AVANTES
MEMBER OF THE NYNOMIC GROUP

AvaSpec-Library Manual

2020

ID: 020381

AvaSpec Library

Interface Package for 32 and 64 bit Windows Applications

Interface Package for Linux Applications

Interface Package for macOS Applications

Version 9.10.0.0

USER'S MANUAL

April 2020

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: 020381

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 BV. 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: 020381

Table of Contents

1 Installation

1.1 Installation on Windows

The AvaSpec Library for Windows is offered as two packages for the following platforms:

AvaSpec-DLL package for 32-bit Windows Platforms

The AvaSpec-DLL package is a 32-bit driver interface package for the current Avantes spectrometer boards, that can be installed under the following operating systems:

- 32-bit Windows Vista/Windows7/Windows 8/Windows 10
- 64-bit Windows Vista/Windows7/Windows 8/Windows 10

The installation program for the AvaSpec-DLL package can be started by running the file "setup32.exe" from the CD-ROM. Note that the 32-bit versions of the programming environments run perfectly well on 64-bit versions of Windows.

AvaSpecX64-DLL package for 64-bit Windows Platforms

The AvaSpecX64-DLL package is the 64-bit version of the AvaSpec driver interface. It is needed when you want the Library to cooperate with 64-bit programs, like self-written 64-bit programs or the 64-bit versions of LabVIEW or MATLAB. Note that the 32-bit versions of LabVIEW or MATLAB can run perfectly well on 64-bit versions of Windows. You will have to determine your version, generally this is displayed in the *About* box of each program. The Visual Studio IDE is a 32-bit application that can generate either 32 or 64-bit programs.

The AvaSpecX64-DLL package version can be installed under the following operating systems:

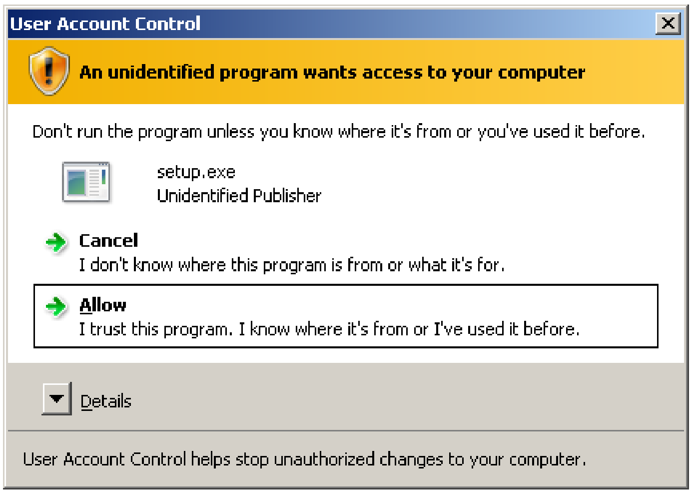
- 64-bit Windows Vista/Windows 7/Windows 8/Windows 10

The installation program for the AvaSpecX64-DLL package can be started by running the file "setup64.exe" from the CD-ROM.

The file "32 versus 64 bit development.pdf" in the root of the CD includes more detailed information which installation file to select.

This manual describes the installation, functions and sample programs of the 32 bit as well as the 64 bit AvaSpec Library.

1.2 Installation Dialogs on Windows



If you use Windows Vista, and the UAC setting is enabled, you will get the warning displayed to the left. Please select “Allow” to install the package.

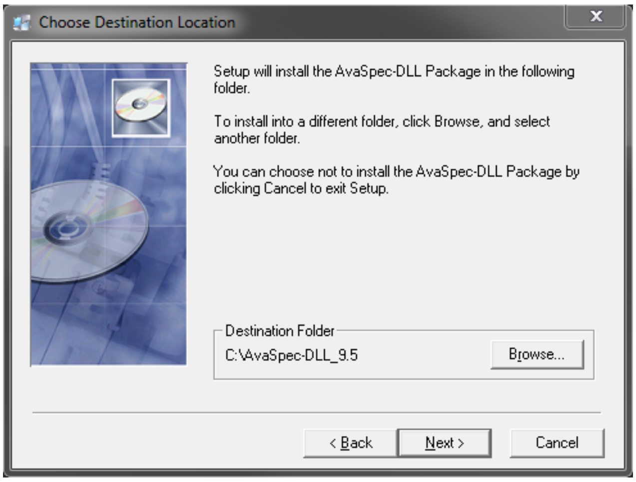
This installation is password protected. Enter the following password to proceed with the installation:

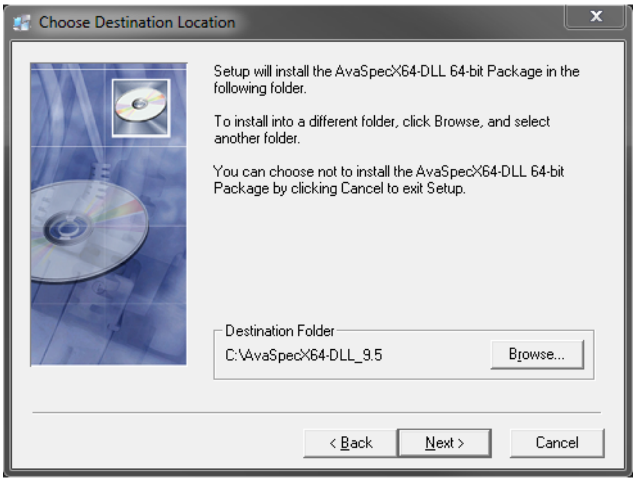
Password: `Avantes6961LL4a`

The setup program will check the system configuration of the computer. If no problems are detected, the first dialog is the “Welcome” dialog with some general information.

In the next dialog, the destination directory for the AvaSpec Library package can be selected. The default destination directory is `C:\AvaSpec-DLL_`*<version_nr>* for the 32-bit version and `C:\AvaSpecX64-DLL_`*<version_nr>* for the 64-bit version of the Library. If you want to install the software to a different directory, click the Browse button, select a new directory and click OK.

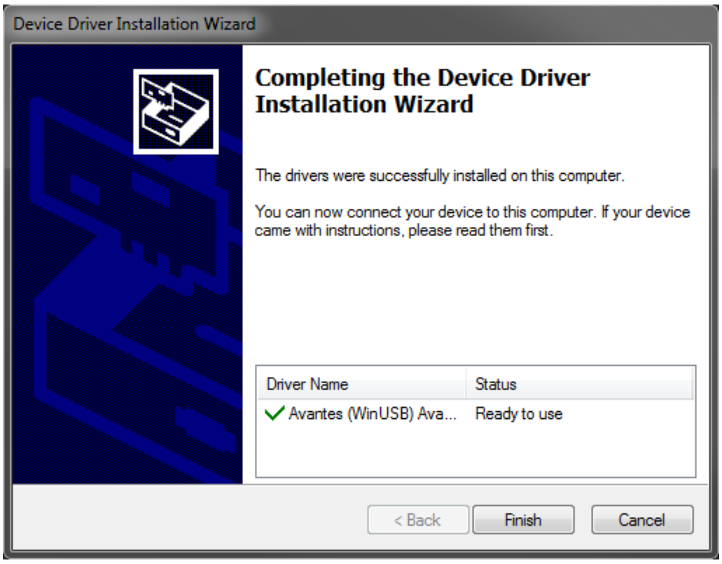
ID: 020381





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

Next, the WinUSB device driver will be installed for the AvaSpec boards. 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. If you experience problems here, please refer to Appendix A, which describes some special cases.



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 AvaSpec Library manual can be opened from the Windows Start Menu. The source code of the example programs can be found in the Examples folder, default location for both package versions is;

```
C:\AvaSpec-DLL_<version_nr>\examples\
C:\AvaSpecX64-DLL_<version_nr>\examples\
```

If you are using the Ethernet interface of the AS7010 board and experience problems opening a connection, please refer to Appendix B, which describes some common pitfalls.

1.3 Installation on Linux

The AvaSpec Library is a dynamically linked shared library for the Linux operating system. It uses the *libusb* library for USB communications. The AvaSpec Linux Library supports all available Avantes boards: the AS5216 board, the Mini board and the AS7010 board.

At present, the library is available in binary form only. Binary versions are available for a number of Linux versions. Please contact Avantes when a binary version is needed for a specific Linux version or distribution which 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)

Sample programs using the library are also available. They are written for C++ / Qt5 and Python / PyQt5. They include source code.

Connecting the hardware

Connect the USB connector to a USB port on your computer with the supplied USB cable. The AS7010 can also be connected to your network through an Ethernet cable. Depending on the presence of a DHCP server in your network, you may have to assign a fixed IP address to the board. It is recommended that you do this through the USB interface, using the full Qt5 sample, or the IP settings utility (available for Windows only).

1.4 Installation on macOS

The AvaSpec Library is a dynamically linked shared library for the macOS operating system. It uses the *libusb* library for USB communications. The AvaSpec Library supports all available Avantes boards: the AS5216 board, the Mini board and the AS7010 board.

At present, the library is available in binary form only.

Sample programs using the library are also available. They are written for C++ / Qt5, Objective-C, Swift and Python / PyQt5. They include source code.

Connecting the hardware

Connect the USB connector to a USB port on your computer with the supplied USB cable. The AS7010 can also be connected to your network through an Ethernet cable. Depending on the presence of a DHCP server in your network, you may have to assign a fixed IP address to the board. It is recommended that you do this through the USB interface, using the full Qt5 sample, or the IP settings utility (available for Windows only).

2 AvaSpec Library Description

2.1 Interface Overview

The interface from the PC to the Library is based on a function interface. The interface allows the application to configure a spectrometer and to receive and send data from and to the spectrometer.

2.2 Usage of the AvaSpec Library

The Library uses a single pair of open and close functions (AVS_Init() and AVS_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 (AVS_Init()) tries to open a communication port for all connected devices.
According to its parameter it will use Ethernet and/or USB interface to open the ports.
Please note that you will need administrative rights to open a USB port under Linux, please refer to Appendix D for more details.
The close function (AVS_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 for both USB and Ethernet devices.

2.2.1 Activating a USB Device

For a USB connected device, the following steps have to be taken (see also the sequence diagram depicted in Figure 1):

1. First call AVS_Init(0) to initialize the library to use with USB spectrometers.
2. Call AVS_UpdateUSBDevices() (formerly it was AVS_GetNrOfDevices() in earlier versions) to determine the number of attached devices.
3. Allocate buffer to store identity info;
   ```
   RequiredSize = NrDevices * sizeof(AvsIdentityType)
   ```
   NrDevices is obtained in the previous step!
4. Call AVS_GetList() with the *RequiredSize* and obtain the list of connected spectrometers. The resulting device list will be sorted on serial number.
5. Select the spectrometers you want to use with AVS_Activate() or AVS_ActivateConnCb()
6. Register a notification window handle with AVS_Register() to detect device attachment/removal (Windows and USB only)
7. If needed call AVS_Heartbeat() to check if the spectrometer is ‘alive’ and to track down internal errors, spectrometer configuration and its status.

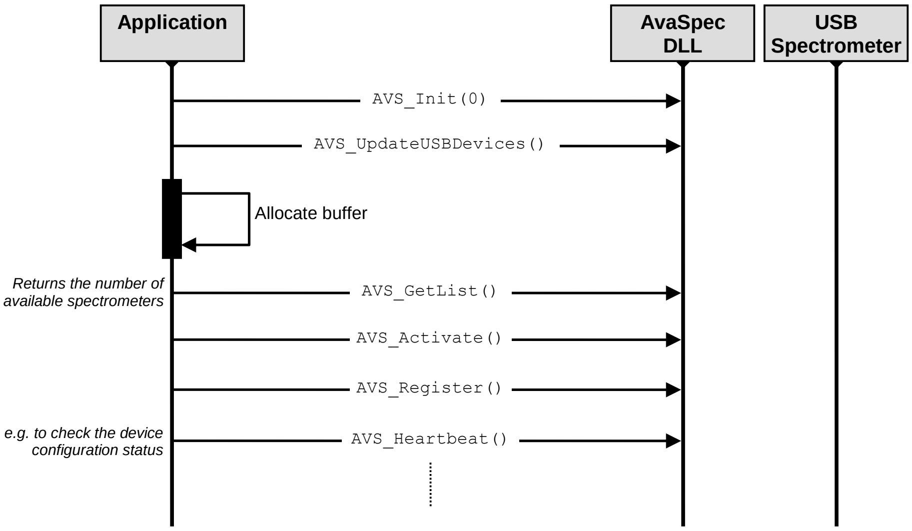


Figure 1: Command sequence to activate a USB device.

2.2.2 Activating an Ethernet Device

The following steps should be performed to search and claim the connected Ethernet spectrometers (see also the sequence diagram depicted in Figure 2);

1. First call AVS_Init(256) to initialize the library to use with Ethernet spectrometers.
2. Call AVS_UpdateETHDevices() twice to find the number of available devices without claiming one, as given with a C programming example below;

```
AVS_UpdateETHDevices(0, &RequiredSize, NULL);
DiscoveredDevs = new char[RequiredSize];
DeviceList     = (BroadcastAnswerType*)DiscoveredDevs;
DeviceListSize = AVS_UpdateETHDevices(RequiredSize,
                                      &RequiredSize,
                                      DeviceList);
```

   `DeviceList` contains now the list with available Ethernet spectrometers with their serial numbers. AVS_UpdateETHDevices() can be invoked even when AVS_Init() returns ERR_ETHCONN_REUSE error. In this way it is still possible to retrieve a list of available devices within the local network. However it is not possible to activate any spectrometer because of the presence of another Library instance on the same workstation.
3. To obtain the list of found devices (see previous step), call AVS_GetList() twice, as given with a C programming example below;

```
AVS_GetList(0, &RequiredSize, NULL );
FoundDevices   = new char[RequiredSize];
DeviceList     = (AvsIdentityType*)FoundDevices;
DeviceListSize = AVS_GetList(RequiredSize,
                             &RequiredSize, DeviceList);
```

   `DeviceList` contains now the list with available Ethernet spectrometers with their serial numbers and friendly names. The device list will be sorted on serial number.
4. With AVS_Activate() or AVS_ActivateConnCb() select the spectrometer you want to use. Use the spectrometers' serial number which is retrieved with the previous step.
5. If needed call AVS_Heartbeat() to check if the spectrometer is 'alive' and to track down internal errors, spectrometer configuration and its status.

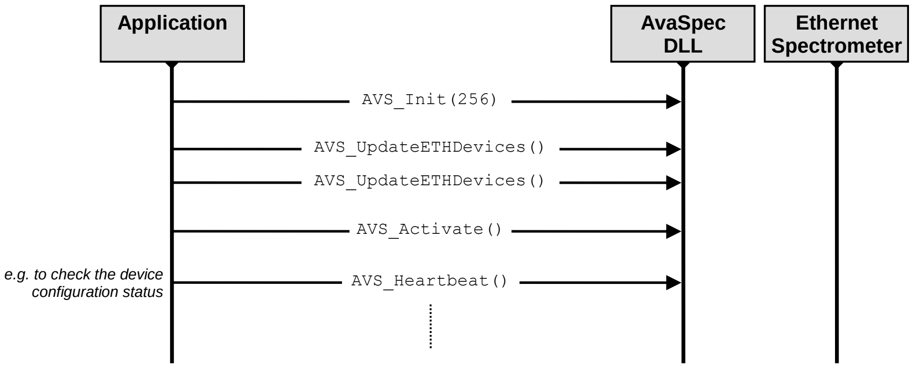


Figure 2: Command sequence to activate an Ethernet device.

2.3 Data acquisition

A spectrum can be collected by calling the function AVS_Measure(), and when a scan has been sent to the PC, it can be retrieved with the function AVS_GetScopeData(). On Windows, the preferred way to signal that a new measurement is available is by way of a Windows message. AVS_Measure() will initiate this. You can also use AVS_MeasureCallback() that a callback function to signal that a new measurement is available. On Linux and MacOS, AVS_Measure will use a callback function as well. As a final option, AVS_PollScan is available, which allows you to poll the arrival of new data. It has a much lower throughput, but is compatible with programming environments that do not support callbacks or Windows messages.

2.4 Ethernet Connection Recovery and Heartbeat (AS7010)

Once an Ethernet interface connection has been established with the spectrometer, a control sequence is used by the Library to monitor the connection status and to re-establish the connection when an unintended connection abort occurs. In case an unintended connection abort happens, the application layer will be notified by the AvaSpec Library, while the Library is still trying to re-establish the connection in the background.

An unintended Ethernet connection abort may occur due to the following reasons;

- TCP connection time-outs, triggered by the TCP layer.
- Pulling out the Spectrometer Ethernet cable manually (without a proper connection shut off).
- Cable errors.
- Problems caused by internal software issues.

For automatic connection recovery, the AvaSpec Library internally uses a heartbeat messaging mechanism. Section 2.4.1 describes the reconnection procedure that is maintained by the Library in case of an Ethernet connection failure.

2.4.1 Ethernet Auto-Recovery Sequence

Below the process flow of the used method is clarified when a spectrometer connection failure has occurred and the spectrometer connection must be re-established;

- When the application has activated an AS7010 Ethernet spectrometer (through AvaSpec Library functions AVS_Init() and AVS_Activate() or AVS_ActivateConn() or AVS_ActivateConn()), the Library will start sending Heartbeat messages to the Spectrometer with a pre-defined time interval (usually HBI=1 second, HBMW=10 seconds, see section 2.5.44). In this manner, the Ethernet auto-recovery function is started automatically and both the Library and the spectrometer will monitor the status of the Ethernet connection. When the spectrometer misses at least one heartbeat message within the time window given by HBMW, it will start the connection recovery sequence. Alternatively, when the AvaSpec Library misses at least one heartbeat response message from the spectrometer within the time window given by HBMW, it will in turn start its connection recovery sequence.

- The Ethernet auto-recovery function parameters may, however be overridden by the application using the AVS_Heartbeat() function, see section 2.5.44 for the function parameters. The application should, for this purpose, use the HeartbeatReqType fields as follows;

  HBCE=1
  EAR=1
  HBI=any desired value
  HBMW=any desired value

- In case of a connection failure, after some delay depending on the HBMW value, the Library will invoke the callback function passed to AVS_ActivateConnCb() or send a Windows message to the window with the handle passed in the AVS_ActivateConn() function. This callback function indicates to the application level that a connection status change of a particular spectrometer has occurred. Below the prototype for this callback function is given (in case AVS_ActivateConnCb() is used);

```
void(*__Conn) (AvsHandle* apHandle, int aConnStat)
```

  The first parameter is the AvsIdentityType of the spectrometer while the second parameter (aConnStat) indicates the new connection state as determined by the Library. Table 1 gives an overview of the used connection status codes.

Value	Name	Description
0	ETH_CONN_STATUS_ CONNECTING	Trying to establish ethernet. This connection state is given at start-up or after connection loss has occurred and the library is trying to reconnect with the particular spectrometer. In case of connection delay this event will be sent in between every 10 and 30 seconds time interval until the Ethernet connection has been established.
1	ETH_CONN_STATUS_ CONNECTED	Ethernet connection established and connection auto-recovery function is enabled in the Library.
2	ETH_CONN_STATUS_ CONNECTED_NOMON	Ethernet connection ready, auto-recovery function is <u>not</u> enabled in the library. No connection status updates will be received through the __Conn callback.
3	ETH_CONN_STATUS_ NOCONNECTION	Connection failure or disconnect event from the user application. The internal Library auto-recovery and network monitoring functions are stopped, from now on no restart of network monitoring functions is possible.

Table 1 Connection status codes returned from Library.

See section 2.6.2 for the Windows message parameters in case AVS_ActivateConn() is used.

In case of a connection failure has occurred with any spectrometer in the network the application can do the following;

<u>Wait for auto-recovery</u>
Just wait until the connection is re-established by the Library (which is a background task of the Library). If auto-recovery succeeds, the Library will indicate this with ETH_CONN_STATUS_CONNECTED or ETH_CONN_STATUS_CONNECTED_NOMON.

Reinitialize AvaSpec

During auto-recovery state, when a particular spectrometer connection is lost the application may try to reinitialize AvaSpec to rebuild the spectrometer list. The procedure to perform for this re-initialization sequence is further explained in section 2.4.2.

2.4.2 Application Level Reinitialization

After a connection failure has been detected by the Library it will start the connection auto-recovery sequence. Just like with the initial connection set-up, during auto-recovery the Library will start sending the status ETH_CONN_STATUS_CONNECTING to the application through the __Conn callback. When the Library is still busy reestablishing its connection with the particular spectrometer, it will still notify with the ETH_CONN_STATUS_CONNECTING status in between every 10 and 30 seconds by using the same callback. During these “connecting” events the application may try to reinitialize AvaSpec to rebuild the spectrometer list at any time.

Figure 3 depicts a sequence diagram of the complete sequence of establishing connection with a spectrometer, connection failure detection and re-establishing the connection from the application point of view (in case of a temporary connection failure).

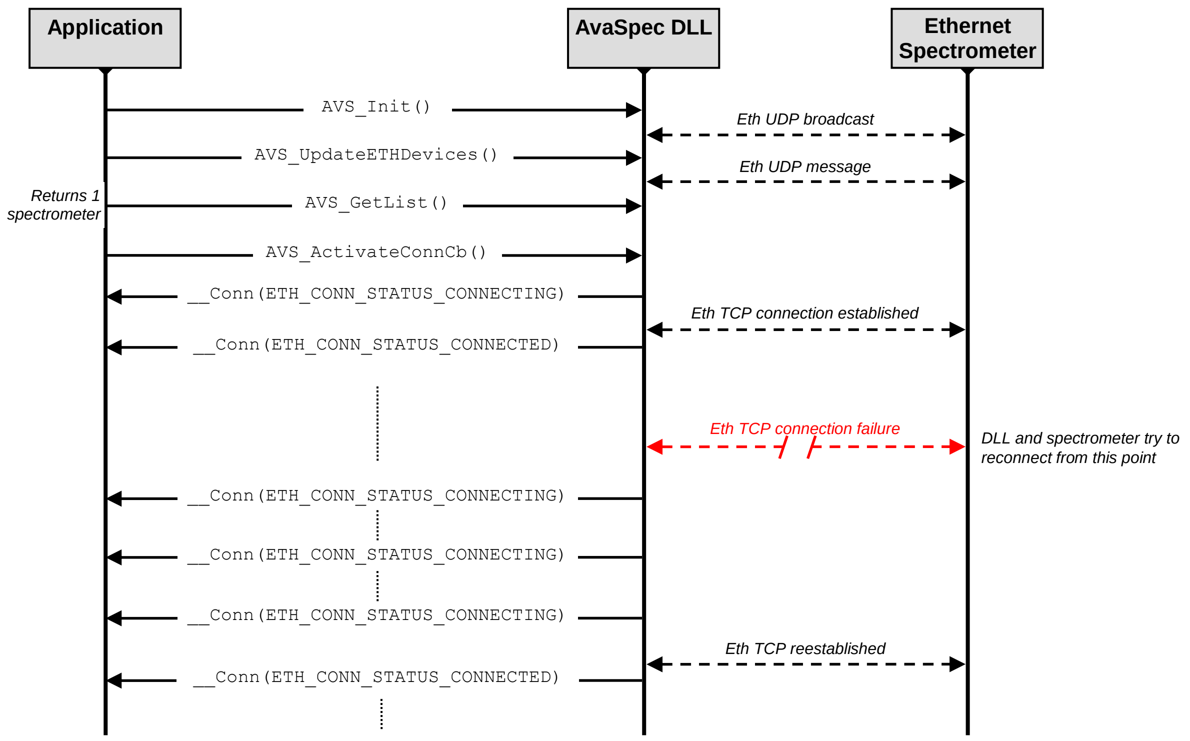


Figure 3: Establishing / re-establishing sequence for TCP connections.

Figure 4 shows a use-case of a permanent TCP connection failure, possibly caused by Ethernet cable issues or other hardware problems.

