7.4.3 Datapoint 3: irrad_integration_time

Datapoint	Calibration - irrad_integration_time	cat	tag
		4	3
Access	Read/Write	Persistent Value	
Type	std::uint32_t		

Default value	1000
Description	Integration time used during irradiance calibration; to be used to normalise acquired values when calculating irradiance values. Unit is microseconds.

7.4.4 Datapoint 4: irrad_conversion_table

Datapoint	Calibration - irrad_conversion_table	**cat**	**tag**
		4	4
Access	Read/Write	Persistent Value	
Type	float[4096]		
Default value	{}		
Description	Per-Pixel Conversion table obtained during irradiance calibration, unit is: 1/((16384/fullscale)*(cm^2*nm / uWatt)). The irradiance value per pixel can be calculated in (uWatt / (cm^2*nm)).		

7.4.5 Datapoint 5: irrad_calibration_type

Datapoint	Calibration - irrad_calibration_type	**cat**	**tag**
		4	5
Access	Read/Write	Persistent Value	
Type	std::uint8_t		
Default value	0		
Description	Indicates whether a diffuser was used or not, and whether stray light correction was used		

7.4.6 Datapoint 6: irrad_fiber_diameter

Datapoint	Calibration - irrad_fiber_diameter	**cat**	**tag**
		4	6
Access	Read/Write	Persistent Value	
Type	std::uint32_t		
Default value	0		
Description	Fiber diameter used during intensity calibration		

7.4.7 Datapoint 7: refl_smoothing

Datapoint	Calibration - refl_smoothing	**cat**	**tag**
		4	7
Access	Read/Write	Persistent Value	
Type	std::uint16_t		
Default value	0		
Description	Number of smoothing pixels used in Reflectance calibration		

7.4.8 Datapoint 9: refl_integration_time

Datapoint	Calibration - refl_integration_time	**cat**	**tag**
		4	9
Access	Read/Write	Persistent Value	
Type	std::uint32_t		
Default value	1000		
Description	Integration time used during Reflectance calibration		

7.4.9 Datapoint 10: refl_conversion_table

Datapoint	Calibration - refl_conversion_table	**cat**	**tag**
		4	10
Access	Read/Write	Persistent Value	
Type	float[4096]		
Default value	{}		
Description	Table of values that are used in correction of reflectance measurements		

7.4.10 Datapoint 11: straylight_correction_data_available

Datapoint	Calibration - straylight_correction_data_available	**cat**	**tag**
		4	11
Access	Read/Write	Persistent Value	
Type	bool		
Default value	0		
Description	Set to true to indicate that the straylight spectrum correction values are valid		

7.4.11 Datapoint 12: straylight_correction_pixel_begin

Datapoint	Calibration - straylight_correction_pixel_begin	**cat**	**tag**
		4	12
Access	Read/Write	Persistent Value	
Type	std::uint16_t		
Default value	0		
Description	The pixel number in which to begin the straylight corection		

7.4.12 Datapoint 13: straylight_correction_pixel_end

Datapoint	Calibration - straylight_correction_pixel_end	**cat**	**tag**
		4	13
Access	Read/Write	Persistent Value	
Type	std::uint16_t		
Default value	0		
Description	The pixel number of the last pixel in the straylight correction range		

7.5 Category 5: Data

7.5.1 Datapoint 0: oem

Datapoint	Data - oem	**cat**	**tag**
		5	0
Access	Read/Write	Persistent Value	
Type	std::uint8_t[4096]		
Default value	{}		
Description	Data region in non-volatile memory for customer to use. The content is not used nor checked by spectrometer.		

7.5.2 Datapoint 2: standalone

Datapoint	Data - standalone	cat	tag
		5	2
Access	Read/Write		
Type	std::uint8_t[44]		
Default value	{0, 0, 0, 0xFD, 0x0F, 0, 0, 0x80, 0x3F, 0, 0, 0, 0, 0x01, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}		
Description	Standalone parameters; not used by any standalone mode... is used by Demo-Application for typical measurement configuration storage		

7.6 Category 6: Digital_IO

7.6.1 Datapoint 0: digital_input_pins

Datapoint	Digital_IO - digital_input_pins	cat	tag
		6	0
Access	Read-only		
Type	std::uint32_t		
Description	Digital input status bitfield, reflecting the input value of each digital IO.		

