AS-5216-DLL

Interface Package for 32 bit Windows Applications

Version 1.4.0.0

USER'S MANUAL
October 2007

Avantes B.V.

Soerense Zand 4a
NL-6961 LL Eerbeek
The Netherlands
Tel: +31-313-670170
Fax: +31-313-670179

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

Software License

0 INSTALLATION 6

1 VERSION HISTORY 8

1.1 New in version 1.4.0.0 8

1.2 New in version 1.3.0.0 9

1.3 New in version 1.2.0.0 9

1.4 New in version 1.1.0.0 9

1.5 New in version 1.0.0.0: as5216.dll versus as161.dll 10
1.5.1 Data acquisition 10
1.5.2 Synchronization in Multichannel systems 11
1.5.3 Laser control and integration time delay, e.g. for LIBS 11
1.5.4 USB2 platform specific functions 12

2 AS-5216-DLL DESCRIPTION 13

2.1 Interface overview 13

2.2 Usage AS-5216-DLL 13

2.3 Exported functions 14
2.3.1 AVS_Init 14
2.3.2 AVS_Done 14
2.3.3 AVS_GetNrOfDevices 14
2.3.4 AVS_GetList 15
2.3.5 AVS_Activate 15
2.3.6 AVS_Deactivate 15
2.3.7 AVS_Register 16
2.3.8 AVS_PrepareMeasure 16
2.3.9 AVS_Measure 16
2.3.10 AVS_GetLambda 17
2.3.11 AVS_GetNumPixels 17
2.3.12 AVS_GetParameter 18
2.3.13 AVS_PollScan 18
2.3.14 AVS_GetScopeData 18
2.3.15 AVS_GetSaturatedPixels 19
2.3.16 AVS_GetAnalogIn 19
2.3.17 AVS_GetDigIn 20
2.3.18 AVS_GetVersionInfo 20
2.3.19 AVS_GetFileSize 21
2.3.20 AVS_GetFile 21
2.3.21 AVS_GetFirstFile 22
2.3.22 AVS_GetNextFile 22
2.3.23 AVS_DeleteFile 23
2.3.24 AVS_GetFirstDirectory 23
2.3.25 AVS_GetNextDirectory 23
2.3.26 AVS_DeleteDirectory 24
2.3.27 AVS_SetDirectory 24

2.3.28 AVS_SaveSpectraToSDCard 25
2.3.29 AVS_SetParameter 26
2.3.30 AVS_SetAnalogOut 26
2.3.31 AVS_SetDigOut 26
2.3.32 AVS_SetPwmOut 27
2.3.33 AVS_SetSyncMode 28
2.3.34 AVS_StopMeasure 28
2.3.35 AVS_SetPrescanMode 29
2.3.36 AVS_UseHighResAdc 30

2.4 Data Elements 31
2.4.1 Return value constants 38
2.4.2 Windows messages 40

3 EXAMPLE SOURCE CODE 41

3.1 Initialization and Activation of a spectrometer 41

3.2 Starting a measurement 42
3.2.1 Measurement structure: Start- and Stoppixel 42
3.2.2 Measurement structure: Integration Time 43
3.2.3 Measurement structure: Integration Delay 43
3.2.4 Measurement structure: Number of Averages 43
3.2.5 Measurement structure: Dynamic Dark Correction 43
3.2.6 Measurement structure: Smoothing 44
3.2.7 Measurement structure: Saturation Detection 45
3.2.8 Measurement structure: Trigger Type 45
3.2.9 Measurement structure: Control Settings 47

3.3 Measurement result 48

3.4 Digital IO 49

3.5 Analog IO 49

3.6 EEProm 50
3.6.1 EEProm structure: Detector Parameters 50
3.6.2 EEProm structure: Standalone Parameters 54
3.6.3 EEProm structure: Irradiance, Reflectance Calibration and Spectrum Correction 55
3.6.4 EEProm structure: Temperature Sensors 56
3.6.5 EEProm structure: Tec Control 56
3.6.6 EEProm structure: ProcessControl 57

0 Installation

AS-5216-DLL is a 32-bit driver interface package and can be installed under the following operating systems:

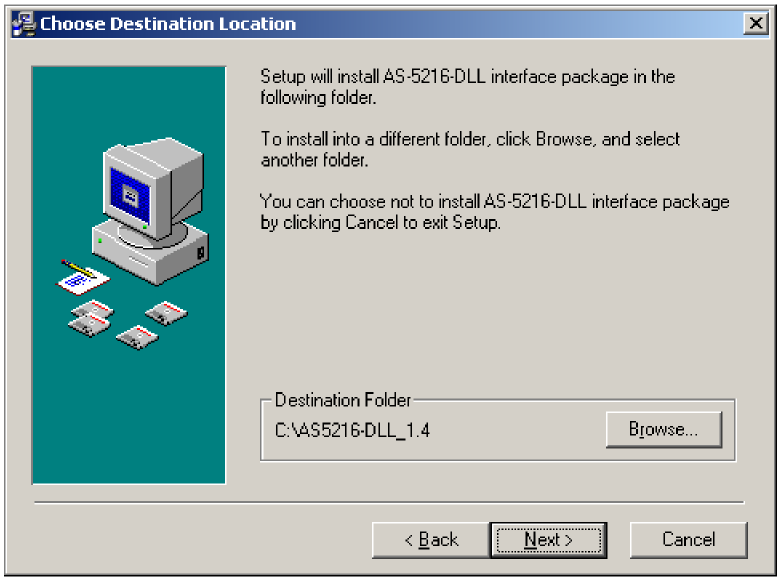
- Windows 95/98/Me
- Windows NT/2000/XP/Vista

The installation program can be started by running the file “setup.exe” from the CD-ROM.

Installation Dialogs

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 AS5216-DLL software can be changed. The default destination directory is C:\AS5216-DLL_1.4. If you want to install the software to a different directory, click the Browse button, select a new directory and click OK. If the specified directory does not exist, it will be created.

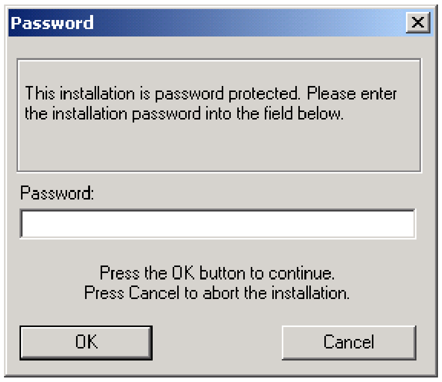

Choose Destination Location

Setup will install AS-5216-DLL interface package in the following folder.

To install into a different folder, click Browse, and select another folder.

You can choose not to install AS-5216-DLL interface package by clicking Cancel to exit Setup.

Destination Folder: C:\AS5216-DLL_1.4


Password

This installation is password protected. Please enter the installation password into the field below.

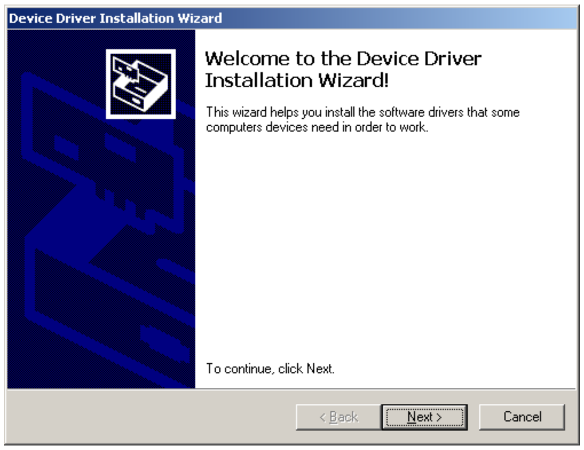
Password:

Press the OK button to continue.
Press Cancel to abort the installation.