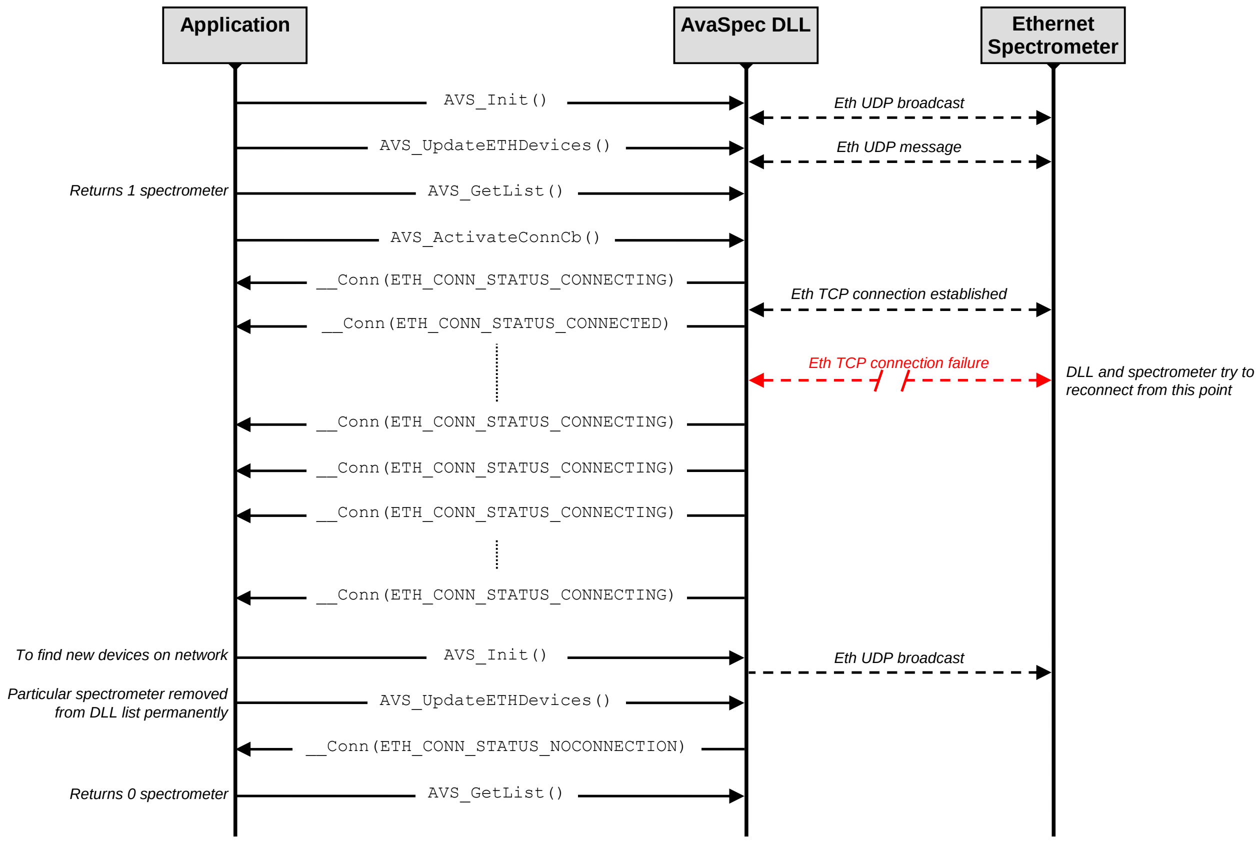


Figure 4: Permanent connection failure sequence diagram.

2.5 Exported Functions

2.5.1 AVS_Init

Function:	**int AVS_Init** (short a_Port)		
Group:	Blocking control function		
Description:	Initializes the communication interface with the spectrometers and the internal data structures. For Ethernet devices this function will create a list of available Ethernet spectrometers within all the network interfaces of the host.		
Parameters:	a_Port:	ID of port to be used, defined as follows;	
		-1:	Use both Ethernet (AS7010) and USB ports
		0:	Use USB port
		1..255:	Not supported in this version
		256:	AS7010: Use Ethernet port
Return:	On success:	Number of USB connected and/or found (through Ethernet) devices. Zero when non found.	
	On error:	ERR_CONNECTION_FAILURE ERR_ETHCONN_REUSE	

2.5.2 AVS_Done

Function:	**int AVS_Done** (void)
Group:	Blocking control function
Description:	Closes the communication and releases internal storage.
Parameters:	None
Return:	SUCCESS

2.5.3 AVS_GetNrOfDevices

Deprecated function, replaced by AVS_UpdateUSBDevices(). The functionality is identical.

2.5.4 AVS_UpdateUSBDevices

Function:	**int AVS_UpdateUSBDevices** (void)	
Group:	Blocking control function	
Description:	Internally checks the list of connected USB devices and returns the number of devices attached. If AVS_Init() is called with a_Port=-1, the return value also includes the number of ETH devices.	
Parameters:	None	
Return:	> 0:	Number of devices in the list
	0:	No devices found

2.5.5 AVS_UpdateETHDevices

Function:	**int AVS_UpdateETHDevices** (unsigned int unsigned int* BroadcastAnswerType*)	 a_ListSize, a_pRequiredSize, a_pList
Group:	Blocking control function	
Description:	Internally checks the list of connected Ethernet devices and returns the number of devices in the device list. If AVS_Init() is called with a_Port=-1, the return value also includes the number of USB devices. a_pList points to a buffer containing the information returned by all Ethernet devices after receiving an UDP broadcast sent by the function.	
Parameters:	a_ListSize:	Number of bytes allocated by the caller to store the list data
	a_pRequiredSize:	Number of bytes needed to store information
	a_pList:	Pointer to allocated buffer to store the broadcast answer
Return:	> 0:	Number of devices in the list
	0:	No devices found
	ERROR_INVALID_SIZE	If (a_pRequiredSize > a_ListSize) then allocate larger buffer and retry operation

2.5.6 AVS_GetList

Function:	**int AVS_GetList** (unsigned int unsigned int* AvsIdentityType*)	 a_ListSize, a_pRequiredSize, a_pList
Group:	Blocking control function	
Description:	Returns device information for each spectrometer connected to the ports indicated at AVS_Init().	
Parameters:	a_ListSize:	Number of bytes allocated by the caller to store the list data
	a_pRequiredSize:	Number of bytes needed to store information
	a_pList:	Pointer to allocated buffer to store identity information. The resulting device list will be sorted on serial number.
Return:	> 0:	Number of devices in the list
	0:	No devices found
	ERROR_INVALID_SIZE	If (a_pRequiredSize > a_ListSize) then allocate larger buffer and retry operation

2.5.7 AVS_GetHandleFromSerial

Function:	**AvsHandle AVS_GetHandleFromSerial** (char*)	 a_pSerial
Group:	Blocking control function	
Description:	Retrieves the AvsHandle for the spectrometer with serialnumber a_pSerial.	
Parameters:	a_ pSerial	The serialnumber of the spectrometer
Return:	On success:	AvsHandle, handle to be used in subsequent function calls
	On error:	INVALID_AVS_HANDLE_VALUE

2.5.8 AVS_Activate

Function:	**AvsHandle AVS_Activate** (AvsIdentityType* a_pDeviceId)	
Group:	Blocking control function	
Description:	Activates selected spectrometer for communication.	
Parameters:	a_pDeviceId	Device identifier as specified by AvsIdentityType
Return:	On success:	AvsHandle, handle to be used in subsequent function calls
	On error:	INVALID_AVS_HANDLE_VALUE

2.5.9 AVS_ActivateConnCb

Function:	**AvsHandle AVS_ActivateConnCb** (AvsIdentityType* void)	 a_pDeviceId (*__Conn) (AvsHandle*, int)
Group:	Blocking control function	
Description:	Activates selected spectrometer for communication and registers a Connection Status callback routine. This callback routine will be called by the Library when a connection status change has occurred. This callback routine must be used by the application layer to ensure connection reliability. Either AVS_Activate() or AVS_ActivateConnCb() must be used to activate the device. The callback function works only with Ethernet spectrometers. See section 2.4 for more information.	
Parameters:	a_pDeviceId	Device identifier as specified by AvsIdentityType
	(*__Conn) (AvsHandle*, int)	Pointer to a Callback function to notify Ethernet connection status change.
Return:	On success:	AvsHandle, handle to be used in subsequent function calls
	On error:	INVALID_AVS_HANDLE_VALUE

2.5.10 AVS_ActivateConn

In the Linux/macOS version of this library both versions of this function perform the same action. In the Windows DLL version the "activate" function with "callback" is available by calling AVS_ActivateConnCb() while AVS_ActivateConn() uses the Windows messaging system.

Function:	**AvsHandle AVS_ActivateConn** (AvsIdentityType* a_pDeviceId, void* a_hWnd)	
Group:	Blocking control function	
Description:	Activates selected spectrometer for communication and registers a Connection Status Windows message. This Windows message callback will be activated through the Library when a connection status change has occurred. This Windows message callback must be used by the application layer to ensure and monitor connection reliability. Either AVS_Activate(), AVS_ActivateConnCb() or AVS_ActivateConn() must be used to activate the device. The callback function works only with Ethernet spectrometers. See section 2.4 for more information.	
Parameters:	a_pDeviceId	Device identifier as specified by AvsIdentityType
	a_hWnd	Windows handle to notify application that the Ethernet connection status has changed. The Library sends a message with the identifier WM_CONN_STATUS to the window with handle a_hWnd. See sections 2.4 and 2.6.2 for detailed information on the message parameters.
Return:	On success:	AvsHandle, handle to be used in subsequent function calls
	On error:	INVALID_AVS_HANDLE_VALUE

2.5.11 AVS_Deactivate

Function:	**bool AVS_Deactivate** (AvsHandle a_hDeviceId)	
Group:	Blocking control function	
Description:	Closes communication with selected spectrometer.	
Parameters:	a_hDeviceId:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
Return:	true:	Device successfully closed
	false:	Device identifier not found

2.5.12 AVS_Register (Windows only)

Function:	**bool AVS_Register**	
	(	
	HWND	a_hWnd
	)	
Group:	Blocking control function	
Description:	Installs an application windows handle to which device attachment/removal messages have to be sent. This is only supported on USB connections, and only on Windows. Note that when connecting an AS7010 to USB, the Windows Arrival message will be sent before the board receives USB power. Therefore it is recommended to implement a delay of 5 seconds after receiving the DEVICEARRIVAL message, before the application calls AVS_GetNrOfDevices() and rebuilds the list (AVS_GetList())	
Parameters:	a_hWnd:	Application window handle
Return:	true:	Registration successful
	false:	registration failed or function not supported on OS

2.5.13 AVS_PrepareMeasure

Function:	**int AVS_PrepareMeasure**	
	(	
	AvsHandle	a_hDevice,
	MeasConfigType*	a_pMeasConfig
	)	
Group:	Blocking data write function	
Description:	Prepares measurement on the spectrometer using the specified measurement configuration.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pMeasConfig:	Pointer to structure containing measurement configuration
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_OPERATION_PENDING ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER ERR_INVALID_PIXEL_RANGE ERR_INVALID_CONFIGURATION (invalid fpga type) ERR_TIMEOUT ERR_INVALID_MEASPARAM_DYNDARK

2.5.14 AVS_Measure

In the Linux/macOS version of this library both versions of this function perform the same action. In the Windows DLL version the "measure" function with "callback" is available by calling AVS_MeasureCallback() while AVS_Measure() uses the Windows messaging system.

Function:	**int AVS_Measure**	
	(	
	AvsHandle	a_hDevice,
	HWND	a_hWnd,
	short	a_Nmsr
	)	
Group:	Non-Blocking data write function	
Description:	Starts measurement on the spectrometer	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_hWnd:	Window handle to notify application measurement result data is available. The Library sends a Windows message to the window with command WM_MEAS_READY, with SUCCESS, the number of scans that were saved in RAM (if StoreToRAM parameter > 0), or INVALID_MEAS_DATA as WPARM value and a_hDevice as LPARM value. Use this parameter on Windows only.
	a_Nmsr:	Number of measurements to do after one single call to AVS_Measure() (-1 is infinite, -2 is used to start Dynamic StoreToRam)
Return:	On success:	ERR_SUCCESS
	On error:	ERR_OPERATION_PENDING ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER ERR_INVALID_STATE

2.5.15 AVS_MeasureCallback

Function:	**int AVS_MeasureCallback**	
	(	
	AvsHandle	a_hDevice,
	void	(*__Done)(AvsHandle*, int*),
	short	a_Nmsr
	)	
Group:	Non-Blocking data write function	
Description:	Starts measurement on the spectrometer	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	(*__Done)(AvsHandle*, int*):	Pointer to a Callback function to notify application measurement result data is available. The Library will call the given function to notify a measurement is ready, the int* parameter will be set to the value SUCCESS if a new scan is available, to the number of scans that were saved in RAM (if StoreToRAM parameter > 0), or to INVALID_MEAS_DATA. Set this value to NULL if callback is not supported or needed.
	a_Nmsr:	Number of measurements to do after one single call to AVS_Measure() (-1 is infinite, -2 is used to start Dynamic StoreToRam)
Return:	On success:	ERR_SUCCESS
	On error:	ERR_OPERATION_PENDING ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER ERR_INVALID_STATE

2.5.16 AVS_MeasureLV

Windows/LabView measurement function only. See LabView support documentation.

2.5.17 AVS_GetLambda

Function:	**int AVS_GetLambda**	
	(	
	AvsHandle	a_hDevice,
	double*	a_pWavelength
	)	
Group:	Internal data read function	
Description:	Returns the wavelength values corresponding to the pixels if available. This information is stored in the Library during the AVS_Activate() procedure. The Library does not test if a_pWaveLength is correctly allocated by the caller!	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pWaveLength:	array of double, with array size equal to number of pixels
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID

2.5.18 AVS_GetNumPixels

Function:	**int AVS_GetNumPixels**	
	(	
	AvsHandle	a_hDevice,
	unsigned short*	a_pNumPixels
	)	
Group:	Internal data read function	
Description:	Returns the number of pixels of a spectrometer. This information is stored in the Library during the AVS_Activate() procedure.	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pNumPixels:	pointer to unsigned integer to store number of pixels
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID

2.5.19 AVS_GetParameter

Function:	**int AVS_GetParameter**	
	(	
	AvsHandle	a_hDevice,
	unsigned int	a_Size,
	unsigned int*	a_pRequiredSize,
	DeviceConfigType*	a_pData
	)	
Group:	Blocking control function	
Description:	Returns the device information of the spectrometer. Use the AVS_Heartbeat() AvaSpec function (DCS field) to determine the status of the used device configuration on the spectrometer.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_Size:	Number of bytes allocated by caller to store DeviceConfigType
	a_pRequiredSize:	Number of bytes needed to store DeviceConfigType
	a_pData:	Pointer to buffer that will be filled with the spectrometer configuration data
Return:	On success:	ERR_SUCCESS

	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_SIZE (a_Size is smaller than required size) ERR_INTERNAL_READ

2.5.20 AVS_GetOemParameter

Function:	**int AVS_GetOemParameter** (AvsHandle OemDataType*)	 a_hDevice, a_pOemData
Group:	Blocking control function	
Description:	Returns the OEM data structure available on the spectrometer.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pOemData:	Pointer to an allocated buffer in which the spectrometer OEM data will be copied
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DLL_INITIALISATION ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER ERR_NO_MEMORY ERR_INVALID_REPLY ERR_TIMEOUT ERR_COMMUNICATION

2.5.21 AVS_GetIPAddress

Function:	**int AVS_GetIPAddress** (AvsIdentityType* char* int*)	 a_pDeviceId, a_pIp, a_size
Group:	Blocking read function	
Description:	Returns the IP address for the spectrometer identified by a_pDeviceId.	
Parameters:	a_pDeviceId:	AvsIdentity of desired spectrometer
	a_pIp:	Output; will be filled with A NULL terminated character string representing a "." (dotted) notation number. Size of this buffer must be at least 16 bytes long (including NULL termination).
	a_size	Number of allocated char for a_pIp
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DLL_INITIALISATION ERR_DEVICE_NOT_FOUND ERR_OPERATION_NOT_SUPPORTED (On USB devices)

2.5.22 AVS_GetComType

Function:	**int AVS_GetComType**	
	(	
	AvsIdentityType*	a_pDeviceId,
	int*	a_type
	)	
Group:	Blocking read function	
Description:	Returns the communication protocol	
Parameters:	a_pDeviceId:	AvsIdentity of desired spectrometer
	a_type:	Output, communication type: RS232 = 0, USB5216 = 1, USBMINI = 2, USB7010 = 3, ETH7010 = 4 -1 when identity given with a_pDeviceId is unknown
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DLL_INITIALISATION -1 when identity given with a_pDeviceId is unknown

2.5.23 AVS_PollScan

Function:	**int AVS_PollScan**	
	(	
	AvsHandle	a_hDevice
	)	
Group:	Internal data read function	
Description:	Determines if new measurement results are available. The most effective way to let the application know when a new measurement is ready, is by using Windows Messaging in which case the AvaSpec Library sends a WM_MEAS_READY message to the application as soon as a measurement is ready to be imported into the application software (see also section 2.6.2 explaining Windows messages). But if the programming environment does not support Windows Messaging, it is also possible to use AVS_PollScan() for this purpose. After a measurement request has been posted by calling AVS_Measure(), the function AVS_PollScan() can be called in a loop until it returns "1". Note that it should be avoided that AVS_PollScan() is called continuously without any delay. This can cause such a heavy CPU load that this can freeze the application software after a while. Adding a 1 millisecond delay (so polling every ms) already solves this problem.	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
Return:	On success:	0: no data available 1: data available
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID

2.5.24 AVS_GetScopeData

Function:	**int AVS_GetScopeData** (AvsHandle a_hDevice, unsigned int* a_pTimeLabel, double* a_pSpectrum)	
Group:	Internal data read function,	
Description:	Returns the pixel values of the last performed measurement. Should be called by the application after the notification on AVS_Measure() is triggered. The Library does not check the allocated buffer size!	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pTimeLabel:	ticks count last pixel of spectrum is received by microcontroller ticks in 10 μS units since spectrometer started
	a_pSpectrum:	array of doubles, size equal to the selected pixelrange
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_MEAS_DATA (no measurement data received)

2.5.25 AVS_GetSaturatedPixels

Function:	**int AVS_GetSaturatedPixels** (AvsHandle a_hDevice, unsigned char* a_pSaturated)	
Group:	Internal data read function,	
Description:	Returns for each pixel if that pixel was saturated (1) or not (0).	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pSaturated:	array of chars (each char indicates if saturation occurred for corresponding pixel), size equal to the selected pixel range
Return:	On success:	ERR_ SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_MEAS_DATA (no measurement data received) ERR_OPERATION_NOT_SUPPORTED ERR_OPERATION_NOT_ENABLED

2.5.26 AVS_GetAnalogIn

Function:	**int AVS_GetAnalogIn**	
	(	
	AvsHandle	a_hDevice,
	unsigned char	a_AnalogInId,
	float*	a_pAnalogIn
	)	
Group:	Blocking control function.	
Description:	Returns the status of the specified analog input	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_AnalogInId:	identifier of analog input <u>AS5216</u> 0 = thermistor on optical bench (NIR 2.0 / NIR2.2 / NIR 2.5 / TEC) 1 = 1V2 2 = 5VIO 3 = 5VUSB 4 = AI2 = pin 18 at 26-pins connector 5 = AI1 = pin 9 at 26-pins connector 6 = NTC1 onboard thermistor 7 = Not used <u>Mini</u> 0 = NTC1 onboard thermistor 1 = Not used 2 = Not used 3 = Not used 4 = AI2 = pin 13 on micro HDMI = pin 11 on HDMI Terminal 5 = AI1 = pin 16 on micro HDMI = pin 17 on HDMI Terminal 6 = Not used 7 = Not used <u>Mini MkII:</u> 0 = AI1 = pin 8 on 10-pins WR-WTB connector <u>AS7010</u> 0 = thermistor on optical bench (NIR 2.0 / NIR2.2 / NIR 2.5 / TEC) 1 = Not used 2 = Not used 3 = Not used 4 = AI2 = pin 18 at 26-pins connector 5 = AI1 = pin 9 at 26-pins connector 6 = digital temperature sensor, returns degrees Celsius (not Volts) 7 = Not used
	a_pAnalogIn:	pointer to float for analog input value [Volts or degrees Celsius]
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER (invalid analog input ID) ERR_TIMEOUT (error in communication) ERR_INTERNAL_READ

2.5.27 AVS_GetDigIn

Function:	**int AVS_GetDigIn** (AvsHandle unsigned char unsigned char*)	 a_hDevice, a_DigInId, a_pDigIn
Group:	Blocking control function.	
Description:	Returns the status of the specified digital input	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_DigInId:	identifier of digital input AS5216: 0 = DI1 = Pin 24 at 26-pins connector 1 = DI2 = Pin 7 at 26-pins connector 2 = DI3 = Pin 16 at 26-pins connector Mini: 0 = DI1 = Pin 7 on Micro HDMI = Pin 5 on HDMI terminal 1 = DI2 = Pin 5 on Micro HDMI = Pin 3 on HDMI Terminal 2 = DI3 = Pin 3 on Micro HDMI = Pin 1 on HDMI Terminal 3 = DI4 = Pin 1 on Micro HDMI = Pin 19 on HDMI Terminal 4 = DI5 = Pin 4 on Micro HDMI = Pin 2 on HDMI Terminal 5 = DI6 = Pin 2 on Micro HDMI = Pin 14 on HDMI Terminal Mini MkII: 0 = DI1 = Pin 1 on 10-pins WR-WTB connector 1 = DI2 = Pin 2 on 10-pins WR-WTB connector 2 = DI3 = Pin 3 on 10-pins WR-WTB connector AS7010: 0 = DI1 = Pin 24 at 26-pins connector 1 = DI2 = Pin 7 at 26-pins connector 2 = DI3 = Pin 16 at 26-pins
	a_pDigIn:	pointer to digital input status (0 – 1)
Return:	On success:	ERR_SUCCESS, a_pDigIn contains valid value
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER (invalid digital input id.) ERR_TIMEOUT (error in communication)

2.5.28 AVS_GetVersionInfo

Function:	**int AVS_GetVersionInfo** (AvsHandle a_hDevice, unsigned char* a_pFPGAVersion, unsigned char* a_pFirmwareVersion, unsigned char* a_pDLLVersion)	
Group:	Blocking read function	
Description:	Returns the status of the software version of the different parts. Library does not check the size of the buffers allocated by the caller.	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pFPGAVersion:	pointer to buffer to store FPGA software version (16 char.)
	a_pFirmwareVersion:	pointer to buffer to store Microcontroller software version (16 char.)
	a_pDLLVersion:	pointer to buffer to store Library software version (16 char.)
Return:	On success:	ERR_SUCCESS, buffer contains valid value
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication)

2.5.29 AVS_GetDLLVersion

Function:	**int AVS_GetDLLVersion** (char* a_pVersionString)	
Group:	Blocking read function.	
Description:	Gets the file version information for the Avaspec Library.	
Parameters:	a_pVersionString:	The version information string
Return:		ERR_SUCCESS

2.5.30 AVS_SetParameter

Function:	**int AVS_SetParameter** (AvsHandle a_hDevice, DeviceConfigType* a_pData)	
Group:	Blocking data send function.	
Description:	Overwrites the device configuration data on the spectrometer. The data is not checked. ***Please note that OemDataType is part of the DeviceConfigType in EEPROM (see section 2.6). Precautions must be taken to prevent OemData overwrites when using AVS_SetParameter() function together with AVS_SetOemParameter().*** Use the AVS_Heartbeat() AvaSpec function (DCS field) to determine the status of the used device configuration on the spectrometer.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pData:	Pointer to a DeviceConfigType structure
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_OPERATION_PENDING ERR_INVALID_STATE (measurement pending) ERR_INTERNAL_WRITE

2.5.31 AVS_ResetParameter

Function:	**int AVS_ResetParameter** (AvsHandle a_hDevice)	
Group:	Blocking data send function.	
Description:	Resets onboard device parameter section to its factory defaults. This command will result in the loss of all user-specific device configuration settings which is set through the AvaSpec function AVS_SetParameter(), as defined in section 2.5.29. ***Please note that OemDataType is part of the DeviceConfigType in EEPROM (see section 2.6). When invoking AVS_ResetParameter() the OEM data part will also be erased.*** Use the AVS_Heartbeat() AvaSpec function (DCS field) to determine the status of the used device configuration on the spectrometer.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication)