7.6.2 Datapoint 1: digital_output_pins

Datapoint	Digital_IO - digital_output_pins	cat	tag
		6	1
Access	Read/Write		
Type	std::uint32_t		
Description	Digital output status bitfield, for setting new state of the digital IO's		

7.6.3 Datapoint 2: digital_IO_directions

Datapoint	Digital_IO - digital_IO_directions	cat	tag
		6	2
Access	Read/Write		
Type	std::uint32_t		
Description	Digital i/o direction bitfield, 0 (default) for input, 1 for output		

7.6.4 Datapoint 7: pwm_pin_1_duty_cycle

Datapoint	Digital_IO - pwm_pin_1_duty_cycle	cat	tag
		6	7
Access	Read/Write		
Type	std::uint8_t	Min	Max
		0	100
Description	Set PWM duty cycle for PWM channel 0 in percentage, the output mode of the pin is changed accordingly		

7.6.5 Datapoint 8: pwm_pin_1_frequency

Datapoint	Digital_IO - pwm_pin_1_frequency	cat	tag
		6	8
Access	Read/Write		
Type	std::uint32_t		

Description	Set PWM frequency for PWM channel 0 in Hz

7.6.6 Datapoint 9: pwm_pin_2_duty_cycle

Datapoint	Digital_IO - pwm_pin_2_duty_cycle	**cat**	**tag**
		6	9
Access	Read/Write		
Type	std::uint8_t	**Min**	**Max**
		0	100
Description	Set PWM duty cycle for PWM channel 1 in percentage		

7.6.7 Datapoint 10: pwm_pin_2_frequency

Datapoint	Digital_IO - pwm_pin_2_frequency	**cat**	**tag**
		6	10
Access	Read/Write		
Type	std::uint32_t		
Description	Set PWM frequency for PWM channel 1 in Hz		

7.6.8 Datapoint 11: pwm_pin_3_duty_cycle

Datapoint	Digital_IO - pwm_pin_3_duty_cycle	**cat**	**tag**
		6	11
Access	Read/Write		
Type	std::uint8_t	**Min**	**Max**
		0	100
Description	Set PWM duty cycle for PWM channel 2 in percentage		

7.6.9 Datapoint 12: pwm_pin_3_frequency

Datapoint	Digital_IO - pwm_pin_3_frequency	**cat**	**tag**
		6	12
Access	Read/Write		
Type	std::uint32_t		
Description	Set PWM frequency for PWM channel 2 in Hz		

7.6.10 Datapoint 13: pwm_pin_4_duty_cycle

Datapoint	Digital_IO - pwm_pin_4_duty_cycle	**cat**	**tag**
		6	13
Access	Read/Write		
Type	std::uint8_t	**Min**	**Max**
		0	100
Description	Set PWM duty cycle for PWM channel 3 in percentage		

7.6.11 Datapoint 14: pwm_pin_4_frequency

Datapoint	Digital_IO - pwm_pin_4_frequency	**cat**	**tag**
		6	14
Access	Read/Write		

Type	std::uint32_t
Description	Set PWM frequency for PWM channel 3 in Hz

7.6.12 Datapoint 15: pwm_pin_5_duty_cycle

Datapoint	Digital_IO - pwm_pin_5_duty_cycle	**cat**	**tag**
		6	15
Access	Read/Write		
Type	std::uint8_t	**Min**	**Max**
		0	100
Description	Set PWM duty cycle for PWM channel 4 in percentage, the output mode of the pin is changed accordingly		

7.6.13 Datapoint 16: pwm_pin_5_frequency

Datapoint	Digital_IO - pwm_pin_5_frequency	**cat**	**tag**
		6	16
Access	Read/Write		
Type	std::uint32_t		
Description	Set PWM frequency for PWM channel 4 in Herz		

7.6.14 Datapoint 19: board_temperature

Datapoint	Digital_IO - board_temperature	**cat**	**tag**
		6	19
Access	Read-only		
Type	std::uint16_t		
Description	Electronics board temperature in 1/100 degrees kelvin		