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

Avantes6961LL4a

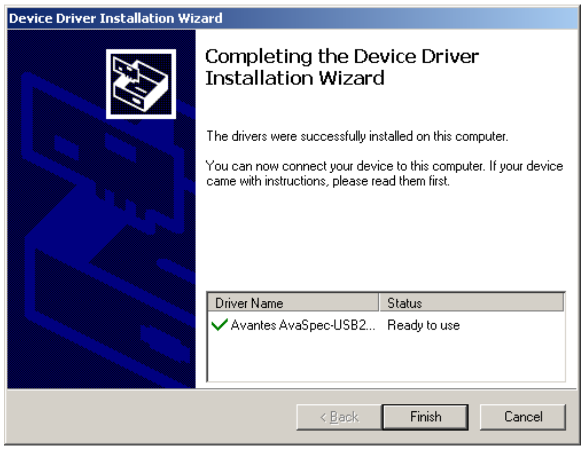
During this installation, the installation program will check if the most recent USB driver has been installed already at the PC. If no driver is found, or if the driver needs to be upgraded, the Device Driver Installation Wizard is launched automatically, click Next. If the Operating System is Windows Vista, it will display a message that it can't verify the publisher of the driver software, select "Install this driver software anyway".



Welcome to the Device Driver Installation Wizard!

This wizard helps you install the software drivers that some computers devices need in order to work.

To continue, click Next.



Completing the Device Driver Installation Wizard

The drivers were successfully installed on this computer.

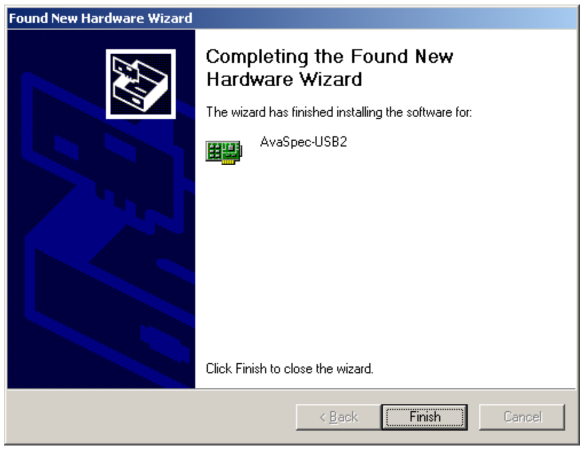
You can now connect your device to this computer. If your device came with instructions, please read them first.

Driver Name	Status
✓ Avantes AvaSpec-USB2...	Ready to use

After the drivers have been installed successfully, the dialog at the right is displayed, click Finish. 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. Windows XP will display the "Found New Hardware" dialog. Select the (default) option to install the software automatically, and click next. After the Hardware Wizard has completed, the following dialog is displayed under Windows XP:



Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

AvaSpec-USB2

Click Finish to close the wizard.

Click Finish to complete the installation.

Please note that if the spectrometer is Connected to another USB port to which it has not been connected before, the "Found New Hardware Wizard" will need to install the software for this port as well. For this reason, this Wizard will run "NrOfChannel" times for a multichannel AvaSpec-USB2 spectrometer system. This happens because inside the housing, the USB ports for each spectrometer channel are connected to a USB-Hub.

Windows Vista will install the driver automatically, without displaying the "Found New Hardware Wizard" dialogs.

Launching the software

This AS-5216-DLL manual and several example programs can be started from the Windows Start Menu. The source code of the example programs can be found in the Examples folder.

1 Version History

This section will be used to describe the new features in the as5216.dll, compared to the previous versions.