2.5.32 AVS_SetOemParameter

Function:	**int AVS_SetOemParameter** (AvsHandle OemDataType*)	 a_hDevice, a_pOemData
Group:	Blocking data send function.	
Description:	Sends the OEM data structure to the spectrometer. ***Please note that OemDataType is part of the DeviceConfigType in EEPROM (see section 2.6). Precautions must be taken to prevent OemData overwrites when using AVS_SetParameter() function together with AVS_SetOemParameter().***	
Parameters:	a_hDevice: a_pOemData:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb() Pointer to an allocated buffer, containing the data which will be sent to the spectrometer
Return:	On success: On error:	ERR_SUCCESS ERR_DLL_INITIALISATION ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER ERR_NO_MEMORY ERR_INVALID_REPLY ERR_TIMEOUT ERR_COMMUNICATION

2.5.33 AVS_SetAnalogOut

Function:	**int AVS_SetAnalogOut**	
	(	
	AvsHandle	a_hDevice,
	unsigned char	a_PortId,
	float	a_Value
	)	
Group:	Blocking data send function	
Description:	Sets the analog output value for the specified analog output	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_PortId:	identifier for one of the two output signals: AS5216: 0 = AO1 = pin 17 at 26-pins connector 1 = AO2 = pin 26 at 26-pins connector Mini: 0 = AO1 = Pin 12 on Micro HDMI = Pin 10 on HDMI terminal 1 = AO2 = Pin 14 on Micro HDMI = Pin 12 on HDMI terminal Mini MkII: 0 = AO1 = pin 9 on 10-pins WR-WTB connector AS7010: 0 = AO1 = pin 17 at 26-pins connector 1 = AO2 = pin 26 at 26-pins connector
	a_Value:	DAC value to be set in Volts (internally an 8-bits DAC is used) with range 0 – 5.0V
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER

2.5.34 AVS_SetDigOut

Function	**int AVS_SetDigOut**	
	(	
	AvsHandle	a_hDevice
	unsigned char	a_PortId,
	unsigned char	a_Value
	)	
Group:	Blocking data send function.	
Description:	Sets the digital output value for the specified digital output	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_PortId:	identifier for one of the 10 output signals: AS5216: 0 = DO1 = pin 11 at 26-pins connector 1 = DO2 = pin 2 at 26-pins connector 2 = DO3 = pin 20 at 26-pins connector 3 = DO4 = pin 12 at 26-pins connector 4 = DO5 = pin 3 at 26-pins connector 5 = DO6 = pin 21 at 26-pins connector 6 = DO7 = pin 13 at 26-pins connector 7 = DO8 = pin 4 at 26-pins connector 8 = DO9 = pin 22 at 26-pins connector 9 = DO10 = pin 25 at 26-pins connector Mini: 0 = DO1 = Pin 7 on Micro HDMI = Pin 5 on HDMI terminal 1 = DO2 = Pin 5 on Micro HDMI = Pin 3 on HDMI Terminal 2 = DO3 = Pin 3 on Micro HDMI = Pin 1 on HDMI Terminal 3 = DO4 = Pin 1 on Micro HDMI = Pin 19 on HDMI Terminal 4 = DO5 = Pin 4 on Micro HDMI = Pin 2 on HDMI Terminal 5 = DO6 = Pin 2 on Micro HDMI = Pin 14 on HDMI Terminal Mini MkII: 0 = DO1 = Pin 1 on 10-pins WR-WTB connector 1 = DO2 = Pin 2 on 10-pins WR-WTB connector 2 = DO3 = Pin 3 on 10-pins WR-WTB connector 3 = DO4 = Pin 4 on 10-pins WR-WTB connector 4 = DO5 = Pin 5 on 10-pins WR-WTB connector AS7010: 0 = DO1 = pin 11 at 26-pins connector 1 = DO2 = pin 2 at 26-pins connector 2 = DO3 = pin 20 at 26-pins connector 3 = DO4 = pin 12 at 26-pins connector 4 = DO5 = pin 3 at 26-pins connector 5 = DO6 = pin 21 at 26-pins connector 6 = DO7 = pin 13 at 26-pins connector 7 = DO8 = pin 4 at 26-pins connector 8 = DO9 = pin 22 at 26-pins connector 9 = DO10 = pin 25 at 26-pins connector
	a_Value:	value to be set (0-1)
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER

 www.avantes.com info@avantes.com

2.5.35 AVS_SetPwmOut

Function:	**int AVS_SetPwmOut** (AvsHandle unsigned char unsigned int unsigned char)	 a_hDevice, a_PortId, a_Frequency, a_DutyCycle
Group:	Blocking data send function.	
Description:	Selects the PWM functionality for the specified digital output	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_PortId:	identifier for one of the 6 PWM output signals: 0 = DO1 = pin 11 at 26-pins connector 1 = DO2 = pin 2 at 26-pins connector 2 = DO3 = pin 20 at 26-pins connector 4 = DO5 = pin 3 at 26-pins connector 5 = DO6 = pin 21 at 26-pins connector 6 = DO7 = pin 13 at 26-pins connector
	a_Frequency:	desired PWM frequency (500 – 300000) [Hz] For the AS5216, the frequency of outputs 0, 1 and 2 is the same (the last specified frequency is used) and also the frequency of outputs 4, 5 and 6 is the same. For the AS7010, you can define six different frequencies.
	a_DutyCycle:	percentage high time in one cycle (0 – 100) For the AS5216, channels 0, 1 and 2 have a synchronized rising edge, the same holds for channels 4, 5 and 6. For the AS7010, rising edges are unsynchronized.
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER
Remark:	The PWM functionality is not supported on the Mini	

2.5.36 AVS_SetSyncMode

Function:	**int AVS_SetSyncMode** (AvsHandle unsigned char)	 a_hDevice, a_Enable
Group:	Internal Library write function	
Description	Disables/enables support for synchronous measurement. Library takes care of dividing Nmsr request into Nmsr number of single measurement requests.	
Parameters	a_hDevice:	master device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_Enable:	0 is disable sync mode, 1 is enables sync mode
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID

2.5.37 AVS_StopMeasure

Function:	**int AVS_StopMeasure**	
	(	
	AvsHandle	a_hDevice
	)	
Group:	Blocking data send function	
Description:	Stops the measurements (needed if Nmsr = infinite), can also be used to stop a pending measurement with long integration time and/or high number of averages	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER

2.5.38 AVS_SetPrescanMode

Function:	**int AVS_SetPrescanMode**	
	(	
	AvsHandle	a_hDevice
	bool	a_Prescan
	)	
Group:	Blocking data send function	
Description:	If a_Prescan is set, the first measurement result will be skipped. This function is only useful for the AvaSpec-3648 because this detector can be operated in prescan mode, or clearbuffer mode (see below)	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_Prescan:	If true, the first measurement result will be skipped (prescan mode), else the detector will be cleared before each new scan (clearbuffer mode)
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication)

The Toshiba detector in the AvaSpec-3648, can be used in 2 different control modes:

- **The Prescan mode (default mode).**
  In this mode the Toshiba detector will automatically generate an additional prescan for every request from the PC, the first scan contains non-linear data and will be rejected, the 2^{nd} scan contains linear data and will be sent to the PC. This prescan mode is default and should be used in most applications, like with averaging (only one prescan is generated for a nr of averages), with the use of an AvaLight-XE (one or more flashes per scan) and with multichannel spectrometers. The advantage of this mode is a very stable and linear spectrum. The disadvantage of this mode is that a minor (<5%) image of the previous scan (ghost spectrum) is included in the signal. This mode cannot be used if the integration time cycle needs to start within microseconds after the spectrometer is externally triggered, but since the prescan duration is exactly known at each integration time, accurate timing (21 nanoseconds precision in external trigger mode) is very well possible in prescan mode.

- **The Clear-Buffer mode.**
  In this mode the Toshiba detector buffer will be cleared, before a scan is taken. This clear-buffer mode should be used when timing is important, like with fast external triggering. The advantage of this mode is that a scan will start at the time of an external trigger, the disadvantage of this mode is that after clearing the buffer, the detector will have a minor threshold, in which small signals (<500 counts) will not appear and with different integration times the detector is not linear.

2.5.39 AVS_UseHighResAdc

Function:	**int AVS_UseHighResAdc**	
	(	
	AvsHandle	a_hDevice
	bool	a_Enable
	)	
Group:	Internal Library write function.	
Description:	With the AS5216 electronic board revision 1D and later, a 16bit resolution AD Converter is used instead of a 14bit in earlier hardware versions. As a result, the ADC Counts scale can be set to the full 16 bit (0..65535) Counts. For compatibility reasons with previous hardware revisions, the default range is set to 14 bit (0..16383.75) ADC Counts.	
Remark:	When using the 16 bit ADC in full High Resolution mode (0..65535), please note that the irradiance intensity calibration, as well as the nonlinearity calibration are based on the 14bit ADC range. Therefore, if using the nonlinearity correction or irradiance calibration in your own software using the High Resolution mode, you need to apply the additional correction with ADCFactor (= 4.0), as explained in detail in section 3.8.1 and 3.8.3.	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_Enable:	True: use 16bit resolution, ADC Counts range 0..65535 False: use 14bit resolution ADC Counts range 0..16383.75
Return:	On success:	ERR_SUCCESS
	On error:	ERR_OPERATION_NOT_SUPPORTED: this function is not supported by AS5216 hardware version R1C or earlier

2.5.40 AVS_SetSensitivityMode

Function:	**int AVS_SetSensitivityMode** (AvsHandle a_hDevice unsigned int a_SensitivityMode)	
Group:	Blocking data send function.	
Description:	The AvaSpec-NIR models can be operated in LowNoise (a_SensitivityMode = 0) or High Sensitivity Mode (a_SensitivityMode > 0).	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_SensitivityMode:	0 = LowNoise, >0 = High Sensitivity
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_NOT_SUPPORTED_BY_SENSOR_TYPE ERR_NOT_SUPPORTED_BY_FW_VER ERR_NOT_SUPPORTED_BY_FPGA_VER
Remark:	AVS_SetSensitivityMode() is supported by the following detector types: HAMS9201, HAMG9208_512, SU256LSB and SU512LDB. Calling this function for another detector type will result in a return value of -120 (ERR_NOT_SUPPORTED_BY_SENSOR_TYPE). This function requires a firmware function x.30.x.x or later. Calling this function for a spectrometer for which an older firmware version is loaded will result in a return value of -121 (ERR_NOT_SUPPORTED_BY_FW_VER). The detector specific FPGA needs to support the sensitivity selection feature as well. The table below shows the minimum required version for the 3 detector types. Calling AVS_SetSensitivityMode() for a spectrometer for which an older FPGA version is loaded will result in a return value of -122 (ERR_NOT_SUPPORTED_BY_FPGA_VER). The table below also lists the Default Mode for each detector type. This is the mode in which the detector operates if the function AVS_SetSensitivityMode() is not called. The default mode is also the mode that is used in models with older firmware and FPGA versions. Note that irradiance calibrated systems are calibrated in the default mode. Changing the sensitivity mode for an irradiance and/or nonlinearity calibrated system requires a recalibration of the system.	

Spectrometer	Detector Type	FPGA version	Default Mode
AvaSpec-NIR256-1.7, AvaSpec-NIR256-2.0TEC, AvaSpec-NIR256-2.5TEC	SENS_HAMS9201	x.13.x.x	Low Noise
AvaSpec-NIR512-2.5-HSC	SENS_HAMG9208_512	x.x.x.x	Low Noise
AvaSpec-NIR256-1.7TEC, AvaSpec-NIR256-2.2TEC	SENS_SU256LSB	x.5.x.x	High Sensitivity
AvaSpec-NIR512-1.7TEC AvaSpec-NIR512-2.2TEC	SENS_SU512LDB	x.4.x.x	High Sensitivity

2.5.41 AVS_GetIpConfig

Function:	**int AVS_GetIpConfig** (AvsHandle EthernetSettingsType*)	 a_hDevice a_Data
Group:	Blocking data send function.	
Description:	Retrieve IP settings from the spectrometer.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	EthernetSettingsType:	Pointer to buffer that will be filled with the Ethernet settings data
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_OPERATION_NOT_SUPPORTED
Remark:	Use this function to read the Ethernet settings of the spectrometer and the run-time used IP address, without having to read the complete device configuration structure. This function works only when using the Ethernet interface (AS7010 platforms). With other interfaces and platforms (AS-MINI, AS5216, etc.), ERR_OPERATION_NOT_SUPPORTED error will be returned.	

2.5.42 AVS_SuppressStrayLight

Function:	**int AVS_SuppressStrayLight** (AvsHandle float double* double*)	 a_hDevice a_MultiFactor, a_pSrcSpectrum, a_pDestSpectrum
Group:	Internal data read function.	
Description:	Returns the stray light corrected pixel values of a dark corrected measurement. Can be called by the application if a new spectrum is received (AVS_GetScopeData()) and a dark spectrum has been subtracted. The Library does not check the allocated buffer size! Please refer to the section under EEPROM structure (section 3.88) for more details.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_Multifactor:	Multiplication factor in stray light algorithm
	a_pSrcSpectrum:	Array of doubles (scope minus dark), with array size equal to maximum number of detector pixels
	a_pDestSpectrum:	Array of doubles stray light suppressed, with array size equal to maximum number of detector pixels
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_MEAS_DATA (no measurement data received) ERR_SL_CALIBRATION_NOT_AVAILABLE ERR_SL_STARTPIXEL_NOT_IN_RANGE ERR_SL_ENDPIXEL_NOT_IN_RANGE ERR_SL_STARTPIX_GT_ENDPIX ERR_SL_MFACTOR_OUT_OF_RANGE
Remark:	In the Qt5_demo_SLS sample program, this function is called to demonstrate SLS In section 3.8.4, more detailed information about using this function can be found	

2.5.43 AVS_ResetDevice

Function:	**int AVS_ResetDevice** (AvsHandle a_hDevice)	
Group:	Blocking control function.	
Description:	Performs a hard reset on the given spectrometer.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_COMMUNICATION
Remark:	This function works only with the AS7010 and AS-MINI platforms. During reset of the spectrometer, all spectrometer HW modules (microprocessor and USB controller) will be reset at once. The spectrometer will start its reset procedure right after sending the command response back to the host.	

2.5.44 AVS_Heartbeat

Function:	**int AVS_Heartbeat** (AvsHandle a_hDevice HeartbeatReqType* a_pHbReq HeartbeatRespType* a_pHbResp)	
Group:	Blocking control function.	
Description:	The Heartbeat message is used for the following purposes; ▪ Spectrometer alive check. With a Heartbeat message sent to the device, the device will indicate that it is alive through the response a_pHbResp. ▪ Retrieving BIT information from a particular spectrometer. ▪ Managing special network monitoring (heartbeat) functions on the spectrometer. Please note that not all BIT fields are used by available spectrometer platforms (AS7010, AS5216, AS-MINI). Please refer to Table 3 and Table 5 for more information. Regardless of any particular Heartbeat function addressed, when Heartbeat message is sent to the Spectrometer it will respond with internal spectrometer information (a BIT matrix). On the application level there is no requirement to send the Heartbeat message at a regular interval when Heartbeat functions are enabled. The Library uses the Heartbeat message internally to monitor and guard the Ethernet connection.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pHbReq:	Bitmapped Heartbeat request values (input), used to control Heartbeat functions of the Spectrometer. Table 2 gives an overview of the HeartbeatReqType data elements defined.
	a_pHbResp:	Heartbeat response structure (output), as received from the spectrometer. Table 4 gives an overview of the HeartbeatRespType data elements defined.
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_COMMUNICATION ERR_INVALID_PARAMETER

www.avantes.com info@avantes.com

Remark:	This function applies only to the AS7010 and AS-MINI platforms. However not all Heartbeat Control fields apply to these platforms. Please refer to Table 3 and Table 5 for the matrix table of the used bitfields per platform. The Library uses also the AVS_Heartbeat() message internally to monitor the Ethernet connection. However the application may override the internal Heartbeat functions with AVS_Heartbeat() by itself. For example; the application can stop the Ethernet auto-recovery function with EAR=0, or it can override the EAR function parameters HBI and HBMW for its own purpose.

The Request Data element used with AVS_Heartbeat() function is given below;

<table>
<tr><th>Parameter</th><th>Abbr</th><th>bit7 MSb</th><th>bit6</th><th>bit5</th><th>bit4</th><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0 LSb</th><th>Byte Count</th></tr>
<tr><td rowspan="4">Heartbeat Control</td><td rowspan="4">HBC</td><td colspan="6">0x00</td><td>EAR</td><td>HBCE</td><td>1</td></tr>
<tr><td colspan="8">0x00</td><td>2</td></tr>
<tr><td colspan="4">HBI</td><td colspan="4">0x00</td><td>3</td></tr>
<tr><td colspan="8">HBMW</td><td>4</td></tr>
</table>

Table 2 HeartbeatReqType type data elements.

HBC Heartbeat Control: bit-word used to control Heartbeat functions of the spectrometer. The following bit words are defined;

- HBCE: Heartbeat Control Enable; this control bit will enable all other control bits of the heartbeat control (HBC) parameter. With HBCE the Heartbeat message can be used for;
  - HBCE=0: Just heartbeat. All other bits of Heartbeat Control (HBC) will be ignored by the Spectrometer. The Heartbeat message is then only used to trigger the Spectrometer.
  - HBCE=1: Set heartbeat controls; all HBC bit field settings will be handled by the Spectrometer.
- EAR: Ethernet Auto-Recovery function control (see also section 2.4); control bit used to enable or disable the Ethernet connection auto-recovery function. The Library will manage the auto-recovery function internally when enabled. EAR control bit is defined as follows;
  - EAR=0: Disable Ethernet Auto Recovery function
  - EAR=1: Enable Ethernet Auto Recovery function
- HBI: Heartbeat Interval; The heartbeat interval time in seconds. Although this value is sent to the spectrometer it is only used by the Library internally. Since the Library will send the Heartbeat message to the spectrometer. The Library will receive this value from the spectrometer internally and take the HBI value as time interval to send the internal heartbeat messages to the spectrometer. HBI represents a 4-bit value as follows;
  - HBI=0: Let the spectrometer use its default HBI value, which is usually 1 second.
  - HBI>0: Interval time in seconds.
- HBMW: Heartbeat Maximum Window; maximum interval window (time in seconds) in which the Heartbeat message must be sent at least one time to the spectrometer. If no Heartbeat message is received by the spectrometer within the time window defined by HBMW, the spectrometer will start the recovery sequence of the enabled heartbeat functions. E.g. in case EAR is enabled, the spectrometer will start the Ethernet recovery sequence as explained above. HBMW represents an 8-bit unsigned value as defined below;
  - HBMW=0: Not allowed, this will result in an ERR_INVALID_PARAMETER error message.
  - HBMW>0: A valid time-out value in seconds.

ID: 020381

Table 3 provides an overview of the supported Heartbeat Request functions associated with the spectrometer platforms (✓=Supported by the platform, ✗=Not supported by the platform).

HBC function	AS7010	AS-MINI	AS5216
HBCE	✓	✗	✗
EAR	✓	✗	✗
HBI	✓	✗	✗
HBMW	✓	✗	✗

Table 3 Matrix table request functions vs. spectrometer platforms.

The Response Data element used with AVS_Heartbeat() function is given below;

<table>
<tr><th>Parameter</th><th>Abbr</th><th>bit7
MSb</th><th>bit6</th><th>bit5</th><th>bit4</th><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0
LSb</th><th>Byte
Count</th></tr>
<tr><td rowspan="4">BIT Matrix</td><td rowspan="4">BIT</td><td>SBME</td><td>SBO</td><td colspan="2">UCT</td><td>ST</td><td>UMF</td><td colspan="2">ADC</td><td>1</td></tr>
<tr><td>DCS</td><td>STO</td><td>STI</td><td colspan="2">0x00</td><td>SCIS</td><td>EAR</td><td>DMAE</td><td>2</td></tr>
<tr><td colspan="4">HBI</td><td colspan="2">0x00</td><td>SCS</td><td>DCS</td><td>3</td></tr>
<tr><td colspan="8">HBMW</td><td>4</td></tr>
<tr><td rowspan="4">Reserved</td><td rowspan="4">-</td><td colspan="8">0x00</td><td>5</td></tr>
<tr><td colspan="8">0x00</td><td>6</td></tr>
<tr><td colspan="8">0x00</td><td>7</td></tr>
<tr><td colspan="8">0x00</td><td>8</td></tr>
</table>

Table 4 HeartbeatRespType data elements.

BIT Built-In Test matrix: bit-word indicating faulty components or spectrometer internal HW and SW related status. The following bit words are defined;

ADC: Analog to Digital Converter type, indicates the status of the AS7010 ADC circuits on board as follows;
ADC[0]: 0=Single ended ADC is disabled, 1=Single ended ADC is enabled
ADC[1]: 0=Differential ADC is disabled, 1= Differential ADC is enabled

UMF: USB Monitoring Failure, status of the internal communication bus with the USB controller;
UMF=0: No error
UMF=1: Error, internal communication with the USB controller is not possible, not able to detect USB status

ST: Sensor Type, Spectrometer detector type does not match with the used firmware/FPGA, no measurement possible;
ST=0: No error, programmed sensor type matches the detector type
ST=1: Error, programmed sensor type is wrong, no measurement possible

UCT: USB Connection Type, indicates the type of USB connection currently in use by Spectrometer. Following USB connections are possible;
0x00: USB 2 type
0x01: USB 3 type
0x02: USB not connected

SBO: Scan Buffer Overflow status;
SBO=0: No error
SBO=1: Error, one or more scans are missed due to internal scan buffer overflow
At each start scan sequence SBO will be reset.

SBME: Scan Buffer Mutex Error status;
SBME=0: No error
SBME=1: Error, one or more scans are missed due to internal scan buffer mutex error
At each start scan sequence SBME will be reset.

DMAE: DMA Error status;
DMAE=0: No error
DMAE=1: Error, one or more scans are missed due to internal DMA error
At each start scan sequence DMAE field will be reset.

EAR: Ethernet Auto-Recovery function status (see also section 2.4);

EAR=0: Ethernet Auto Recovery on Spectrometer disabled, no Ethernet connection management (auto-recovery) running
EAR=1: Ethernet Auto Recovery on Spectrometer enabled.

SCIS: Spectrometer Control Interface (SCI) Status; SCI is the name of TCP connection in between the Library and the Spectrometer. SCIS indicates the connection status of the Ethernet SCI interface as follows;
SCIS=0: SCI TCP connection not established, no communication through SCI possible with the host.
SCIS=1: SCI TCP connection established.

STI Spurious Trigger Idle error; external trigger detected when no measurement sequence was started or measurement was running. STI values are;
STI=0: No trigger error.
STI=1: External trigger detected while no measurement request is pending in hardware trigger mode, since this type of error is not given when Single Scan trigger or Hardware trigger is enabled (see section 2.6).
At each start scan sequence STI will be reset.

STO Spurious Trigger Overflow error; external trigger was detected during a running measurement. This type of error is relevant only with trigger type *Single Scan Trigger* (SST). See section 2.6 for the trigger types. STO values are;
STO=0: No trigger overflow error (or SST not enabled).
STO=1: External trigger detected while a measurement was running.
At each start scan sequence STO will be reset.

DCS Device Configuration Status; 2 bit value indicating the device configuration settings currently being used. DCS values are;
DCS=0x00: User-specific settings loaded. User-specific device configuration is set by using the AVS_SetParameter() function (see section 2.5.29).
DCS=0x01: Factory device settings loaded. The spectrometer uses the factory settings in case of;
a) User-specific device configuration is never been set before, or
b) Spectrometer was not able to load the user-specific device settings due to internal read or write errors. In this case the factory device settings are loaded from internal flash memory.
DCS=0x02: Factory settings corrupted due to internal errors. Device settings retrieved through AVS_GetParameter() are insignificant and hence not suitable for use. A hard reset or resetting factory settings, through AVS_ResetParameter() as described in section 2.5.31, may help solve this issue.