7.6.15 Datapoint 20: detector_temperature

Datapoint	Digital_IO - detector_temperature	**cat**	**tag**
		6	20
Access	Read-only		
Type	std::uint16_t		
Description	Detector board temperature in 1/100 degrees kelvin		

7.7 Category 7: Analog_IO

7.7.1 Datapoint 0: analog_in_pin_1

Datapoint	Analog_IO - analog_in_pin_1	**cat**	**tag**
		7	0
Access	Read-only		
Type	std::uint16_t	**Min**	**Max**
		0	5000
Description	ADC value for pin 1 in millivolt		

7.7.2 Datapoint 1: analog_out_pin_1

Datapoint	Analog_IO - analog_out_pin_1	cat	tag
		7	1
Access	Read/Write		
Type	std::uint16_t	Min	Max
		0	5000
Description	DAC value for pin 1 in millivolt		

7.8 Category 8: Version

7.8.1 Datapoint 0: firmware_version

Datapoint	Version - firmware_version	cat	tag
		8	0
Access	Read-only		
Type	std::uint32_t		
Description	Firmware version number		

7.8.2 Datapoint 1: spectrOS_version

Datapoint	Version - spectrOS_version	cat	tag
		8	1
Access	Read-only		
Type	std::uint32_t		
Description	SpectrOS version number, Version of OS used in firmware		

7.8.3 Datapoint 2: libhal_version

Datapoint	Version - libhal_version	cat	tag
		8	2
Access	Read-only		
Type	std::uint32_t		
Description	Libhal version number, Version of HAL used in firmware		

7.8.4 Datapoint 3: spectrOS_identifier

Datapoint	Version - spectrOS_identifier	cat	tag
		8	3
Access	Read-only		
Type	char[12]		
Description	SpectrOS Version Identifier, used by Avantes to identify firmware version		

7.8.5 Datapoint 4: libhal_identifier

Datapoint	Version - libhal_identifier	cat	tag
		8	4
Access	Read-only		
Type	char[12]		
Description	Libhal Version Identifier, used by Avantes to identify firmware version		

7.8.6 Datapoint 5: libhal_hardware_identifier

Datapoint	Version - libhal_hardware_identifier	cat	tag
		8	5
Access	Read-only		
Type	char[12]		
Description	Libhal Hardware Version Identifier, used by Avantes to identify firmware version		

7.8.7 Datapoint 6: fpga_version

Datapoint	Version - fpga_version	cat	tag
		8	6
Access	Read-only		
Type	std::uint32_t		
Description	FPGA version number		

7.8.8 Datapoint 7: hardware_version

Datapoint	Version - hardware_version	cat	tag
		8	7
Access	Read-only		
Type	std::uint32_t		
Description	Electronics platform version number		

8 Appendix A: Notes on Recompiling Library and Qt Demos

Except for the QT Full demo, the core library and demo's can be compiled using CMake. The library and demo's are tested on Windows, Mac, and Linux.

Build instructions are available in the readme.txt files in the folders that contain the source code of the library and demo's.

For QT Full Demo, QT and Qwt are required.

Installing Qt and Qwt

To install the Qt framework on Windows, download the QT Online installer from qt.io. The installer will guide you through the installation process. When selecting components, make sure to select the version that corresponds with your compiler, e.g MSVC 2019 64-bit and Qt Creator.

To install the charting library Qwt, download QwT from https://qwt.sourceforge.io/ and follow compilation/installation instructions.

9 Appendix B: USB permissions on Linux

By default, a standard user has no rights to open USB ports on Linux. If you run the demos as a standard user, you cannot open communication with a USB spectrometer.

One solution is to run the demos as superuser, e.g. use "sudo ./testavs" instead of "./testavs"

A better solution is to add rights to specific USB devices. How this is done differs per Linux distribution. This is how it is done on Ubuntu:

Add a file "10-avantes-rule.rules" to the directory /etc/udev/rules.d

This file contains a line per spectrometer product id:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="1992", ATTRS{idProduct}=="0667", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1992", ATTRS{idProduct}=="0668", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1992", ATTRS{idProduct}=="0669", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1992", ATTRS{idProduct}=="0670", MODE="0666"
```

After a reboot, you should now be able to open Avantes spectrometers on USB ports as a standard user.