SCS Secure Configuration Status; indicates the status of the secure settings currently being applied by the spectrometer. Secure settings comprise special read-only factory settings of the device. It contains for example the serial ID of the spectrometer and various parameters which are used by the spectrometer internally for its correct operation. SCS values are;
SCS=0: Secure settings loaded (is valid) and is being used.
SCS=1: Secure settings corrupted due to internal errors. In this case the spectrometer will not operate properly. A hard reset may help solve this issue.

HBI: Heartbeat Interval; 4-bit value which represents the interval time in seconds;
HBI>0: Interval time in seconds.

HBMW: Heartbeat Maximum Window value; HBMW value as set with AVS_Heartbeat() message. HBMW represents an 8-bit unsigned value as defined below;
HBMW=0: EAR function not enabled.
HBMW>0: Time window in seconds, in which at least one Heartbeat message must be received by the spectrometer, which is managed internally by the Library.

Table 5 provides an overview of the available Heartbeat status information associated with the spectrometer platforms (✓=Supported by the platform, ✘=Not supported by the platform).

BIT Field	AS7010	AS-MINI	AS5216
ADC	✓	✘	✘
UMF	✓	✘	✘
ST	✓	✘	✘
UCT	✓	✘	✘
SBO	✓	✘	✘
SBME	✓	✘	✘
DMAE	✓	✘	✘
EAR	✓	✘	✘
SCIS	✓	✘	✘
STI	✓	✘	✘
STO	✓	✘	✘
DCS	✓	✓	✘
SCS	✓	✓	✘
HBI	✓	✘	✘
HBMW	✓	✘	✘

Table 5 Matrix table heartbeat BIT status vs. spectrometer platforms.

2.5.45 AVS_EnableLogging

Function:	**bool AVS_EnableLogging** (Bool a_EnableLogging)
Group:	Blocking control function.
Description:	Enables or disables writing debug information to a log file.
Parameters:	a_EnableLogging: Set to true when logging should be enabled, false otherwise.
Return:	TRUE
Remark:	When logging is enabled, debug information will be written to a file called “avaspec.dll.log”. The log file is located in the user folder (c:\Users\[username]).

2.5.46 AVS_GetDeviceType

Function:	**int AVS_GetDeviceType** (AvsHandle a_hDevice AVSDeviceType* a_pDeviceType)	
Group:	Blocking read function.	
Description:	Returns a value indicating the type of spectrometer used (5216, Mini or 7010)	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_pDeviceType	A number indicating the type of spectrometer. (See AVSDeviceType enum)
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_TYPE ERR_DLL_INITIALISATION ERR_OPERATION_NOT_SUPPORTED ERR_SECURE_CFG_NOT_READ

2.5.47 AVS_SetDstrStatus

In the Linux/macOS version of this library both versions of this function perform the same action. In the Windows DLL version the “setdstrstatus” function with “callback” is available by calling AVS_SetDstrStatusCallback() while AVS_SetDstrStatus() uses the Windows messaging system.

Function:	**int AVS_SetDstrStatus** (AvsHandle a_hDevice void* a_hWnd)	
Group:	Blocking control function.	
Description:	Used to set the address of the window the DSTR (Dynamic StoreToRam) status message is sent to when the DSTR status has changed.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb()
	a_hWnd	Windows handle to notify the application that the Dynamic StoreToRam status has changed. The Library sends a message with the identifier WM_DSTR_STATUS to the window with handle a_hWnd.
Return:	ERR_SUCCESS	
Remark:	When logging is enabled, debug information will be written to a file called “avaspec.dll.log”. The log file is located in the user folder (c:\Users\[username]).	

2.5.48 AVS_SetDstrStatusCallback

Function:	**int AVS_SetDstrStatusCallback**	
	(	
	AvsHandle	a_hDevice
	void	(*__Dstr)(AvsHandle*, unsigned int)
	)	
Group:	Blocking control function.	
Description:	Used to set the address of the callback function that is called when the DSTR (Dynamic StoreToRam) status has changed.	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb().
	(*__Dstr)(AvsHandle *, unsigned int)	Pointer to a Callback function to notify Dynamic StoreToRam status change.
Return:	ERR_SUCCESS	
Remark:	When logging is enabled, debug information will be written to a file called “avaspec.dll.log”. The log file is located in the user folder (c:\Users\[username]).	

2.5.49 AVS_GetDstrStatus

Function:	**int AVS_GetDstrStatus**	
	(	
	AvsHandle	a_hDevice
	DstrStatusType*	a_pDstrStatus
	)	
Group:	Blocking control function.	
Description:	Reads the DSTR (Dynamic StoreToRam) status received from the spectrometer. This function should be called immediately after the DSTR callback function is invoked (see paragraph 2.5.48).	
Parameters:	a_hDevice:	Device identifier returned by AVS_Activate() or AVS_ActivateConnCb().
	a_pDstrStatus	Pointer to the DSTR status context.
Return:	ERR_SUCCESS	
Remark:	When logging is enabled, debug information will be written to a file called “avaspec.dll.log”. The log file is located in the user folder (c:\Users\[username]).	

2.6 Data Elements

Several data-types used by the Library and necessary for the application interface are given below.

Note: To match the structures that are used in the AvaSpec firmware the structures mentioned here have to be compiled with *byte alignment*.

Type	Format	Value/Range	Description
bool	8 bits value	0 - 1	False - True
char	8 bits value	-128 <= x <= 127	Signed character
unsigned char	8 bits value	0 <= x <= 255	Unsigned character
short	16 bits value	-32768 <= x <= 32767	Signed integer
unsigned short	16 bits value	0 <= x <= 65535	Unsigned integer
int	32 bits value	2,147,483,648 <= x <= 2,147,483,647	Signed integer
unsigned int	32 bits value	0 <= x <= 4294967295	Unsigned integer
float	32 bits value		Floating point number (7 digits precision)
double	64 bits value		Double sized floating point number (15 digits precision)
HWND	32 bits value		Windows typedef for window identification, HWND is used for Windows API calls that require a Window handle.

AvsDeviceType	enum
	{
	TYPE_UNKNOWN,
	TYPE_AS5216,
	TYPE_ASMINI,
	TYPE_AS7010
	}
		
		0
		1
		2
		3
			See AVS_GetDeviceType

AvsIdentityType	struct
	{
	char SerialNumber[10],
	char UserFriendlyName[64],
	DeviceStatus Status
	}
			
			Serial identification number, null terminated 8-bit ASCII string
			User friendly name to be defined by application
			Device status
			(Size = 75 bytes)

Type	Format	Value/Range	Description

BroadcastAnswer Type	struct
	{
	unsigned char InterfaceType,
	unsigned char serial[10],
	unsigned short port,
	unsigned char status,
	unsigned int RemoteHostIp,
	unsigned int LocalIp,
	unsigned char reserved[4]
	}
			
			Shows type of device that is answering
			Serial identification number, null terminated 8-bit ASCII string
			TCP port used in communications
			DeviceStatus
			IP address of computer connected to spectrometer
			IP address of spectrometer
			reserved for future expansion
			(Size = 26 bytes)

ControlSettings Type	struct
	{
	unsigned short m_StrobeControl,
	
	unsigned int m_LaserDelay,
	unsigned int m_LaserWidth,
	
	float m_LaserWaveLength
	unsigned short m_StoreToRam,
	
	}
		
		0 – 0xFFFF
		
		0 – 0xFFFFFFFF
		0 – 0xFFFF
		
		0 – 0xFFFF
			
			Number of strobe pulses during integration period (high time of pulse is 1 ms),
			(0 = no strobe pulses)
			Laser delay since trigger, unit is internal FPGA clock cycle
			Laser pulse width , unit is internal FPGA clock cycle,
			(0 = no laser pulse)
			Peak wavelength of laser (nm), used for Raman Spectroscopy
			0 = no storage to RAM
			> 0 = number of spectra to be stored
			(Size = 16 bytes)

DarkCorrection Type	struct
	{
	unsigned char m_Enable;
	unsigned char m_ForgetPercentage;
	
	}
		
		0 – 1
		0 - 100
			
			Disable – Enable dynamic dark correction (sensor dependent)
			Percentage of the new dark value pixels that has to be used. e.g., a percentage of 100 means only new dark values are used. A percentage of 10 means that 10 percent of the new dark values is used and 90 percent of the old values is used for drift correction
			(Size = 2 bytes)

Type	Format	Value/Range	Description

DeviceConfig Type	struct
	{
	unsigned short m_Len;
	unsigned short m_ConfigVersion;
	char m_aUserFriendlyId[64];
	DetectorType m_Detector;
	IrradianceType m_Irradiance;
	SpectrumCalibrationType m_Reflectance;
	SpectrumCorrectionType m_SpectrumCorrect;
	StandaloneType m_StandAlone;
	DynamicStorageType m_DynamicStorage;
	TempSensorType m_Temperature[3];
	TecControlType m_TecControl;
	ProcessControlType m_ProcessControl;
	EthernetSettingsType m_EthernetSettings;
	unsigned char m_aReserved[9720];
	OemDataType m_OemData;
	}
		
		0 – 0xFFFF
			Configuration data structure:
			
			Size of this structure in bytes
			Version of this structure
			User friendly identification string
			Sensor/detector related parameters
			Intensity calibration parameters
			Reflectance calibration parameters
			Correction parameters
			Stand-alone related parameters (e.g. measure mode, control)
			Dynamic storage parameters
			Calibration parameters of three temperature sensors
			TecControl parameters
			ProcessControl parameters
			EthernetSettings parameters
			Reserved field for future use
			OEM specific data field which can be used for own purpose
			(Size = 63484)

DeviceStatus	enum
	{
	UNKNOWN,
	USB_AVAILABLE,
	USB_IN_USE_BY_APPLICATION,
	USB_IN_USE_BY_OTHER,
	ETH_AVAILABLE,
	ETH_IN_USE_BY_APPLICATION,
	ETH_IN_USE_BY_OTHER,
	ETH_ALREADY_IN_USE_USB
	}
		
		0
		1
		2
		3
		4
		5
		6
		7
			
			Initial state
			Device connected by USB and not in use
			Device connected by USB and in use by caller
			Device connected by USB and in use by other application
			Device connected by ETH and not in use
			Device connected by ETH and in use by caller
			Device connected by ETH and in use by other application
			Device is already in use, connected by USB

ID: 020381

Type	Format	Value/Range	Description

DetectorType	struct
	{
	SensorType m_SensorType,
	unsigned short m_NrPixels,
	float m_aFit[5],
	bool m_NLEnable,
	double m_aNLCorrect[8],
	double m_aLowNLCounts,
	double m_aHighNLCounts,
	float m_Gain[2],
	
	float m_Reserved,
	float m_Offset[2],
	
	float m_ExtOffset,
	unsigned short m_DefectivePixels[30],
	}
		
		0 – 4096
		
		1.0 – 6.0
		
		-0.30 - +0.30
		
		0.0 – 2.0
			Sensor configuration structure:
			
			Sensor identification
			Number of pixels of sensor
			Polynomial coefficients needed to determine wavelength
			Enable/disable nonlinearity correction
			Polynomial coefficients needed for non-linearity correction
			Lower counts limit for non-linearity correction
			Higher counts limit for non-linearity correction
			Gain correction for spectrometer ADC (range is divided in 64 steps)
			Not used
			Offset correction for spectrometer ADC in Volt (range is divided in 512 steps)
			Offset to match the detector output range with the ADC range
			Defective pixel numbers
			
			(Size = 188 bytes)

Dynamic StorageType	struct
	{
	int32 m_Nmsr,
	uint8 m_Reserved[8]
	}
			
			Number of measurements (future use)
			For future use and backwards compatibility
			(Size = 12 bytes)

DstrStatusType	Struct
	{
	uint32 m_TotalScans
	uint32 m_UsedScans
	uint32 m_Flags
	unsigned char m_IsStopEvent
	
	unsigned char m_IsOverflowEvent
	unsigned char m_IsInternalErrorEvent
	
	unsigned char m_Reserved
	}
			
			Number of scans which fit in the FIFO
			Number of scans available in the FIFO, ready to be sent
			Bitmapped flags / errors (as described below)
			m_Flags:bit<0> 1 = Measurement stopped due to STOP received or measurement ready, 0 otherwise.
			m_Flags:bit<1> 1 = FIFO overflow error occurred, 0 otherwise.
			m_Flags:bit<2> 1 = DSTR measurement has stopped due to an internal error, 0 otherwise.
			Padding byte (reserved for future use).
			(Size = 16 bytes)

Type	Format		Value/Range	Description
Ethernet SettingsType	struct {			
	unsigned int	m_IpAddr;	0 – 0xFFFFFFFFFF	Static IP Address (when not using a DHCP server)
	unsigned int	m_NetMask;		Net Mask value (e.g. 255.255.255.0)
	unsigned int	m_Gateway;		Default gateway value (e.g. 192.168.1.254)
	unsigned char	m_DhcpEnabled;		0 = Static IP Address used, 1=DHCP enabled
	unsigned short	m_TcpPort;		Default values is 4500, used to connect to spectrometer
	unsigned char	m_LinkStatus;		Reserved
	unsigned char	m_ClientIdType;		Type of the 'DHCP Client Identifier'; 0 = DHCP Client Identifier disabled. 1 = Board MAC address is used by the DHCP client for option 61. 2 = Board serial number is used by the DHCP client for option 61. 3 = A fixed identifier is used by the DHCP client for option 61. The identifier text is given with the m_ClientIdCustom field.
	char	m_ClientIdCustom[32];		Identifier text field contains a null terminated string with maximum length of 31 bytes (excluding the terminating zero). Minimum length is 1 (excluding the terminating zero). m_ClientIdCustom is used with m_ClientIdType=2 only. In case of other types, the m_ClientIdCustom field is ignored by the spectrometer. This Client Identifier is used along with DHCP option 61 by the DHCP client, running on the spectrometer (AS7010 only).
	unsigned char	m_Reserved[79];		Reserved.
	}			(Size = 128 bytes)
InterfaceType	enum {			
	RS232,		0	Used to tell the different AvaSpec models apart, e.g. in the Broadcast answer
	USB5216,		1	
	USBMINI,		2	
	USB7010,		3	
	ETH7010		4	
	}			

Type	Format	Value/Range	Description

IrradianceType	struct
	{
	SpectrumCalibrationType m_IntensityCalib,
	unsigned char m_CalibrationType,
	
	unsigned int m_FiberDiameter,
	}
			
			Setting during intensity calibration
			0 = Bare fiber, No Stray Light Correction used
			1 = Diffuser, No Stray Light Correction used
			10 = Bare Fiber, Stray Light Correction was used
			11 = Diffuser, Stray Light Correction was used
			Fiber diameter during intensity calibration
			(Size = 16391+1+4 = 16396 bytes)

MeasConfig Type	struct
	{
	unsigned short m_StartPixel,
	unsigned short m_StopPixel,
	float m_IntegrationTime,
	unsigned int m_IntegrationDelay,
	
	unsigned int m_NrAverages,
	DarkCorrectionType m_CorDynDark,
	SmoothingType m_Smoothing,
	unsigned char m_SaturationDetection,
	
	TriggerType m_Trigger,
	ControlSettingsType m_Control,
	}
		
		0-4095
		0–4095
		0.002 – 600000
		0 – 0xFFFFFFFF
		
		1 – 0xFFFFFFFF
		
		0 – 2
			
			First pixel to be sent to PC
			Last pixel to be sent to PC
			Integration time in ms
			Integration delay, unit is internal FPGA clock cycle
			(0 = one unit before laser start)
			Number of averages in a single measurement
			Dynamic dark correction parameters
			Smoothing parameters
			0 = disabled,
			1 = enabled, determines during each measurement if pixels are saturated (ADC value = 2^16 –1)
			2 = enabled, and also corrects inverted pixels (only ILX554)
			Trigger parameters
			Control parameters
			(Size = 41 bytes)

OemDataType	struct
	{
	unsigned char data[4096]
	}
			
			Data field reserved for external applications own usage
			(Size = 4096 bytes)

ProcessControl Type	struct
	{
	float m_AnalogLow[2]
	float m_AnalogHigh[2]
	float m_DigitalLow[10]
	float m_DigitalHigh[10]
	}
			
			Settings that can be used for the 2 analog and 10 digital output signals at the DB26 connector. The analog settings can be used to define a function output range that should correspond to the 0-5V range of the analog output signals. The digital output settings can be used as lower- and upper thresholds.
			
			(Size = 96 bytes)

Type	Format	Value/Range	Description
SensorType	unsigned char	0 – 0x18	0x00 = Reserved 0x01 = Hams8378-256 0x02 = Hams8378-1024 0x03 = ILX554 0x04 = Hams9201 0x05 = Toshiba TCD1304 0x06 = TSL1301 0x07 = TSL1401 0x08 = Hams8378-512 0x09 = Hams9840 0x0A = ILX511 0x0B = Hams10420_11850 0x0C = Hams11071-2048x64 0x0D = Hams7031_11501 0x0E = Hams7031-1024x58 0x0F = Hams11071-2048x16 0x10 = Hams11155 0x11 = SU256LSB 0x12 = SU512LDB 0x13 = reserved 0x14 = reserved 0x15 = HAMS11638 0x16 = HAMS11639 0x17 = HAMS12443 0x18 = HAMG9208_512 0x19 = HAMG13913 0x1A = HAMS13496

Smoothing Type	struct
	{
	unsigned short m_SmoothPix,
	
	unsigned char m_SmoothModel
	}
		
		0 – 2048
		
		0
			
			Number of neighbor pixels used for smoothing, max. has to be smaller than half the selected pixel range because both the pixels on the left and on the right are used
			Only one model defined so far
			(Size = 3 bytes)

Type	Format		Value/Range	Description

Spectrum Calibration Type	struct
	{
	SmoothingType
	float
	float
	}
		
		m_Smoothing,
		m_CalInttime,
		m_aCalibConvers[4096]
			
			0.002 – 600000
				
				Smoothing parameter during calibration
				Integration time during calibration (ms)
				Conversion table from Scopedata to calibrated data
				(Size = 16391 bytes)

Spectrum Correction Type	struct
	{
	float
	}
		
		m_aSpectrumCorrect[4096]
				
				Correct pixel values, e.g. for PRNU
				(Size = 16384 bytes)

Standalone Type	struct
	{
	bool
	MeasConfigType
	signed short
	}
		
		m_Enable,
		m_Meas,
		m_Nmsr
				
				(Size = 44 bytes)

TecControl Type	struct
	{
	bool
	float
	float
	}
		
		m_Enable,
		m_Setpoint,
		m_aFit[2]
				Tec Control parameters
				
				Set to True if device supports TE Cooling
				SetPoint for detector temperature in degr. Celsius
				DAC polynomial
				(Size = 13 bytes)

TempSensor Type	struct
	{
	float
	}
		
		m_aFit[5]
				
				Calibration coefficients temperature sensor
				(Size = 20 bytes)

TimeStamp Type	struct
	{
	unsigned short
	
	unsigned short
	
	}
		
		m_Date,
		
		m_Time
				
				bit 0..4 (day, 0 – 31)
				bit 5..8 (month, 1 – 12)
				bit 9..15 (years since 1980, 0 – 119)
				bit 0..4 (2-second unit, 0 - 30)
				bit 5..10 (minutes, 0 - 59)
				bit 11..15(hours, 0 – 23)
				(Size = 4 bytes)

Type	Format	Value/Range	Description

TriggerType	struct
	{
	unsigned char m_Mode,
	unsigned char m_Source,
	unsigned char m_SourceType
	}
		
		0 – 2
		0 – 1
		0 – 1
			Trigger parameters
			
			mode, (0 = Software, 1 = Hardware, 2 = Single Scan)
			trigger source, (0 = external trigger, 1 = sync input)
			source type, (0 = edge trigger, 1 = level trigger)
			See section 3.4.8 for detailed information on trigger types.
			(Size = 3 bytes)

Table 6 API data elements.

2.6.1 Return Value Constants

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

Return code	Value	Description
ERR_SUCCESS	0	Operation succeeded
ERR_INVALID_PARAMETER	-1	Function called with invalid parameter value.
ERR_OPERATION_NOT_SUPPORTED	-2	e.g. Function called to use 16bit ADC mode, with 14bit ADC hardware
ERR_DEVICE_NOT_FOUND	-3	Opening communication failed or time-out during communication occurred.
ERR_INVALID_DEVICE_ID	-4	AvsHandle is unknown in the Library
ERR_OPERATION_PENDING	-5	Function is called while result of previous call to AVS_Measure() is not received yet
ERR_TIMEOUT	-6	No answer received from device
Reserved	-7	-
ERR_INVALID_MEAS_DATA	-8	No measurement data is received at the point AVS_GetScopeData() is called
ERR_INVALID_SIZE	-9	Allocated buffer size too small
ERR_INVALID_PIXEL_RANGE	-10	Measurement preparation failed because pixel range is invalid
ERR_INVALID_INT_TIME	-11	Measurement preparation failed because integration time is invalid (for selected sensor)
ERR_INVALID_COMBINATION	-12	Measurement preparation failed because of an invalid combination of parameters, e.g. integration time of (600000) and (Navg > 5000)
Reserved	-13	-
ERR_NO_MEAS_BUFFER_AVAIL	-14	Measurement preparation failed because no measurement buffers available
ERR_UNKNOWN	-15	Unknown error reason received from spectrometer
ERR_COMMUNICATION	-16	Error in communication or Ethernet connection failure
ERR_NO_SPECTRA_IN_RAM	-17	No more spectra available in RAM, all read or measurement not started yet
ERR_INVALID_DLL_VERSION	-18	Library version information could not be retrieved
ERR_NO_MEMORY	-19	Memory allocation error in the Library
ERR_DLL_INITIALISATION	-20	Function called before AVS_Init() is called
ERR_INVALID_STATE	-21	Function failed because AvaSpec is in wrong state (e.g. AVS_Measure() without calling AVS_PrepareMeasurement() first)
ERR_INVALID_REPLY	-22	Reply is not a recognized protocol message
Reserved	-23	-
ERR_ACCESS	-24	Error occurred while opening a bus device on the host. E.g. USB device access denied due to user rights
ERR_INTERNAL_READ	-25	A read error has occurred. Spectrometer has failed when reading, for example, the Device Configuration settings from the internal flash memory or the temperature value from the on-board temperature sensor
ERR_INTERNAL_WRITE	-26	A write error has occurred. Spectrometer has failed when writing, for example, the Device Configuration settings into the internal flash memory
ERR_ETHCONN_REUSE	-27	Library could not be initialized due to an Ethernet

Return code	Value	Description
		connection initialization error which is caused by the presence of another Library instance running on the same machine. It also could have been caused by calling the AVS_Init() function too quickly after calling AVS_Done(). In case of ERR_ETHCONN_REUSE, AVS_Init() can be invoked again to retry the initialization.
ERR_INVALID_DEVICE_TYPE	-28	The device-type information stored in the spectrometer isn't recognized as one of the known device types
ERR_SECURE_CFG_NOT_READ	-29	The AVS_GetDeviceType function is used, but the secure config (holding the device type information) hasn't been read yet. Most likely the device isn't initialised correctly.
ERR_UNEXPECTED_MEAS_RESPONSE	-30	Unexpected response from spectrometer while getting measurement data. (Most likely, the measurement was stopped.)
ERR_INVALID_PARAMETER_NR_PIXEL	-100	NrOfPixel in Device data incorrect
ERR_INVALID_PARAMETER_ADC_GAIN	-101	Gain Setting out of range
ERR_INVALID_PARAMETER_ADC_OFFSET	-102	Offset Setting out of range
ERR_INVALID_MEASPARAM_AVG_SAT2	-110	Use of Saturation Detection Level 2 is not compatible with the Averaging function
ERR_INVALID_MEASPARAM_AVG_RAM	-111	Use of Averaging is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_SYNC_RAM	-112	Use of the Synchronize setting is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_LEVEL_RAM	-113	Use of Level Triggering is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_SAT2_RAM	-114	Use of Saturation Detection Level 2 Parameter is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_FWVER_RAM	-115	The StoreToRam function is only supported with firmware version 0.20.0.0 or later (AS5216 only).
ERR_INVALID_MEASPARAM_DYNDARK	-116	Dynamic Dark Correction not supported
ERR_NOT_SUPPORTED_BY_SENSOR_TYPE	-120	Use of AVS_SetSensitivityMode() not supported by detector type
ERR_NOT_SUPPORTED_BY_FW_VER	-121	Use of AVS_SetSensitivityMode() not supported by firmware version
ERR_NOT_SUPPORTED_BY_FPGA_VER	-122	Use of AVS_SetSensitivityMode() not supported by FPGA version
ERR_SL_CALIBRATION_NOT_AVAILABLE	-140	Spectrometer was not calibrated for stray light correction
ERR_SL_STARTPIXEL_NOT_IN_RANGE	-141	Incorrect start pixel found in EEPROM
ERR_SL_ENDPIXEL_NOT_IN_RANGE	-142	Incorrect end pixel found in EEPROM
ERR_SL_STARTPIX_GT_ENDPIX	-143	Incorrect start or end pixel found in EEPROM
ERR_SL_MFACTOR_OUT_OF_RANGE	-144	Factor should be in range 0.0 – 4.0

2.6.2 Windows Messages

The following table gives an overview of window messages.

Windows message identifier	WPARM	LPARM	Description
WM_MEAS_READY	0: On success < 0: One of the above error reasons > 0: In StoreToRAM mode	Device handle	After measurement data is available the Library sends this message to the application. The command value used is WM_MEAS_READY and is defined as; ▪ (WM_USER + 1) for the 32-bit version or, ▪ (WM_APP + 1) for the 64-bit version of the Library (avaspecx64.dll)
WM_CONN_STATUS	See Table 1 in section 2.4.1	Device handle	After Ethernet connection status has changed the Library sends this message to the application. The command value used is WM_CONN_STATUS and is defined as; (WM_APP + 15)
WM_DSTR_STATUS	0: On success	Device handle	(WM_APP + 16)
WM_DEVICECHANGE	DBT_DEVNODES_CHANGED(7)	0	After device attachment/removal Windows sends this message to the application.

2.6.3 Callback Function on Measurement Ready

The callback function that indicates measurement data ready takes two parameters, the first is the handle of the spectrometer acquired by AVS_Activate() or AVS_ActivateConnCb(), the second is an integer value representing the value of the callback function.

The following table gives an overview of values for the second integer parameter.

Value	Description
0 (on success)	Measurement data is available.
> 0 (in StoreToRAM mode)	Value is the number of scans actually stored in RAM (which can be smaller than the amount requested)
< 0	The measurement failed. See section 2.6.1 for a description of the error message.

3 Example Source Code

3.1 Examples for Windows

Example source code can be found in the directory tree of the driver. 32-Bit sample programs (including header files and link libraries, where appropriate) are provided for the following programming environments:

- Borland C++ Builder 5.0 (native code)
- Borland Delphi 6.0 (native code)
- Embarcadero C++Builder 2009 (native code)
- Embarcadero Delphi 2009 (native code)
- Java (managed code)
- LabVIEW 8.2, older versions on request (native code)
- MATLAB R2013b (native code)
- Microsoft Visual C++ 2017 using MFC (native code)
- Python 3.6, using PyQt5 and VS2017
- Microsoft Visual C++ 2017 combined with the Qt5 framework (native code)
- Microsoft Visual Basic 2017 (managed code, for .net)
- Microsoft C# 2017 (managed code, for .net)

64-Bit sample programs are provided for the following programming environments:

- Embarcadero Delphi XE3, both simple and comprehensive samples (native code)
- Java (managed code)
- LabVIEW 2009, 64-bit version (native code)
- MATLAB R2013b, 64-bit version (native code)
- Microsoft Visual C++ 2017 using MFC (native code)
- Python 3.6, using PyQt5 and VS2017
- Microsoft Visual C++ 2017 combined with the Qt5 framework (native code)
- Microsoft Visual Basic 2017, (managed code, for .net)
- Microsoft Visual C# 2017, (managed code, for .net)

Besides a comprehensive sample program in the main Delphi folder, some dedicated Delphi sample programs are included in the Delphi subfolders. The multichannel folder demonstrates how multiple spectrometer channels can run simultaneously in *sync* or *async* mode. The simple folder demonstrates a minimal Delphi program.

For VC++2017/Qt5, several samples are provided. The use of the Qt framework requires a more complex installation. Please refer to Appendix C for some guidelines on recompiling the samples. The Qt samples include a comprehensive sample, a simple sample and a sample that demonstrates the stray light suppression function. Note the use of the AVS_MeasureCallback() function in the Qt samples. The reason for this is the fact that the WM_USER+1 Windows message that the 32-bit AvaSpec Library uses to signal new data was also used by Qt4. This means that the regular AVS_Measure() function does not work correctly with Qt4. An alternative would be to use the AVS_PollScan() function. There are no issues with Qt5. There are also no issues with the WM_APP+1 message that the avaspecX64.dll uses.

For LabVIEW, the following sample programs are available:

- A comprehensive program for a single channel AvaSpec-USB2, which also includes subvi's for all functions in the AvaSpec Library (LabViewSingleChan.llb in the LabViewSingleChan folder)
- A simple sample program that uses AVS_PollScan() instead of Windows Messaging (polling.llb in the polling folder)

- A multichannel example program which illustrates how to run multiple spectrometer channels (fixed to 2 channels in the example program) in SYNC mode, as well as ASYNC mode (polling_mc.llb in the polling folder)
- A simple sample program that illustrates how the StoreToRam functionality can be implemented in combination with AVS_PollScan() (polling_StoreToRAM.llb in the polling folder)
- Simple sample programs that demonstrate the use of the AVS_MeasureLV() function, which will generate a custom user event to signal arrival of new data. Demos for one, two and four channels are available (eventdemo3.llb, mc_eventdemo.llb and mc4_eventdemo.llb in the events folder)
- Simple sample programs that demonstrate the use of an intermediate Library that will use Windows messaging to signal the arrival of new data. Demos for one and two channels are available. (messaging2.llb and messaging_mc.llb in the messaging folder)
- Simple sample programs that demonstrate the use of an intermediate Library that will use a custom user event to signal arrival of new data. Demos for one and two channels are available. (intermediate.llb and intermediate_mc.llb in the intermediate folder)

Please refer to the separate manual (LabVIEW support.pdf) for more information.

The sample program in MATLAB is described in a separate manual (MATLAB support.pdf) that can be found in the main folder where the AvaSpec Library package (32 or 64-bit) has been installed. A version of the MEX file for multiple spectrometers can be found in the MATLAB_multichannel subdirectory.

3.2 Examples for Linux or macOS

Sample programs (including header files and link libraries, where appropriate) are provided for the following programming environments:

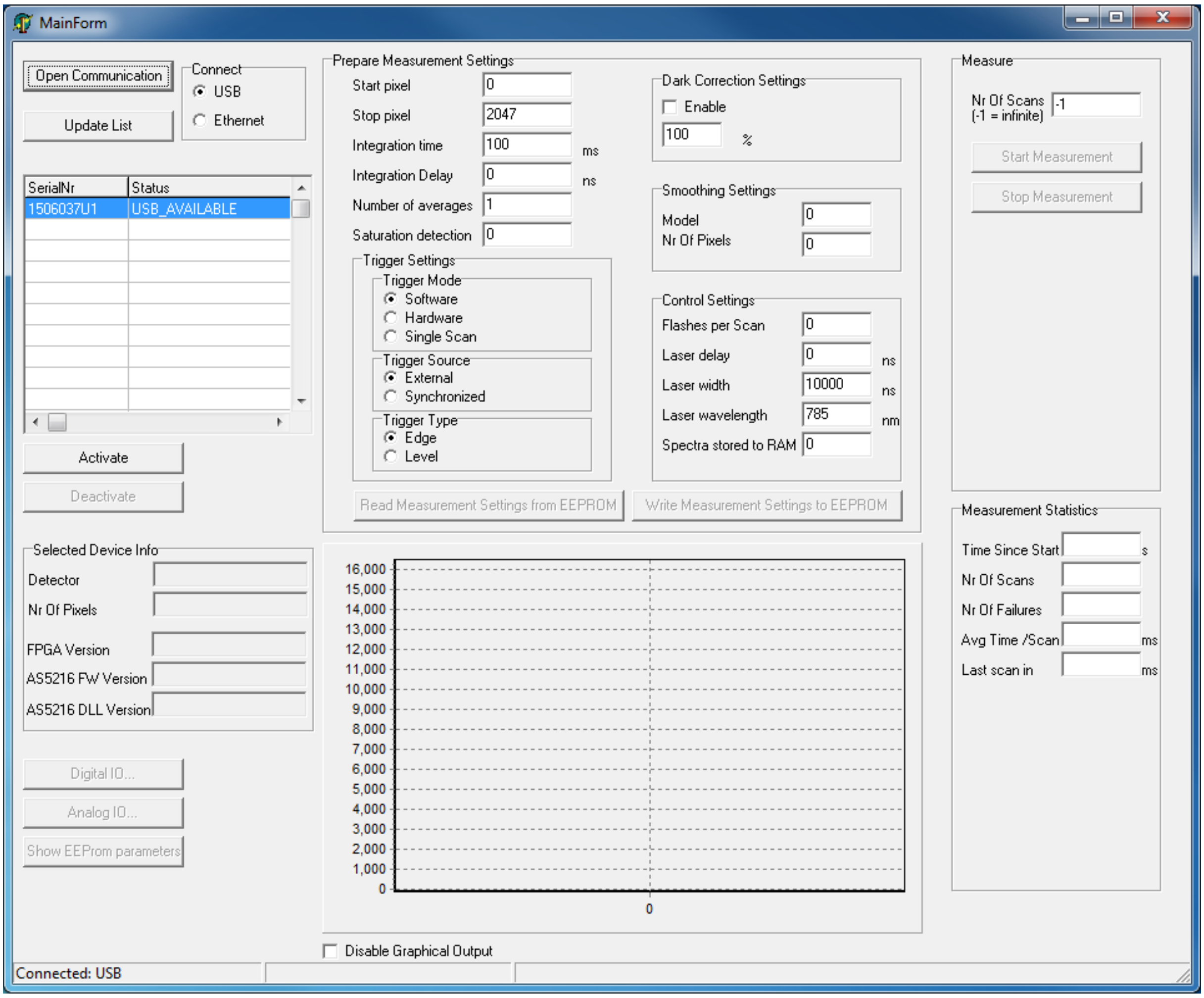
- C++ combined with the Qt5 framework (native code), using QtCreator.
- Python combined with the PyQt5 framework.
- Objective-C (macOS)
- Swift (macOS)

The Qt5 samples include a comprehensive sample, a simple sample, a sample that demonstrates synchronization and a sample that demonstrates the stray light suppression function. Note the use of the AVS_MeasureCallback() function in the samples, which uses a callback mechanism to retrieve scan data. An alternative to callback functions would be the use of the AVS_PollScan() function. Both the Objective-C and the Swift sample are minimal samples.

Please note that some of the screen shots in this section are taken from equivalent Windows DLL sample programs.

3.3 Initialization and Activation of a Spectrometer

After starting one of the full feature example programs (Qt5_demo_full, Delphi (XE3) or C++ Builder folder), the main window will be displayed. By clicking the “Open Communication” button, the AVS_Init() function is called and if successful, the serial number and status for the connected spectrometer(s) is collected (AVS_UpdateUSBDevices() c.q. AVS_GetNrOfDevices() and AVS_GetList()). The result is displayed in the list at the top left of the window, as shown in the figure below.



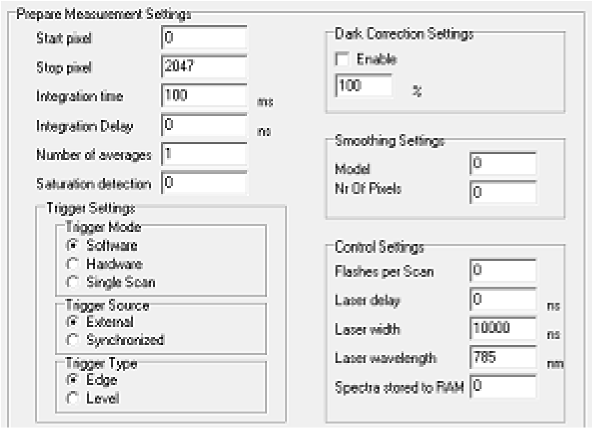
After selecting a spectrometer from the list, clicking the “Activate” button results in a call to the AVS_Activate() function. This function returns a DeviceHandle which needs to be used in further communication between the Library and this device. After a successful call to AVS_Activate(), the status for the selected device will change from “AVAILABLE” to “IN_USE_BY_APPLICATION”.
The sample program uses one DeviceHandle, so if you want to run multiple devices simultaneously, you need to allocate storage space for multiple device handles (see the sample program in the Delphi multichannel subfolder).

For the activated device, the Device information is collected (AVS_GetVersionInfo(), AVS_GetNumPixels(), AVS_GetParameter(), AVS_GetLambda()), and displayed in the main window. Thanks to the Windows API OnDeviceChange function, attachment and removal of spectrometers can be detected by the application (see the OnDeviceChange function in the source code).

NOTE: To match the structures that are used in the AvaSpec firmware the structures used in the AvaSpec Library should be compiled with *byte alignment*.

3.4 Starting a Measurement

Measurements can be started by clicking the “Start Measurement” button. The “Nr of Scans” field displays how many scans will be performed after one measurement request. Before a call to AVS_Measure() is done, the AVS_PrepareMeasurement() function is called with the parameters in the MeasConfigType structure. The “Prepare Measurement Settings” group in the figure below shows all the parameters in this MeasConfigType structure:



unsigned short	m_StartPixel
unsigned short	m_StopPixel
float	m_IntegrationTime
unsigned int	m_IntegrationDelay
unsigned int	m_NrAverages
DarkCorrectionType	m_CorDynDark
SmoothingType	m_Smoothing
unsigned char	m_SaturationDetection
TriggerType	m_Trigger
ControlSettingsType	m_Control

The parameters in the measurement structure have been briefly described in section 2.6. In this section, a more detailed description will be given.

3.4.1 Measurement Structure: Start- and Stoppixel

The start- and stoppixel are the first and last pixel to be sent to the PC. The full range for a spectrometer is between startpixel 0 and stoppixel “NrOfPixels-1”, where NrOfPixels specifies the total pixels available for the detector type used in the spectrometer (see also AVS_GetNumPixels()). If the wavelength range of a spectrometer exceeds 1100nm (1160nm for the AvaSpec-2048XL/x14/x16/x64) and the detector type is different from “HAMS9201, HAMG9208_512, SU256LSB or SU512LDB” (AvaSpec-NIR), the stoppixel can be set to the pixel number that corresponds to a wavelength of 1100 (1160) nm, because the sensitivity is almost zero at this wavelength range. Reducing the range increases the data transfer speed and allows you to transfer only the data that is relevant to the application.

Note that if m_StartPixel is not equal to zero, then a_pSpectrum[n] (see AVS_GetScopeData()), represents the measured data at pixel number m_StartPixel +n. Also, pSaturated[n] (see AVS_GetSaturatedPixels()) represents pixel number m_StartPixel +n. For example, if m_StartPixel = 10, then a_pSpectrum[0] represents the measured data at pixel number 10.

3.4.2 Measurement Structure: Integration Time

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 unit is milliseconds [ms], and the resolution 0.001 ms steps. The minimum integration time is detector dependent. The table below shows the values for the different detector types:

Spectrometer	Detector Type	Min. Integration time [ms]
AvaSpec-256-USB2	SENS_HAMS8378_256	0.56
AvaSpec-1024-USB2	SENS_HAMS8378_1024	2.20
AvaSpec-2048-USB2	SENS_ILX554	1.05
AvaSpec-2048L-USB2	SENS_ILX511	1.05
AvaSpec-NIR256-1.7, AvaSpec-NIR256-2.0TEC, AvaSpec-NIR256-2.5TEC, AvaSpec-NIR512-2.5-HSC	SENS_HAMG9201-256 SENS_HAMG9206-256 SENS_HAMG9208-256 SENS_HAMG9208-512	 0.01* 0.01
AvaSpec-NIR256-1.7TEC, AvaSpec-NIR256-2.2TEC**	SENS_SU256LSB	0.02
AvaSpec-NIR512-1.7TEC AvaSpec-NIR512-2.2TEC	SENS_SU512LDB	0.02
AvaSpec-3648-USB2	SENS_TCD1304	0.01
AvaSpec-102-USB2	SENS_TSL1301	0.06
AvaSpec-128-USB2	SENS_TSL1401	0.07

AvaSpec-2048x14-USB2	SENS_HAMS9840	2.17
AvaSpec-350F-USB2	SENS_ILX554	0.20
AvaSpec-950F-USB2	SENS_ILX554	0.50
AvaSpec-1350F-USB2	SENS_ILX554	0.70
AvaSpec-1650F-USB2	SENS_ILX554	0.85
AvaSpec-2048x16-USB2	SENS_HAMS11071_2048X16	1.82***
AvaSpec-2048x64-USB2	SENS_HAMS11071_2048X64	2.40****
AvaSpec-2048x64TEC-USB2	SENS_HAMS10420_11850	9.70
AvaSpec-HS1024x58-USB2	SENS_HAMS7031_1024X58	5.22
AvaSpec-HS1024x122-USB2	SENS_HAMS7031_11501	6.24
AvaSpec-2048XL-USB2	SENS_HAMS11155	0.002
AvaSpec-2048CL-EVO	SENS_HAMS11639	0.009
AvaSpec-4096CL-EVO	SENS_HAMS13496	0.009

* = 0.01ms for SENS_HAMS9201 in Firmware v. 000.025.000.000 and later, else 0.52ms
** = AvaSpec-NIR256-2.2TEC with SENS_SU256LSB detector released in 2011, and is the successor of the NIR2.2 with SENS_HAMS9201 detector
*** = 1.82 ms for SENS_HAMS11071_2048X16 in FPGA 006.003.000.000 or later, else 0.91ms
**** = 2.40 ms for SENS_HAMS11071_2048X64 in FPGA 006.003.000.000 or later, else 1.75ms.

The longest integration time is 10 minutes (600.000 ms).

3.4.3 Measurement Structure: Integration Delay

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. See also section 3.4.9 about using the integration delay in combination with the control settings: laser delay and pulse width. Integration delay has been implemented and tested for the detectors that support fast triggering. These Fast Triggering detectors (Sony ILX554, Sony ILX511, HamS11639 and HamS11155 in the AvaSpec-2048-USB2, AvaSpec-2048L-USB2, AvaSpec-2048CL/4096CL-EVO and AvaSpec-2048XL) can be reset in respectively 1.3, 3.3. 0.9 and 0.3 microseconds and start a new integration time immediately after this reset. The Toshiba TCD1304 in the AvaSpec-3648-USB2 also supports fast triggering in clearbuffermode (see also section 2.5.38), but because of the nonlinear behavior of the detector and the “missing” lower Counts in clearbuffer mode, this detector is less suitable for the fast triggering than the Fast-Triggering detectors listed above.

For the other detector types, it is recommended to set the integration delay parameter to 0 FPGA cycles.

3.4.4 Measurement Structure: Number of Averages

The signal to noise ratio of the scope data is improved by the square root of NrOfAverage. Averaging is done by the microcontroller at the AvaSpec board, therefore, no time is lost by sending the individual scans from the spectrometer to the PC.

3.4.5 Measurement Structure: Dynamic Dark Correction

The pixels of the CCD detector are thermally sensitive, which causes a small dark current, 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. This will happen if the “Correct for Dynamic Dark” option is enabled. Some detector types (AvaSpec-2048/2048L/3648) include dedicated optical black pixels. 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.

The back illuminated detectors in the AvaSpec-2048x14, 2048x16, 2048x64, 1024x122 and 1024x58 don't include optical black pixels, but a few elements in the shift register can also be used for correcting the raw data. The intensity at these elements may be different from the intensity of the (2048) data pixels in the dark, so the baseline may not fluctuate round zero, but the correction will result in a much more linear behavior of the data pixels when exposed to light. Therefore, it is strongly recommended to leave the (default) Dynamic Dark Correction state "Enabled".

The 2048XL uses 18 dummy pixels for correcting the raw data. Since these 18 pixels are located at positions 2050 to 2067, the stoppixel in the measurement structure should be set to 2067. Setting the stoppixel to a lower value for pixel reduction will have no effect with dynamic dark correction enabled, because these last 18 pixels are needed for the correction algorithm.

Some NIR detector types (NIR256-2.0TEC, NIR256-2.5TEC and NIR512-2.5-HSC) and the HamS11639 in the AvaSpec-2048CL also support dynamic dark, because a few datapixels are blackened during fabrication of the optical bench. These blackened pixels can then be used for dynamic dark correction. If the spectrometer does not include blackened datapixels, nor dedicated optical black pixels, enabling the dynamic dark correction results in a return value of -116 when calling AVS_PrepareMeasure(). This error can be neglected by the application (measurements can be proceeded), but dynamic dark correction is not possible in that case.

The Dark Correction Type structure includes an m_enable and m_ForgetPercentage field (see also section 2.6 on Data Elements). Measurements have shown that taking into account the historical dark scans, does not make much difference. The recommended value for m_ForgetPercentage is therefore 100.

3.4.6 Measurement Structure: Smoothing

The smoothing type structure includes a smoothpix and a smoothmodel field. In the current version of the AvaSpec Library there is just one smoothing model available (0), in which the spectral data is averaged over a number of pixels on the detector array. For example, if the smoothpix parameter is set to 2, the spectral data for all pixels x_n on the detector array will be averaged with their neighbor pixels x_{n-2}, x_{n-1}, x_{n+1} and x_{n+2}.

The optimal smoothpix parameter depends on the distance between the pixels at the detector array and the light beam that enters the spectrometer. For the AvaSpec-2048, the distance between the pixels on the CCD-array is 14 micron.

With a 200 micron fiber (no slit installed) connected, the optical pixel resolution is about 14.3 CCD-pixels. With a smoothing parameter set to 7, each pixel will be averaged with 7 left and 7 right neighbor pixels. Averaging over 15 pixels with a pitch distance between the CCD pixels of 14 micron will cover 15*14 = 210 micron at the CCD array. Using a fiber diameter of 200 micron means that we will lose resolution when setting the smoothing parameter to 7. Theoretically the optimal smoothing parameter is therefore 6. The formula is ((slit size/pixel size) – 1)/2.

In the table below, the recommended smoothing values for the AvaSpec spectrometer are listed as function of the light beam that enters the spectrometer. This light beam is the fiber core diameter, or if a smaller slit has been installed in the spectrometer, the slit width. Note that this table shows the optimal smoothing without losing resolution. If resolution is not an important issue, a higher smoothing parameter can be set to decrease noise at the price of less resolution.

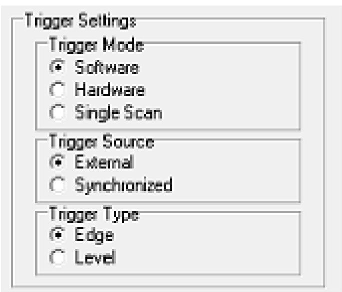
Slit or Fiber	AvaSpec-128 Pixel 63.5 µm	AvaSpec-256 1024 Pixel 25 µm	AvaSpec-HS 1024x58 1024x122 Pixel 24 µm	AvaSpec-2048, 2048L, 2048x14, 2048x16, 2048x64, 2048XL, 2048CL Pixel 14 µm	AvaSpec-4096CL 3648 Pixel 7-8 µm	AvaSpec-NIR256 Pixel 50 µm	AvaSpec-NIR512 Pixel 25 µm
10µm	n.a.	n.a.	0	0	0	n.a.	n.a.
25µm	n.a.	0	0	0-1	1	n.a.	0
50µm	0	0-1	0-1	1-2	2-3	0	0-1
100µm	0-1	1-2	1-2	3	5-6	0-1	1-2
200µm	1	3-4	3-4	6-7	12	1-2	3-4
400µm	2-3	7-8	7-8	13-14	24-25	3-4	7-8
500µm	3-4	9-10	9-10	17	31	4-5	9-10
600µm	4	11-12	11-12	21	37	5-6	11-12

3.4.7 Measurement Structure: Saturation Detection

The 16-bit A/D converter in the AvaSpec results in raw Scope pixel values between 0 and 65535 counts. If the value of 65535 counts is measured at one or more pixels, then these pixels are called to be saturated or overexposed. Saturation detection can be set off (m_SaturationDetection=0) or on (m_SaturationDetection=1). Saturation detection is done by the AvaSpec Library, after a measurement result has been sent to the PC. If a measurement is the result of a number of averages, the AvaSpec Library can only detect saturation if all NrOfAverage scans in a measurement were saturated for one or more pixels.

Only for AvaSpec-2048 spectrometers, the third level is available (m_SaturationDetection=2, autocorrect inverted pixels). The reason for this is that if the detector type in the AvaSpec-2048 (Sony-ILX554) is heavily saturated (at a light intensity of approximately 5 times the intensity at which saturation starts), it will return values <65535 counts. The other detector types in the AvaSpec-102, 128, 256, 1024, 2048L, 2048x14 and 3648 and AvaSpec-NIR do not show this effect, so no correction is needed. Normally, you don't need to use this third level for the AvaSpec-2048, but when measuring a peaky spectrum with some heavily saturated peaks, the autocorrect can be used. A limitation to this level is that it can be used only if no averaging is used (m_NrAverages=1). The AvaSpec-USB2 spectrometers with an AS5216 board Rev C and earlier were equipped with a 14-bit AD converter (range 0 .. 16383). In this case the detector is saturated at 16383 counts.

3.4.8 Measurement Structure: Trigger Type



The trigger type structure includes settings for:
- Trigger Mode (Hardware, Software, Single Scan (AS7010 / Mini only)
- Trigger Source (External, Synchronized) and
- Trigger type (Edge, Level).

Setting the Trigger Source to Synchronized is relevant if multiple spectrometers need to run synchronised (all spectrometers start a measurement at the same time).

This option will be described below under “Running multiple spectrometers Synchronized”.
Single channel spectrometers, or multiple spectrometers in ASYNC mode can operate in one of the three following Trigger settings (Trigger Source should be set to “External”):

Trigger Mode = Software
This Trigger setting is used when one or more (nrms) measurements should start after a measurement request in the software (AVS_Measure() call). The Edge/Level is irrelevant because this only applies to an external hardware trigger.

Trigger Mode = Hardware, Edge triggered
This trigger setting is used when one or more (nrms) measurements should start after an external hardware trigger pulse has been received at pin 6 of the DB26 connector. First a measurement request is posted in the software (AVS_Measure() call). Then the spectrometer waits until a rising edge of the TTL-input pulse is detected at pin 6 of the DB26 connector before nrms scans are started. The delay between the rising edge of the TTL pulse and the start of the integration time cycle depends on the spectrometer type, as shown in the table below.

Spectrometer Type	Minimum Delay [μs]	Maximum Delay [μs]
AvaSpec-128-USB2	9	60
AvaSpec-256-USB2	0.80	0.84
AvaSpec-1024-USB2	0.80	0.84
AvaSpec-2048-USB2*	1.28	1.30
AvaSpec-2048L-USB2	3.28	3.30
AvaSpec-3648-USB2 (clearbuffer mode)**	0.28	0.30
AvaSpec-NIR256-1.7, AvaSpec-NIR256-1.7-EVO AvaSpec-NIR256-2.5-HSC	536.48	536.73
AvaSpec-NIR512-2.5-HSC	531.74	533.93
AvaSpec-NIR512-1.7-EVO AvaSpec-NIR512-2.5-HSC-EVO	527.64	529.83
AvaSpec-NIR256-1.7TEC, AvaSpec-NIR256-2.2TEC	4.92***	5.75***
AvaSpec-NIR512-1.7TEC, AvaSpec-NIR512-2.2TEC	4.92****	5.75****
AvaSpec-2048x14-USB2	-2170	0
AvaSpec-2048x16-USB2	-1820	0
AvaSpec-2048x64-USB2	-2400	0
AvaSpec-2048x64TEC-USB2	-9700	0
AvaSpec-HS1024x58-USB2	-5220	0
AvaSpec-HS1024x122-USB2	-6240	0
AvaSpec-2048XL-USB2	0.37	0.39
AvaSpec-2048CL-EVO	0.90	0.92
AvaSpec-4096CL-EVO	0.90	0.92

* The AvaSpec-350F-USB2, AvaSpec-950F-USB2, AvaSpec-1350F-USB2 and AvaSpec-1650F-USB2 use the same detector as the AvaSpec-2048-USB2 and will therefore have the same trigger response characteristics as the AvaSpec-2048-USB2

** The delay for the AvaSpec-3648-USB2 in prescan mode strongly depends on the integration time setting, but can be calculated within 0.02 μs precision by the following equations:

Scanspassed = floor((Inttime-0.002+3.6961)/(Inttime-0.002))
min_delay = 0.00183 + Scanspassed*(Inttime-0.002) [ms]
max_delay = 0.00185 + Scanspassed*(Inttime-0.002) [ms]
Inttime = Integration time setting in milliseconds in the preparemeasurement structure

Example1: Inttime = 0.1ms
Scanspassed = floor(38.72) = 38
min_delay = 0.00183 + 38*0.098 = 3.72583 ms
max_delay = 3.72585 ms

Example2: Inttime = 0.01ms
Scanspassed = floor(463.01) = 463
min_delay = 0.00183 + 463*0.008 = 3.70583ms
max_delay = 3.70585 ms

So if the application allows that the AvaSpec-3648-USB2 in prescan mode is triggered a couple of milliseconds before the event that needs to be measured, this event can be shifted with high precision into the integration time cycle of the spectrometer. Moreover, the integration delay parameter (section 3.4.3) can be used to add additional delay in steps of 21 nanoseconds to the min_delay calculated above.

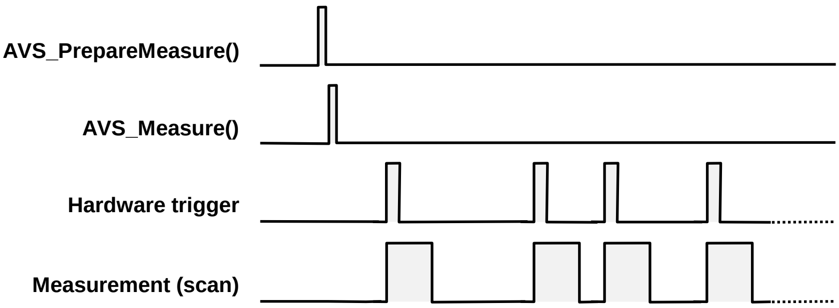
*** 4.92 - 5.75 µs with FPGA version 6.4 and later, 137.5 - 138.3 µs with FPGA version 6.3

**** 4.92 - 5.75 µs with FPGA version 6.4 and later, 251.1 - 252.8 µs with FPGA version 6.3

Trigger Mode = Hardware, Level triggered
This trigger setting is used when scans should be performed as long as the external trigger at pin 6 of the DB26 connector is HIGH. The spectrometer will start to accumulate data (take scans at the selected integration time) at the rising edge of the TTL pulse and will continue to do so as long as the TTL signal remains high. When the signal becomes low, the average of the accumulated data (except for the last scan) will be sent. This mode is especially useful for conveying belt applications, when a product needs to be scanned, independent of the transport speed.
Level triggering is only supported on AS5216 and AS7010 boards and in combination with the trigger mode 'Hardware'.

Trigger Mode = Single Scan (AS7010 / Mini only)
The above hardware trigger modes will start taking all scans that were requested in the AVS_Measure() call after the first trigger. If you want to take a single scan per trigger, you can set the trigger mode to Single Scan. Please note that scans in StoreToRam mode will not be back to back when using Single Scan trigger mode, as the trigger will take some time. For the AS5216, a custom firmware is available that will allow single scan triggering.

The principle of Single Scan triggering is depicted below;



Running multiple spectrometers Synchronized
All USB2 platform spectrometers can be connected by a SYNC cable. In syncmode, one spectrometer is configured as "Master" by calling the AVS_SetSyncMode() function for this channel with the a_Enable flag set to 1. The trigger source for the Master channel should **not** be set to Synchronized, but to External. The trigger mode for the Master can be set to Software (if a measurement should start after a measurement request in the software), or to Hardware (if a measurement should start after an external hardware trigger pulse at pin 6 of the Master DB26 connector. All other ("slave") spectrometers are set into "Synchronized" mode by setting the Trigger Source to "Synchronized" and the Trigger Mode to "Hardware".
A synchronized measurement is started by calling AVS_Measure() first for all slave channels. As a result, these channels start listening to their SYNC input port. Secondly a measurement request (call to AVS_Measure()) needs to be posted for the Master channel. If the trigger mode for the Master is "software", this result in nrms measurements for all channels. If the trigger mode for the Master is "hardware", the nrms measurements for all channels are started after an external trigger has been received at pin 6 of the Master DB26 connector. The nrms parameter in the AVS_Measure() function should be set to the same value for all activated channels.
Source code for the sample programs that support synchronization of multichannel systems can be found in the following folders:

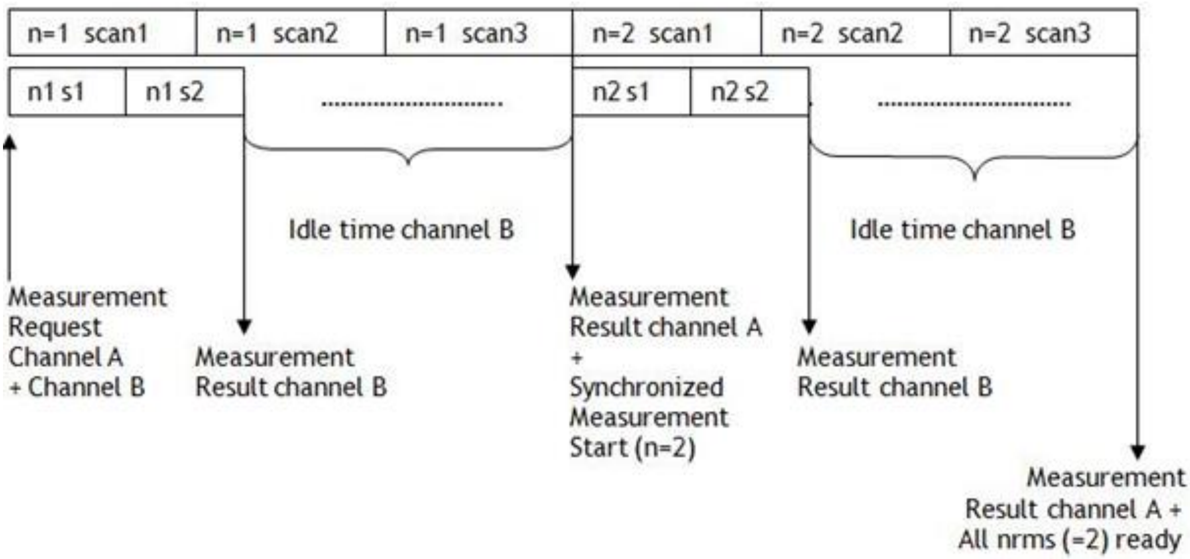
<u>32-bit versions</u>
..\examples\Borland Delphi 6\multichannel\
..\examples\Codegear Delphi 2009\multichannel\
..\examples\LabView\polling_mc\

<u>64-bit versions</u>
..\examples\Delphi XE3\multichannel\
..\examples\LabView\polling\polling_mc.llb

Synchronization is done at a measurement level. A measurement can include a number of scans to average. This "number of average" scans is only synchronized for the first scan. For example, if the number of measurements, integration time and number of average for two channels are set to:

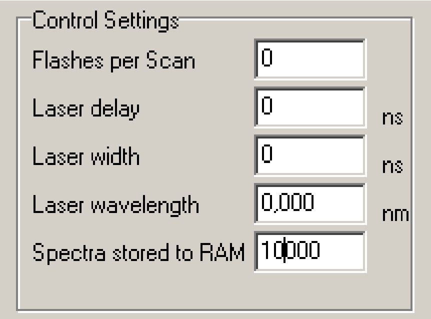
Channel A: nrms=2, integration time 100ms, average 3
Channel B: nrms=2, integration time 65ms, average 2,

then the data acquisition timing and response in synchronized mode will look like:



NOTE: that in the example above, the number of averages for channel B can be set to 4 without losing time because the extra two scans will be taken in the idle time for channel B.

3.4.9 Measurement Structure: Control Settings

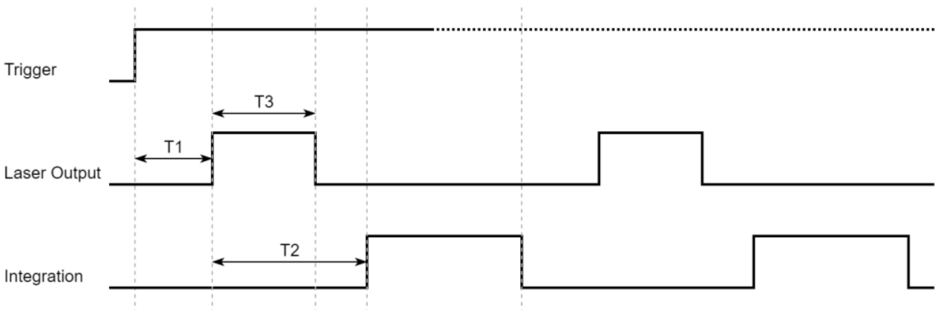


The Control Settings include parameters to control;

- A pulsed lightsource (m_StrobeControl)
- A laser pulse (m_LaserDelay and m_LaserWidth)
- The Number of Spectra that will be stored to onboard RAM (m_StoreToRam)

Pulsed light source control
A pulsed light source like the AvaLight-XE needs to be synchronized with the integration time cycle. The m_StrobeControl parameter determines the number of pulses the spectrometer sends out at pin 5 at the DB26 connector during one integration time cycle. The maximum frequency at which the AvaLight-XE operates is 100 Hz. This means that the minimum integration time for 1 pulse per scan is 10 ms. When setting the number of pulses e.g. to 3, the minimum integration time should be 30 ms. The AvaSpec Library does not check for this limitation because other light sources may operate at higher frequencies, and should also be controllable by the AvaSpec Library.

Laser pulse control
For the fast trigger detectors ILX554, ILX511, S11155 and S11639 in the AvaSpec-2048, 2048L, 2048XL and 2048CL, pin 23 at the DB26[1] connector can be used to send out a TTL signal which is related to the start of the integration time cycle. As depicted in the timing diagram below, a measurement is started at the rising edge of the *Trigger* signal. This can be a hardware or software trigger, see also section 3.4.8. The TTL signal at pin 23 (Laser) is set after the laser delay (T1) expires. The pulse width for the laser pulse (T3) is set by the m_LaserWidth parameter. The integration time cycle starts after the integration delay parameter (see section 3.4.3) expires. After the first (hardware or software) trigger, the laser output of the spectrometer will be driven continuously with every integration cycle.



[1] Please note that pin numbers are valid for DB26 connectors on –USB2 and –EVO spectrometers

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.

Laser Induced Breakdown Spectroscopy (LIBS) is an application where the integration delay is used in combination with a TTL-out at the DB-26 connector to fire a laser. After a measurement request (or on an external hardware trigger), the laser is fired by the TTL-out. The integration time period should not include the laser light, so the start of the integration time needs to be delayed. A typical integration delay in LIBS applications is about 1μs (ILX554 detector in AvaSpec-2048-USB2, see also section 3.4.3).

Laser wavelength
The Laser wavelength (m_LaserWaveLength) control setting is not used in the current version of AvaSpec. A value can be entered, but the AvaSpec firmware does not use this information.

StoreToRam
As of AS5216 firmware version 0.20.0.0 the StoreToRam function has been implemented. To use this function, you must set the requested number of scans in the m_StoreToRam control setting, and start measuring with a call to AVS_Measure() using 1 as the number of measurements (a_Nmsr).
For the AS5216, there is an amount of 4MB available for scans, corresponding with 1013 scans of 2048 pixels. The AvaSpec Mini and the AS7010 have much more memory, allowing for 7783 and 16383 scans of 2048 pixels respectively. Scanning less pixels will in each case yield a larger capacity in scans. The AVS_Measure() message signaling the arrival of data will have a WParam value equal to the number of scans stored in RAM. In regular measurements, this value only signals success (with value ERR_SUCCESS) or failure (with a negative error message). After the application is signaled, due to either finishing the StoreToRam function fully or cancelling it by AVS_StopMeasure(), it can perform one of the following operations;

- Read out the stored scans; the scans can subsequently be read with a corresponding number of calls to AVS_GetScopeData(). If you request more scans than will fit in memory, scanning will continue until the memory is fully used. Therefore you should always request the number of scans that is returned in WParam (when using Windows Messaging).
- Omit the stored scans; the scans that are ready in the spectrometer can be omitted by terminating the StoreToRam mode by calling AVS_StopMeasure() first and then starting another StoreToRam measurement or regular measurement with AVS_Measure() right after. The current measurement data will then be cleared and a new measurement sequence will be started immediately.

Alternatively, when using AVS_PollScan() instead of a message driven interface, the AVS_PollScan() function will return 1 when the StoreToRam scans are available, and 0 as long as they are not.

If using StoreToRAM in combination with AVS_PollScan(), the number of scans that can be processed by subsequently calling AVS_GetScopeData() will normally be equal to the requested number of scans in the StoreToRAM parameter. If more scans are requested than can be stored (e.g. 1500 scans of 2048 pixels), it can happen that AVS_GetScopeData() will be called too many times. In case of the example, only the first 1013 calls to AVS_GetScopeData() will return SUCCESS. The next call will return the error code ERR_NO_SPECTRA_IN_RAM, which can be used by the application software as an additional stop condition for reading spectra from RAM. However, reading beyond the number of scans that can be stored in RAM is a time consuming event, so it is not recommended to request more scans than the maximum that can be stored.

The StoreToRam functionality has been implemented in most sample programs that come with the AvaSpec Library interface package. To illustrate how to use StoreToRAM in combination with AVS_PollScan(), a simple LabView sample program is included.

Dynamic StoreToRam (DSTR)
As of AvaSpec firmware version 1.10.0.0, the StoreToRam feature has been improved with a Dynamic mode. In this mode, you can start to read out the stored scans before the measurement session is finished, thereby freeing up memory and enlarging the number of scans that can be read in a session. The downside to this is that the capacity can now be unpredictable, depending on cycle time. The capacity can be as large as allowing for infinite measurements, but it can also be too small for the requested number, in which case the measurement will be stopped. A memory status value is available that can be read after a notification (a Windows message or a callback function). This status will be updated at the beginning of a session, every 500 millisecond during the session, and when the session is stopped intentionally by the spectrometer (e.g. memory overrun or DSTR finish).

The DSTR mode can be used by defining the number of scans to be read in the m_StoreToRam control setting. Enter 0 for an infinite number. The DSTR measurement can be started by calling AVS_Measure() using -2 as the number of measurements (a_Nmsr). A Windows message or callback function will be issued after each completed measurement (this is like it is done in regular continuous measurements, more than like it is done in the classic StoreToRam mode). You can then read each scan with AVS_GetScopeData, after receiving the corresponding Windows message (WM_MEAS_READY), or in a callback function. The DSTR status can be monitored by calling AVS_GetDstrStatus(), after receiving the corresponding Windows message (WM_DSTR_STATUS) or in a callback function.

The DSTR functionality has been implemented in the full Qt5 sample.

3.5 Measurement Result

If a measurement is ready, the windows message WM_MEAS_READY is sent to the application. The WParam value of the message will be:

- 0 in regular measurements (where the StoreToRam parameter is zero) to indicate SUCCESS
- 0 in StoreToRam mode, WParam holds the number of spectra that were actually saved in RAM
- < 0 in case an error occurred (see section 2.6.1 for return value constants)

The LParam value of the message contains the devicehandle for the spectrometer for which the data is ready.

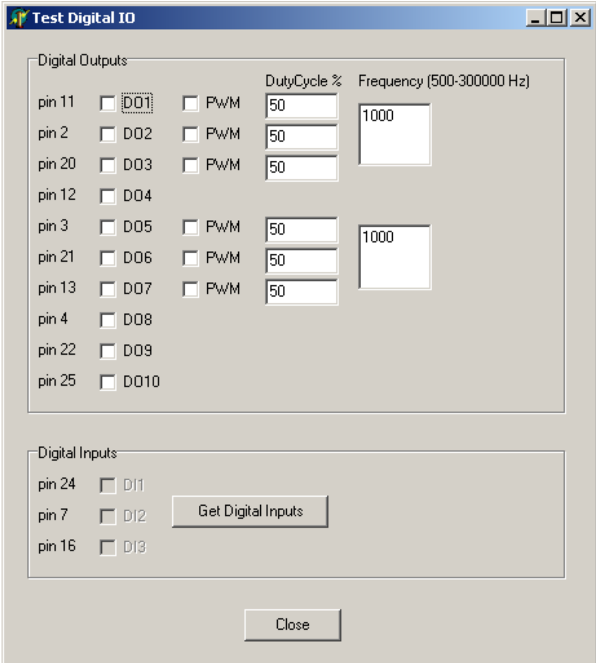
LabVIEW cannot easily respond to the incoming Windows message that signals the arrival of new data. AVS_PollScan() allows the application program to poll the arrival of data, i.e. to actively get the status of this data, instead of letting a message handler react to the Windows message from the Library.
By calling the function AVS_GetScopeData(), the spectral data is stored in the application for further processing.

3.6 Digital IO

The AS5216 and AS7010 platform spectrometers have 10 programmable digital output pins and 3 programmable input pins available at the DB26 connector. The function AVS_SetDigOut() and AVS_GetDigIn() can be used to control these ports. Moreover, 6 out of the 10 programmable output ports can be configured for pulse width modulation. With the AVS_SetPwmOut() function, a frequency and duty cycle can be programmed for these 6 digital output ports.

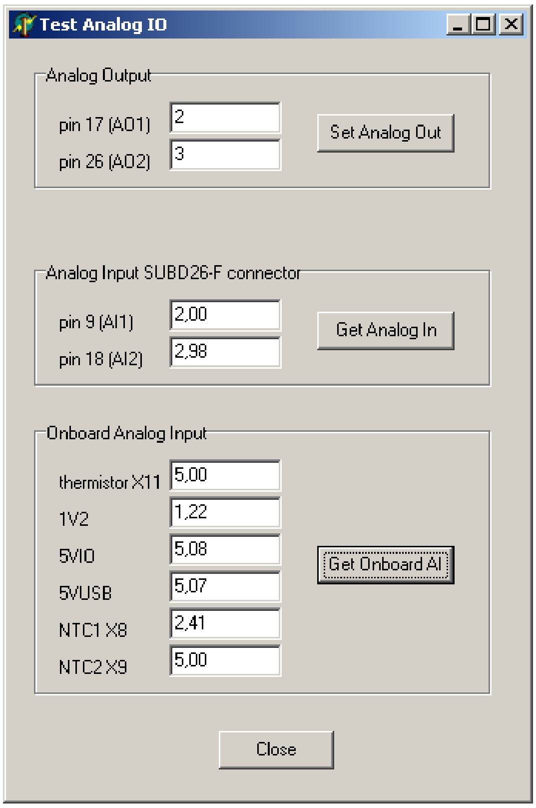
The PWM functionality can be used e.g. in controlling the intensity (duty cycle) of an AvaLight-LED light source, which receives input from DO1 (pin 11 of the DB26 connector).

ID: 020381



The AvaSpec Mini has six bidirectional digital ports, available at the micro HDMI connector. A HDMI terminal adapter is available, which has a different pin numbering scheme. Please refer to the description of the AVS_SetDigOut() and AVS_GetDigIn() functions for more information.

3.7 Analog IO



The AS5216 spectrometers have 2 programmable analog output pins and 2 programmable analog input pins available at the DB26 connector. The functions AVS_SetAnalogOut() and AVS_GetAnalogIn() can be used to control these ports. For the Analog Out signals, an 8-bit DAC is used. The Analog In signals are converted by the internal 10-bit ADC's.
A number of onboard analog signals can be retrieved as well with the AVS_GetAnalogIn() function. One of these onboard signals is the NTC1 X8 thermistor which can be used for onboard temperature measurements. The polynomial for converting the voltage (U) to degrees Celsius for NTC1 is:

$$\text{Temp } [^{o}\text{C}] = 118.69 - 70.361 \cdot U + 21.02 \cdot U^2 - 3.6443 \cdot U^3 + 0.1993 \cdot U^4$$

The thermistor X11 is the signal received from a TE cooled detector and can be used to monitor the detector temperature. NTC2 X9 is not mounted.
The 1V2, 5VIO and 5VUSB are used internally to test the power supply.

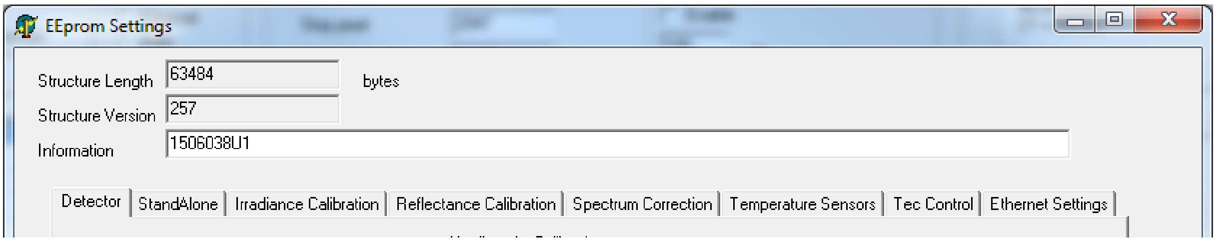
The AvaSpec Mini also has two programmable analog output pins and two analog input pins, , available at the micro HDMI connector. A HDMI terminal adapter is available, which has a different pin numbering scheme. The NTC1 thermistor is read with identifier value 0, in the position of the Thermistor X1. It must be converted to degrees Celsius with the NTC1 polynomial. The 1V2, 5VIO, 5VUSB and NTC1X8 inputs are not supported in the Mini.

The AS7010 has a digital temperature sensor, and the board temperature (identifier value 6) is returned directly as degrees Celsius. You do not have to convert the input with a polynomial, and the NTC1 polynomial in the EEPROM should therefore be set to (0,1,0,0,0). The 1V2, 5VIO and 5VUSB inputs are not supported in the AS7010.

3.8 EEPROM

The EEPROM parameters in the DeviceConfigType structure have been briefly described in section 2.66. In this section, a more detailed description will be given. The C++Builder and Delphi sample programs display most of the parameters in the structure. The Structure Length (m_Len), Structure Version (m_ConfigVersion) and InfoString (m_aUserFriendlyId[64]) are shown on top of the tabs that correspond to the structures that are used to group the parameters into the following categories:

DetectorType	m_Detector,
IrradianceType	m_Irradiance,
SpectrumCalibrationType	m_Reflectance,
SpectrumCorrectionType	m_SpectrumCorrect,
StandaloneType	m_StandAlone,
TempSensorType	m_Temperature[3]
TecControlType	m_TecControl
ProcessControlType	m_ProcessControl (not displayed in sample program)
EthernetSettingsType	m_EthernetSettings
OemDataType	m_OemData (not displayed in sample program)

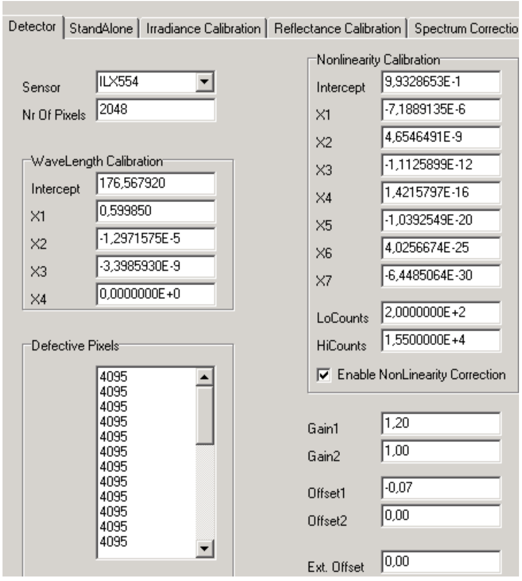


The structure version is used internally to maintain compatible between different versions of the Library and firmware. The Information character string can be used e.g. to write a user friendly name for the spectrometer.

3.8.1 EEPROM Structure: Detector Parameters

ID: 020381

The detector parameters are defined in the DetectorType structure, which includes the following elements:



SensorType	m_SensorType
unsigned short	m_NrPixels
float	m_aFit[5]
bool	m_NLEnable
double	m_aNLCorrect[8]
double	m_aLowNLCounts
double	m_aHighNLCounts
float	m_Gain[2]
float	m_Reserved
float	m_Offset[2]
float	m_ExtOffset
unsigned short	m_DefectivePixels[30]

SensorType and Number of Pixels
The boards support many different detectors which are used in the AvaSpec spectrometers as shown in the tables below:

AS5216:

Spectrometer	DetectorType	Number of Pixels
AvaSpec-102-USB2	SENS_TSL1301	102
AvaSpec-128-USB2	SENS_TSL1401	128
AvaSpec-256-USB2	SENS_HAMS8378_256	256
AvaSpec-1024-USB2	SENS_HAMS8378_1024	1024
AvaSpec-2048x14-USB2	SENS_HAMS9840	2048
AvaSpec-2048x16-USB2	SENS_HAMS11071_2048X16	2048
AvaSpec-2048x64-USB2	SENS_HAMS11071_2048X64	2048
AvaSpec-2048x64TEC-USB2	SENS_HAMS10420_11850	2048
AvaSpec-NIR256-1.7, AvaSpec-NIR256-2.0TEC, AvaSpec-NIR256-2.5TEC	SENS_HAMS9201	256
AvaSpec-NIR512-2.5-HSC	SENS_HAMG9208_512	512
AvaSpec-NIR256-1.7TEC, AvaSpec-NIR256-2.2TEC*	SENS_SU256LSB	256
AvaSpec-NIR512-1.7TEC, AvaSpec-NIR512-2.2TEC	SENS_SU512LDB	512
AvaSpec-2048-USB2	SENS_ILX554	2048
AvaSpec-350F-USB2	SENS_ILX554	350
AvaSpec-950F-USB2	SENS_ILX554	950
AvaSpec-1350F-USB2	SENS_ILX554	1350
AvaSpec-1650F-USB2	SENS_ILX554	1650
AvaSpec-2048L-USB2	SENS_ILX511	2048
AvaSpec-3648-USB2	SENS_TCD1304	3648
AvaSpec-HS1024x58-USB2	SENS_HAMS7031_1024X58	1024
AvaSpec-HS1024x122-USB2	SENS_HAMS7031_11501	1024
AvaSpec-2048XL-USB2	SENS_HAMS11155	2068
AvaSpec-2048CL-USB2	SENS_HAMS11639	2048

* AvaSpec-NIR256-2.2TEC with SENS_SU256LSB detector (released in 2011), it is the successor of the NIR2.2 with SENS_HAMS9201 detector

Mini:

Spectrometer	DetectorType	Number of Pixels
AvaSpec-Mini-2048	SENS_ILX554	2048
AvaSpec-Mini-2048L	SENS_ILX511	2048
AvaSpec-Mini-3648	SENS_TCD1304	3648
AvaSpec-Mini-2048CL	SENS_HAMS11639	2048
AvaSpec-Mini-4096CL	SENS_HAMS13496	4094

AS7010:

Spectrometer	DetectorType	Number of Pixels
AvaSpec-ULS2048L-EVO	SENS_ILX511	2048
AvaSpec-ULS2048CL-EVO	SENS_HAMS11639	2048
AvaSpec-ULS4096CL-EVO	SENS_HAMS13496	4094
AvaSpec-ULS2048XL-EVO	SENS_HAMS11155	2068
Avaspec-HS1024x58TEC-EVO	SENS_HAMS7031_1024X58	1024
AvaSpec-NIR256-2.5-HSC	SENS_HAMS9201	256
AvaSpec-NIR512-2.5-HSC	SENS_HAMG9208_512	512
AvaSpec-NIR256-1.7-EVO	SENS_HAMS9201	256
AvaSpec-NIR512-1.7-EVO	SENS_HAMG9208_512	512
AvaSpec-NIR256-1.7-HSC	SENS_SU256LSB	256
AvaSpec-NIR512-1.7-HSC	SENS_SU512LDB	512

For each detector, different FPGA firmware is needed. The SensorType parameter should therefore not be changed unless new FPGA firmware for another detector type has been loaded.
The number of pixels is determined by the detector type and should therefore not be changed, unless another detector type has been connected and the right FPGA code has been loaded.
Also for the Fast Series (350F, 950F, 1350F, 1650F), the number of pixels is fixed and should not be changed.

Wavelength Calibration
The polynomial coefficients in m_aFit[5] describe the relation between the pixelnumber of the detector array (0..m_NrPixels-1) and the corresponding wavelength in nanometer at this pixelnumber:

λ = m_aFit[0] + m_aFit[1] *pixnr + m_aFit[2] *pixnr^2 + m_aFit[3] *pixnr^3 + m_aFit[4] *pixnr^4
In the function AVS_GetLambda(), the m_aFit coefficients are used internally to store the wavelength numbers into an array.

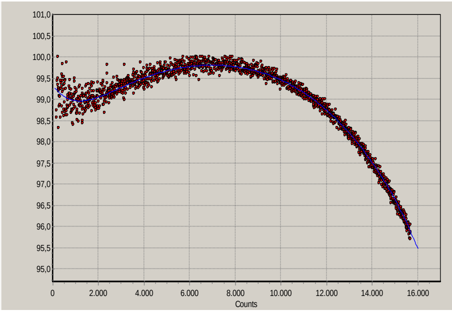
Nonlinearity Calibration and Correction
A polynomial can be used to correct for nonlinear behavior of the detector. The polynomial coefficients can be stored in the EEPROM and used by the application software to correct the raw AD Counts. The nonlinearity calibration service (determination of the polynomial coefficients) is included in the IRRAD-CAL irradiance calibration service, but can also be ordered separately (NL-Calibration). The m_aLowNLCounts and m_aHighNLCounts parameters have been added since AvaSpec Library version 1.1, to be able to limit the range (in counts) for which the correction polynomial should be applied. The correction that needs to be implemented in the application software can be illustrated by using an example:

Suppose the following nonlinearity polynomial has been calculated:

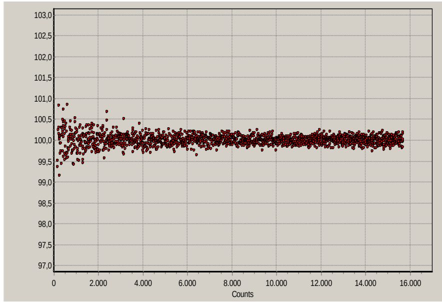
m_aNLCorrect[0] = 9.93286529334744E-001
m_aNLCorrect[1] = -7.18891352982627E-006
m_aNLCorrect[2] = 4.65464905353804E-009
m_aNLCorrect[3] = -1.11258994803382E-012
m_aNLCorrect[4] = 1.42157972847117E-016
m_aNLCorrect[5] = -1.03925487491128E-020
m_aNLCorrect[6] = 4.02566735990250E-025
m_aNLCorrect[7] = -6.44850644473040E-030

m_aLowNLCounts = 200.0
m_aHighNLCounts = 15500.0

The polynomial is calculated by measuring the AD Counts for a number of pixels (10) over different integration times to get the pixel data over a wide range from (in this example) 200 to 15500 counts. The measured AD Counts are corrected for the offset value by subtracting the dark spectrum. For each of the 10 pixels in the measurement the counts per second is calculated and normalized to its maximum value, which is set to 100%. In the left figure below, the normalized counts per second are displayed against the measured AD Counts (corrected for dark). The polynomial is the best fit through these measured points. The right figure below has been created by applying the polynomial to the measured points, and recalculating the normalized counts per second. It is important to realize that the polynomial should be applied to the AD Counts that have been corrected for the dark counts.



Before linearization



After linearization

In the application software, a dark spectrum needs to be saved first and subtracted from the measured AD Counts, before the correction is applied. For example, suppose the measured AD Counts in the dark for a pixel is a value of 300 Counts. At a certain light intensity, the measured AD Counts for this pixel becomes a value of 14000 Counts. The AD Counts corrected for dark therefore becomes 13700. The Normalized Counts Per Second can be calculated from the polynomial:

NCPS= m_aNLCorrect [0] +
m_aNLCorrect [1] *13700 +
m_aNLCorrect [2] *13700^2 +
m_aNLCorrect [3] *13700^3 +
m_aNLCorrect [4] *13700^4 +
m_aNLCorrect [5] *13700^5 +
m_aNLCorrect [6] *13700^6 +
m_aNLCorrect [7] *13700^7 = 0.97741

The AD Counts value corrected for linearity and dark becomes 13700/0.97741 = 14017 Counts. The AD Counts value corrected for linearity only (not for dark) becomes 14017+300 = 14317 Counts.

The m_aLowNLCounts and m_aHighNLCounts parameters can be used to limit the range for the correction (in counts) for which the polynomial should be applied. The use of polynomials beyond the range of measured data points can give erratic corrections. In AvaSoft, Avantes uses the same correction factor (NCPS) for measured counts (corrected for dark) that are lower than m_aLowNLCounts as is used for m_aLowNLCounts, and for counts higher than m_aHighNLCounts the same NCPS as is used for m_aHighNLCounts. In the example above, NCPS[200] = 0.99203 and all counts <= 200 will be corrected in AvaSoft by dividing through 0.99203. Likewise, NCPS[15500] = 0.96099 and all counts >= 15500 will be corrected in AvaSoft by dividing through 0.96099. All counts: 200<counts<15500 will be corrected by the NCPS calculated by the polynomial.

Using the nonlinearity correction polynomial in combination with the 16-bit ADC Counts range (see also section 2.5.39) does require a small modification in your application software, since the polynomial was recorded in 14-bit mode, and therefore should be applied to a 14-bit range when calculating the NCPS. This will be illustrated by introducing the variable “ADCFactor” to the equations that are used in the correction (same example as above, same polynomial). The value of “ADCFactor” becomes 0.25 when running in 16bit ADC mode and 1.0 when running in 14-bit ADC mode.

In 16-bit ADC mode, the measured counts will be a factor 4 higher than in 14-bit mode, or with a 14-bit ADC. Therefore, the same pixel of the same spectrometer in this example returns 4*300 = 1200 Counts for dark data and 4*14000 = 56000 Counts at a certain light intensity. The AD Counts corrected for dark therefore becomes 54800. The Normalized Counts Per Second can be calculated from the polynomial:

NCPS= m_aNLCorrect [0] +
m_aNLCorrect [1] * (ADCFactor * 54800) +
m_aNLCorrect [2] * $(ADCFactor * 54800)^2$ +
m_aNLCorrect [3] * $(ADCFactor * 54800)^3$ +
m_aNLCorrect [4] * $(ADCFactor * 54800)^4$ +
m_aNLCorrect [5] * $(ADCFactor * 54800)^5$ +
m_aNLCorrect [6] * $(ADCFactor * 54800)^6$ +
m_aNLCorrect [7] * $(ADCFactor * 54800)^7$ = 0.97741

The AD Counts value corrected for linearity and dark becomes 54800/0.97741 = 56067 Counts. The AD Counts value corrected for linearity only (not for dark) becomes 56067+1200 = 57267 Counts.

Using the m_aLowNLCounts and m_aHighNLCounts parameters in 16bit mode also requires to include the ADCFactor when comparing the measured Counts to these parameters:
m_aLowNLCounts = 200, therefore:

if ADCFactor*(measured counts (corrected for Dark))<200, use NCPS[200] = 0.99203
else if ADCFactor*(measured counts (corrected for Dark))>15500, use NCPS[15500] = 0.96099
else, calculate NCPS as shown above.

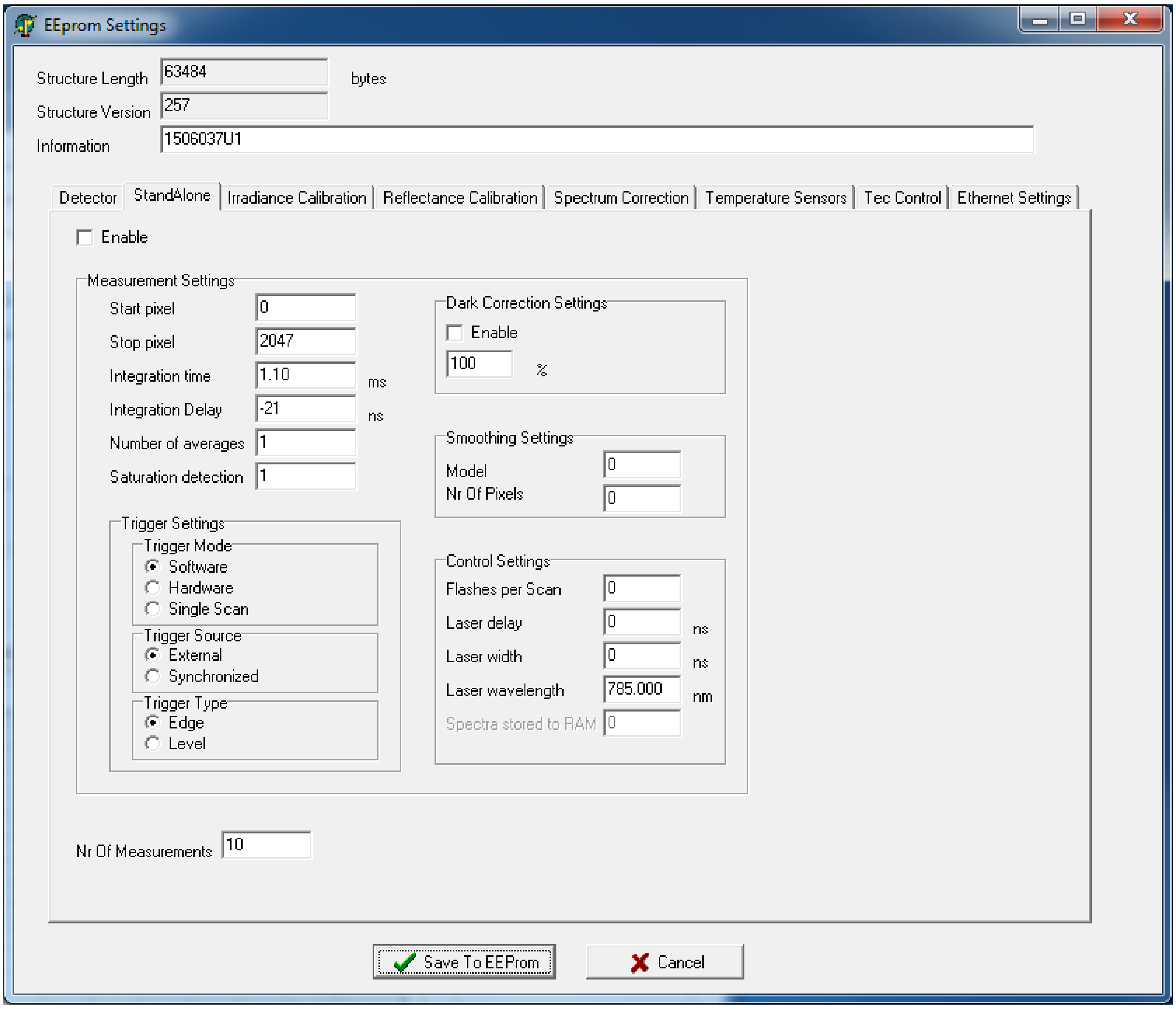
In the example above, all counts (corrected for dark) <= 800 will be corrected in AvaSoft by dividing through 0.99203. Likewise, all counts (corrected for dark) >= 62000 will be corrected in AvaSoft by dividing through 0.96099. All counts (corrected for dark): 800<counts<62000 will be corrected by the NCPS calculated by the polynomial.

Gain and Offset
These parameters have been optimized by Avantes, and there should be no need to change these values. The m_Gain and m_Offset parameters are used to optimize the Gain and Offset of the AD Converter. Most detector types use only the m_Gain[0] and m_Offset[0]. The parameters m_Gain[1] and m_Offset[1] are only used by the SENS_SU512LDB and SENS_HAMG9208_512 detectors (512 pixel NIRs). The m_ExtOffset parameter is used to be able to match the detector output range with the ADC range.

Defective Pixels
The m_DefectivePixels[30] array can be used to store the pixel numbers that should be eliminated from the data transfer. The AvaSpec 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 “NrOfPixels-1”, where NrOfPixels specifies the total pixels available for the detectortype used in the spectrometer (see also AVS_GetNumPixels()).

The AvaSpec Library evaluates the array m_DefectivePixels[i] in an increasing order until a pixel is specified which is equal or larger than the number of pixels in the detector.

3.8.2 EEPROM Structure: Standalone Parameters



The StandaloneType structure includes a boolean (m_Enable) which is not used in the standard version, but which can be used for user specific standalone functionality. The Measurement parameters are also included in this structure, as well as the Number of Measurements parameter (m_Nmsr).

The Measurement parameter structure (MeasConfigType) has been described in detail in section 3.44, as well as the Number of Measurements parameter (m_Nmsr).

3.8.3 EEPROM Structure: Irradiance, Reflectance Calibration and Spectrum Correction

The m_Irradiance, m_Reflectance and m_SpectrumCorrect parameters occupy together over 99% of the defined memory in the EEPROM structure (Sizeof(DeviceParamType) with the m_aReserved block excluded). This is because each of these three parameters include an array of 4096 (MAX_NR_PIXELS) float numbers which can hold pixel specific calibration data.

The Irradiance Calibration structure (IrradianceType) has been defined to store the results of an irradiance intensity calibration in EEPROM, as well as the settings during this calibration (integration time, smoothing, measurement setup, fiber diameter). By reading these data from EEPROM, it will be possible to convert a spectrum with raw scope data into an irradiance spectrum.

How to convert ScopeData (A/D Counts) to a power distribution [μWatt/(cm^2.nm)]

In the application software, the smoothpix value in the preparemeasurement structure should be set to the same value as the smoothpix during the intensity calibration. This value can be found in m_Irradiance.m_IntensityCalib.m_Smoothing.m_SmoothPix.

Also, before the irradiance intensity for a pixel i can be calculated, a dark spectrum (= A/D Counts with no light exposed to spectrometer) should be saved (once) at the integration time that will be used in the measurements. The dark spectrum for each pixel i can be called e.g. darkdata(i).

The irradiance intensity at a certain pixel i (i = 0 .. totalpixels-1) can then be calculated from:

ScopeData(i)	=	Measured A/D Counts at pixel i (AVS_GetScopeData)
DarkData(i)	=	Dark data at pixel i, saved in application software
IntensityCal(i)	=	m_Irradiance.m_IntensityCalib.m_aCalibConvers[i]
CalInttime	=	m_Irradiance.m_IntensityCalib.m_CalInttime
CurInttime	=	Integration time in measurement (used in the PrepareMeasurement structure)

The equation for irradiance intensity at pixel i then becomes:

```
Inttimefactor = (CalInttime/CurInttime)
Irradiance Intensity = Inttimefactor*((ScopeData(i) - DarkData(i))/IntensityCal(i))
```

If Scopedata(i) and Darkdata(i) are taken with the 16-bit ADC Counts range (see also section 2.5.39 on function AVS_UseHighResAdc()), an additional “ADCFactor” needs to be added to the equation above, because the intensity calibration (if performed by Avantes, or by using AvaSoft application software) is always recorded in 14bit mode. The value of “ADCFactor” becomes 0.25 when running in 16-bit ADC mode and 1.0 when running in 14bit ADC mode. The equation becomes:

```
Irradiance Intensity = ADCFactor * Inttimefactor *
                       ((ScopeData(i) - DarkData(i))/IntensityCal(i))
```

The Reflectance Calibration data can be used to convert the scope data into a Reflectance or Absorbance spectrum. It is not yet used for calibration purposes by Avantes. The Spectrum Correction structure is used by Avantes for stray light correction purposes starting with Library version 9.3.

3.8.4 EEPROM Structure: Stray Light Correction

The data in the Spectrum Correction array is used for correction of stray light. A calibration can be performed by Avantes, which then allows you to correct for stray light using the function AVS_SuppressStrayLight(). A few conditions must be met for a successful use of this function:

- The wavelength range determines if a stray light calibration can be done by Avantes. For most spectrometers with a wide range (300 and 600 lines/mm gratings with 75mm AvaBenches) a stray light calibration can be performed. You will receive an error message (-140) if you call AVS_SuppressStraylight() for an uncalibrated spectrometer.
- A valid dark spectrum must be available, even when using Dynamic dark.
- If Dynamic dark correction is available for the spectrometer, it will always be used in the stray light calibration at Avantes. Dynamic dark should then also be used when applying the stray light correction.
- The correction must be performed on a spectrum that has been corrected for dark. To display a corrected scope spectrum without a subtracted dark, you must therefore again add the dark values to the corrected values!
- In measurement modes that require a reference spectrum (like absorbance mode), it is important that this reference spectrum is also recorded with stray light correction.
- It is important that Irradiance calibrations should also be used consistently when it comes to stray light correction status. Do not use an irradiance calibration that was done without stray light correction on a spectrum that was corrected for stray light or vice versa.
- The multiplication factor used in the stray light calibration is 1.0
  The range allowed for this parameter is 0.0 – 4.0, where 0.0 means no correction and a higher factor than 1.0 will linearly apply a larger correction. You should generally also use a factor of 1.0 in your correction function.

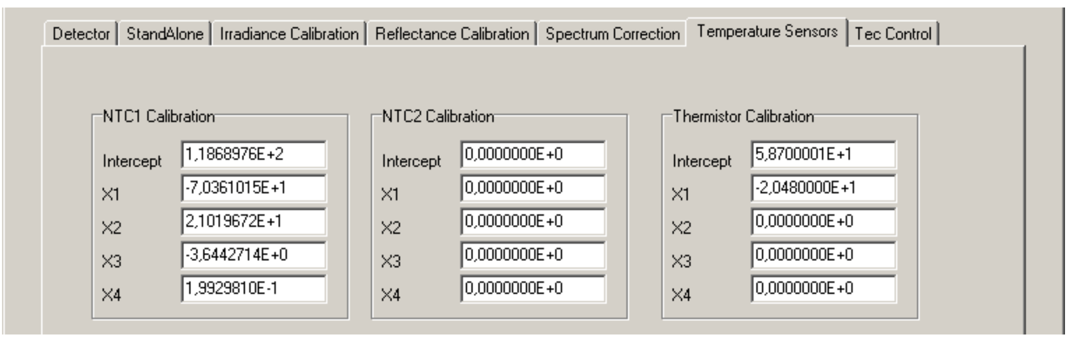
In the Qt5_demo_SLS sample program, AVS_SuppressStrayLight() is called to demonstrate SLS.

3.8.5 EEPROM Structure: Temperature Sensors

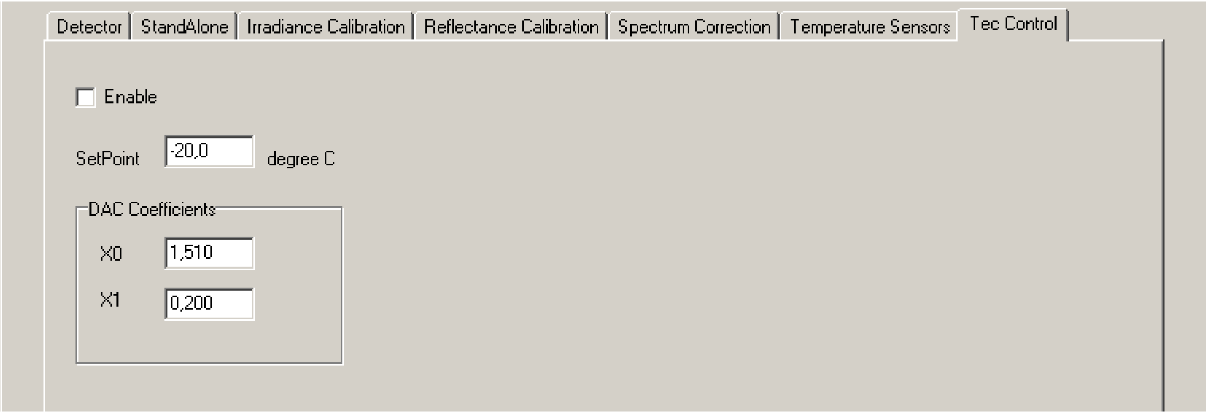
The AS5216 boards are prepared for using up to three thermistors. NTC1 is mounted on the board, NTC2 is not mounted, and the third thermistor is in the detector. The voltage level of the thermistors can be retrieved by calling the AVS_GetAnalogIn() function (see also section 2.5.26 on AVS_GetAnalogIn() and Analog IO).

The structure TempSensorType can hold the coefficients for a polynomial that converts the voltage level into a temperature.

The AS7010 has a digital temperature sensor that already transmits degrees Celsius. Therefore, the NTC1 calibration polynomial will be (0,1,0,0,0) for the AS7010. The detector thermistor calibration of the AS7010 is identical to the AS5216 one.



3.8.6 EEPROM Structure: TEC Control



The TecControl parameters are used to control the cooling of the detectors with TEC support. For these spectrometer types, the m_TecControl.m_Enable flag will be set to true. The default set point in degrees Celsius is –20 °C for the AvaSpec-NIR, and +5°C for the 2048TEC and 3648TEC (one-stage cooling), but it can be changed if needed.

It is not recommended to change the DAC polynomial (m_aFit) which has been optimized for the detector type. For recent models (AvaSpec-ULS2048-TEC and for the ASM5216 boards), the X0 and X1 coefficients in the m_aFit polynomial are 0.0, because the PID control has been entirely implemented in the firmware.

To monitor the detector temperature, use the AVS_GetAnalogIn() function, with a_AnalogInId set to 0 (see also section 2.5.26 and Analog IO). The polynomial coefficients for converting the measured voltage (U) to degrees Celsius can be found in the table below:

Spectrometer	**DetectorType**	**m_aTemperature[2]. m_aFit[0]**	**m_aTemperature[2]. m_aFit[1]**
AvaSpec-NIR-2.0/2.5TEC	SENS_HAMS9201	58.70	-20.48
AvaSpec-NIR512-2.5-HSC	SENS_HAMG9208_512	58.70	-20.48
AvaSpec-NIR256-1.7/2.2TEC	SENS_SU256LSB	56.60	-18.58
AvaSpec-NIR512-1.7/2.2TEC	SENS_SU512LDB	56.60	-18.58
AvaSpec-2048TEC	SENS_ILX554	51.4	-16.38
AvaSpec-3648TEC	SENS_TCD1304	51.4	-16.38
AvaSpec-2048x64TEC	SENS_HAMS10420_11850	51.4	-16.38
AvaSpec-HS1024x58	SENS_HAMS7031	82.15	-22.43
AvaSpec-HS1024x122	SENS_HAMS7031_11501	82.15	-22.43

These coefficients are stored in the TempSensorType structure in the EEPROM as described in section 3.8.5.

3.8.7 EEPROM Structure: ProcessControl

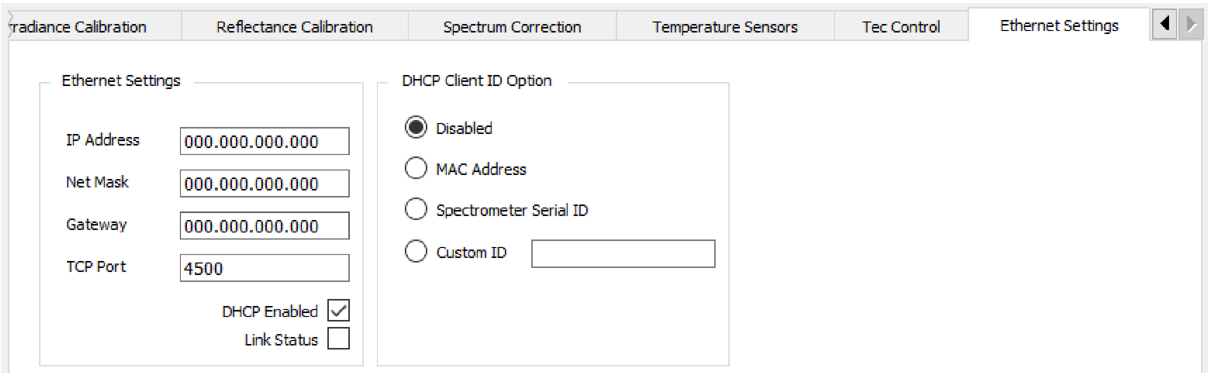
The settings in the ProcessControl structure can be used for the 2 analog and 10 digital output signals at the DB26 connector.

The analog settings can be used to store a function output range that should correspond to the 0-5V range of the analog output signals. For example, if the measured function output is expected to be in a range between 1000 and 2000, these values can be stored in the m_AnalogLow[0] and m_AnalogHigh[0] parameters. The function output can then be converted to a 0-5V analog output at pin 17 by using the range stored in EEPROM.

The digital output settings can be used as lower- and upper thresholds, to set the corresponding pins to 0 or 5V if these thresholds are exceeded.

The Process Control structure has been successfully used in applications, in which the spectrometer runs completely standalone, without a connection to a PC. Data processing is in that case done onboard by dedicated firmware and the analog and digital outputs are used to signal the function output.

3.8.8 EEPROM Structure: EthernetSettings



The Ethernet Settings are used for the AS7010. The AS7010 is shipped with the setting ‘DHCP enabled’ on, this setting needs a DHCP server to be present in your network. If you are not using a DHCP server, you can enter a static IP address, Net Mask and Gateway here, and save these values to EEPROM. Make sure to uncheck the ‘DHCP Enabled’ checkbox, when using a static IP address.

The TCP Port value is the default port number that the AvaSpec Library uses to connect to the AS7010. This value will also be returned by the spectrometer in the answer to the broadcast to all devices that is used to discover spectrometers on the network.

The Link Status value is not used.

With the DHCP Client Option the *Client Identifier*, a DHCP setting sent with DHCP option 61, of the AS7010 can be configured as follows;

- Disabled: DHCP Client Identifier is disabled and will not be sent to the DHCP server.
- MAC Address: AS7010 board MAC address is used by the DHCP client (AS7010) for option 61.
- Spectrometer Serial ID: AS7010 board serial number is used by the DHCP client for option 61.
- Custom ID: A fixed identifier is used by the DHCP client for option 61.

Please note that the *DHCP Client ID Option* is used only when the DHCP client is enabled (see *DHCP Enabled* option of the Ethernet Settings part).

3.8.9 EEPROM Structure: OemData

OEM data part is a special section which can be used by any external application for any purpose. The OemData field is meant to be used to store and read any user specific data. The AvaSpec functions AVS_GetOemParameter() and AVS_SetOemParameter() must be used for this purpose. The length of the OEM specific data field is defined in section 2.6.

The OemData section will not be overwritten by any other Avantes application, other than the *FieldUpgrade* tool provided to restore the whole device configuration of the spectrometer. Since the OemData is part of this configuration the OemData field will then also be erased.

Please note that OemDataType is part of the DeviceConfigType in EEPROM as shown in section 2.6. Precautions must be taken to prevent OemData overwrites when using AVS_SetParameter() function together with AVS_SetOemParameter().

4 Appendix A: USB Driver Installation on Windows

The AvaSpec Library uses the Microsoft WinUSB driver, both for 32-bit and 64-bit Windows Operating Systems.

Until May 2011, on 32-bit Windows systems, an Avantes kernel driver was used as the standard USB driver. On 64-bit Windows, the WinUSB driver has always been used.

Support for the WinUSB driver on 32-bit Windows O/S has been implemented in as5216.dll version 1.8 and later.

Installing the WinUSB driver is now the standard on all Windows O/S, except for ancient Windows versions that lack WinUSB driver support (like Windows 2000 and Windows98). On these versions, you will need to use an older version of the Library, like as5216.dll version 2.2.

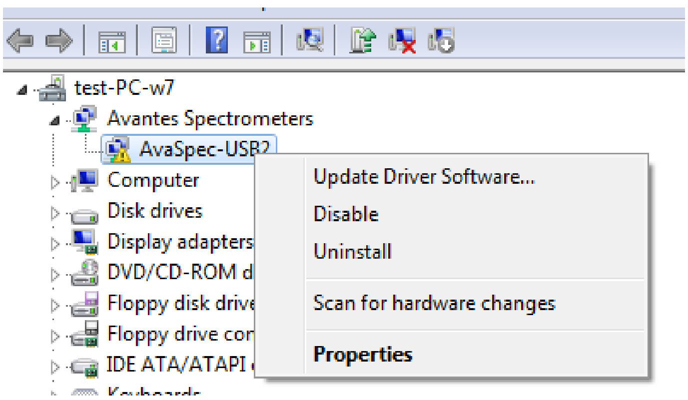
In this Appendix, some compatibility issues will be described that may occur after upgrading to WinUSB:

1. After installing the WinUSB driver and connecting the AvaSpec-USB2 spectrometer, the spectrometer cannot be found.
2. After installing the WinUSB driver, the spectrometers first worked fine, and the Device Manager shows a proper installation of the WinUSB driver. However, after installing some application software the spectrometer cannot be detected anymore. Also the sample programs shipped with as5216.dll v 1.9 cannot detect an AvaSpec-USB2 anymore. The Device Manager still shows a proper WinUSB driver installation.
3. After installing the WinUSB driver, the spectrometer runs fine with the sample programs shipped with as5216.dll v 1.9, but not with other application software.

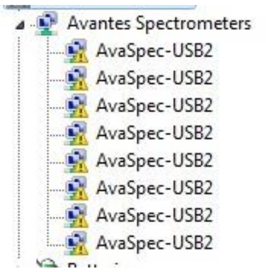
A1: Spectrometer cannot be found after update to WinUSB driver

After connecting the spectrometer to a USB port of your PC, Windows will install the device driver. If all goes well, this will be displayed in the lower right corner of your screen with the message ‘Device driver software installed successfully’.

We have seen instances, where this message will not appear, and where it is necessary to open the Device Manager, to let Windows find the installed driver files. To open the Device Manager, right click ‘Computer’ in Windows Start menu, and select ‘Properties’. Then click the “Device Manager” option.



The ‘Avantes Spectrometers’ entry will show a yellow triangle with an exclamation mark. Right-click the AvaSpec-USB2 line and select the “Update Driver Software” option, as shown in the figure at the left. In the next dialog, select “Search automatically for updated driver software”. Windows should now find the correct files and install the driver software.



In multichannel spectrometer systems, it may be needed to repeat this step several times (once per channel).

A2: Device Manager shows a proper WinUSB driver installation, but AvaSpec-USB2 cannot be detected anymore

If the WinUSB driver has been installed properly, the Device Manager will display the connected devices without the yellow triangle with exclamation mark. The sample programs that are shipped with this version can be executed and everything runs well. However, after installing some application software, it is possible that the spectrometer cannot be detected anymore. The sample programs cannot detect the AvaSpec-USB2 spectrometers either. One reason can be that the application software uses an old as5216.dll version 1.7 or earlier, as will be described below under A3.

The problem can also be caused by the installation program that installed the application software. When installing AvaSoft version 7.6.0 or earlier versions, the Avantes kernel driver (AVSUSB2.sys) will be installed, without uninstalling the WinUSB driver. The as5216.dll will try to communicate through the most recently installed driver (avsusb2.sys), while the WinUSB driver is the one that is active in the Device Manager. The problem can be easily solved by reinstalling the most recent application software (AvaSoft 7.6.1 or later), or reinstalling the as5216-dll package. In the driver selection dialog, select the (recommended) WinUSB driver. The same situation may occur when the Avantes kernel driver is installed by other application software (AvaSoft-Thinfilm-USB2, AvaSoft-Raman-USB2, or third party applications).

A3: After installing the WinUSB driver, the sample programs and AvaSoft are running fine, but other application software cannot detect the AvaSpec-USB2 spectrometer anymore.

Most likely, the as5216.dll version used by the other application software does not support the WinUSB driver. The WinUSB driver is supported by the as5216.dll since version 1.8.0.0.

5 Appendix B: Using the Ethernet Spectrometer

This section describes some details to avoid common pitfalls that can occur when using the Ethernet interface of the AS7010 spectrometer.

B1: DHCP vs. Using Static IP Addresses

The spectrometers are shipped with DHCP enabled. This means that they will be assigned a unique IP address in the correct range if you connect them to a network on which a DHCP server is running. If you connect the spectrometer to your office network for the first time, please ensure that a DHCP server is present on that network.

The spectrometer can also be used with a static IP address configured. You can change the network settings of the spectrometer with the *IP Settings AS7010 utility* that is e.g. distributed with AvaSoft 8. When using the spectrometer with a static IP address please make sure that the used IP address is in the same range as your host PC.

When using the spectrometer within a local network (a network with only one host (PC) and one or more Ethernet spectrometers with static IP addresses), you will have to change the IP settings of your PC from DHCP to static as well. This is also described in the *IP Settings AS7010 manual* in more detail. Of course, you can also use DHCP enabled spectrometers within the local network, provided that there is a DHCP server running on your host PC. There are several simple to use and freeware DHCP servers for Windows, such as one available from *www.dhcpserver.de*.

Please consult your network administrator for the availability of any DHCP server on your network or for using static IP addresses on the spectrometers. It is very important that the Ethernet spectrometers are configured with the right network settings, since otherwise major network problems can occur.

B2: Cables

You can use standard Ethernet patch cables to connect the spectrometer to your network. If you use a direct connection to your PC, make sure you have a GigE network connection, if you want to use a standard Ethernet patch cable. If your PC has an older Fast Ethernet connection (100 Mbps), then you will need a cross-connect cable.

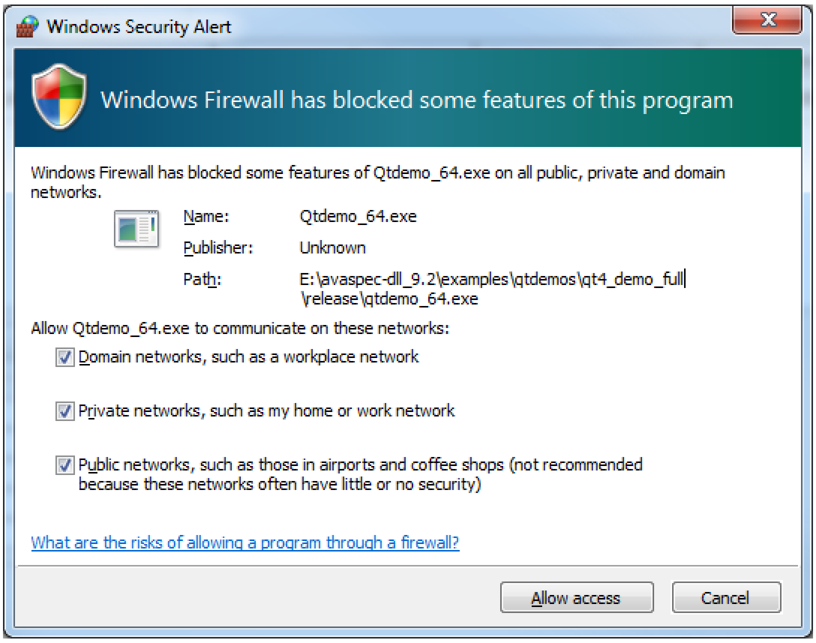
B3: Routers

The Library uses a UDP broadcast to identify the spectrometers on the network. It is sent from each network adapter that is found in the host PC. Most layer 3 switches and routers will not allow this broadcast to pass, which will stop the Library from working. Be sure that your router will forward UDP broadcast packets to all ports within your local network. If there are connection problems, please also test a direct PC connection to make sure your router is not the cause.

B4: Firewall

The host firewall must allow both incoming and outgoing connections to the spectrometer. The spectrometer tries to connect to its host at TCP port numbers 4500 and 2400. Within your firewall settings, both ports should be open for inbound and outbound.

If the Windows Firewall is enabled, then a dialog will appear if you first use a program that will access the network:



Make sure you check all three boxes and press the ‘Allow access’ button, to allow the program access. Switching the firewall off completely will of course also work, but may not be advisable. You may have to restart the program that uses the Library to have things work correctly.

B5: AvaSpec Library / Firmware Version Conflicts

With AS7010 firmware version 1.3, the discovery protocol has been changed. If your firmware is older than version 1.3, you will not be able to use the latest versions of the AvaSpec Library (9.2 and later). Please contact Avantes to upgrade the spectrometer firmware to the latest version with an update utility.

B6: Connecting both Interfaces at the Same Time

If you connect both the USB and Ethernet interfaces, then the USB interface will have priority. If, however, an Ethernet connection has already been made, then plugging in a USB cable will not break the Ethernet connection.

6 Appendix C: Notes on Recompiling the Qt Demos on Windows

Installing Visual Studio

Microsoft offers Visual Studio in a free 'Community' version, or a paid 'Professional' version. Recent editions of the free version are fully equipped to be used with Qt. You can download these versions from www.visualstudio.com. Versions 2013 and 2015 are available as a small web-installer or a large DVD image.
Versions 2017 and 2019 are only available as a web-installer, although you can generate a customized offline installer by running the web-installer with a special syntax. With versions 2017 and 2019, you will only need to install the '.NET desktop development' and the 'Desktop development with C++' for .NET and Qt5 development.

Please note that as of VS2017 Visual Studio no longer supports development of .NET Windows Forms applications using C++. Please look at the full Qt sample instead. It uses the Qwt charting library. (We have also re-included an MFC sample, now compiled with VS2017. This sample does not use Qt.)

Installing Qt and Qwt

To install the framework Qt5 on Windows, either run the online installer, which is a small download from the Qt site, or the offline installer, which is a very large download.

Select the version that corresponds with your compiler and 32/64-bit choice, e.g. either msvc2015 64-bit, msvc2017 32-bit or msvc2017 64-bit.

Select Qt Creator under 'Tools'.

You can either work with Qt Creator, or with Visual Studio.

If you want to work with Visual Studio, download and install the Visual Studio Add-In from the Qt site. This file is called e.g. 'qt-vsaddin-msvc2017-2.3.0.vsix'.

To install the charting library Qwt on Windows, download and unpack e.g. 'qwt-6.1.4.zip' from SourceForge.

To recompile and install, open e.g. the 'x64 Native Tools Command Prompt for VS 2017' from the Windows start menu. Windows will reply with:

'Setting up environment for Qt usage ... Remember to call vcvarsall to complete environment setup!'

The vcvars batch file is needed in order to let the system find the correct Microsoft compiler files.

When using VC2017 to build a 64 bit version of Qwt enter the following lines:

```
cd c:\Program Files (x86)\Microsoft Visual Studio\2017\Professional\VC\Auxiliary\Build\
vcvars64.bat
cd c:\Qwt\vs2017\qwt-6.1.4\
c:\Qt\Qt5.12.3\5.12.3\msvc2017_64\bin\qmake qwt.pro
nmake
```

Recompiling the Qt Demos

The Qt5 samples contain both .pro project files for use with qmake, and .vcxproj files for use with Visual Studio 2017. If you use a different version of Visual Studio, do not attempt to use and auto-update the .vcxproj project files. Please generate new ones from the .pro file with the Qt Visual Studio Add-in.

The path to the Qwt include files and link libraries must be entered correctly in the project. Depending on your disk configuration, you must change the paths to the Qwt files in the project.

The paths are present in the Property Pages, under 'C/C++', 'Additional Include Directories' and 'Linker', 'Additional Library Directories'.

Change e.g. 'E:\Qwt\vs2017\qwt-6.1.4\lib' to 'C:\Qwt\vs2017\qwt-6.1.4\lib'.

Please note that these settings can be present in the project file up to FOUR times, depending on Configuration ('Debug' or 'Release') and Platform ('Win32' or 'x64'). If you use a different Visual Studio version, you can save some time by correcting the paths in the .pro file before importing it.

7 Appendix D: 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:

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"
```

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