1.1 New in version 1.4.0.0

- Implementation of the StoreToRam function, which allows the storing of scans at high speed (as fast as 1.1 msecs per scan for the AvaSpec-2048-USB2, and 0.1 msecs per scan for the AvaSpec-102-USB2) in the spectrometer, without the overhead of USB communication. About 4MB of storage is available, which allows for 1013 full spectra with the AvaSpec-2048-USB2 and 19784 for the AvaSpec-102-USB2, or a lot more if the pixelrange is reduced by selecting the start- and stoppixel. StoreToRam requires firmware version 0.20 or later. A firmware upgrade utility can be downloaded from our website.
- Implementation of directory support for the Secure Digital Card.
- In a few occasions there have been problems with detecting and/or activating AvaSpec-USB2 spectrometers under Windows Vista. The reason for this is that, according to MicroSoft, "a USB device takes a long time to resume from selective suspend mode on a Windows Vista-based computer that uses UHCI (Universal Host Controller Interface) USB controllers. In the as5216.dll version 1.4, a workaround has been implemented to solve this problem.
- New sample programs for Visual C++ 2005, Delphi and LabView
  - The Visual C++ 2005 sample program has been created in the Express version.
  - A few sample programs in Delphi have been added: besides the comprehensive sample program that was already available in earlier as5216.dll versions, 3 new sample programs have been added:
    - A multichannel sample program in which up to 16 spectrometer channels can run simultaneously in SYNC mode or ASYNC mode.
    - An sdcard sample program which demonstrates how to save spectra to an onboard sdcard
    - A simple program with only a few lines of code which demonstrates the basic data acquisition for a single channel AvaSpec-USB2 spectrometer.
  - The LabView sample programs have been updated to LabView version 8.2 (earlier versions can be obtained on request). There are 4 sample programs:
    - a comprehensive program for a single channel AvaSpec-USB2, which also includes subvi's for all functions in the as5216.dll
    - a program that illustrates the use of Windows Messaging in LabView.
    - a simple sample program that uses AVS_PollScan instead of Windows Messaging
    - 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.
- In this manual, examples have been added for using the function AVS_UseHighResAdc in combination with nonlinearity correction and/or irradiance calibration, see section 3.6.1 under "Using the nonlinearity correction polynomial in combination with the 16bit ADC Counts range" and section 3.6.3 under "How to convert ScopeData (A/D Counts) to a power distribution [µWatt/(cm^2.nm)]".

1.2 New in version 1.3.0.0

- Windows Vista support
- New Sample programs in Visual C# and Visual Basic.NET 2001. These sample programs can also be used in more recent .NET versions (2005) in which case the Visual Studio Conversion Wizard will convert the project to the new version. Note that for Visual Basic .NET, there was already a VB .NET 2005 sample program available.
- ProcessControl Structure added for standalone functionality (see section 3.6.6)
- Stability issues solved. Some spectrometer types (mainly the AvaSpec-102-USB2) have shown a lock up in continuous measurements over a long period at short integration time. Another problem that showed up very rarely, concerns multichannel spectrometers running in synchronization mode. Both problems have been solved in as5216.dll version 1.3.
- Support for the AvaSpec-2048x14 High UV-sensitivity back-thinned CCD Spectrometer. The new detector type used in this spectrometer is the HAMS9840 and is supported in as5216.dll version 1.3 and all the sample programs.
- The new function AVS_UseHighResAdc has been added to enable the full 16-bit ADC range which is available with a 16-bit ADC on the as5216 board as of revision 1D. See also section 2.3.36.
- A minor bug in the smoothing routine has been solved.
- The sample program in Visual Basic 6.0 has been modified because it crashed after running continuously for a number of hours. The cause was found to be in the VB6 “Timer” function that was used to show some statistics about the measurement speed. By eliminating the Timer function from the sample program, this problem was solved. Feedback from VB6 programmers who know how to use the Timer function (or an equivalent) without crashing the application is appreciated.

1.3 New in version 1.2.0.0

Visual Basic 6.0 developers may have noticed that the programs developed with as5216.dll v.1.1 or earlier and Visual Basic 6.0 are stable, but the Visual Basic 6.0 Integrated Development Environment (IDE) was not. Running the program from the VB6 IDE, caused the IDE to close down without saving any changes, as soon as the program was closed. To solve this problem, a special as5216.dll version (1.2.0.1) has been created which can be used in the VB6 IDE to develop and debug your programs. AS5216.dll version 1.2.0.1 will be installed in the VB6 example folder, as well as a readme.txt file with recommendations for redistributing programs developed with Visual Basic 6.0.
Furthermore, a parameter structure has been added to the EEProm to control the TEC cooling for the NIR spectrometer (AvaSpec-256-NIR2.2). More detailed information about this TEC Control structure can be found in section 3.6.5.
The last change in version 1.2 is only in this manual. For spectrometers that have been calibrated for irradiance measurements, the IrradianceType structure contains data that can be used to convert the ScopeData (A/D Counts) to an irradiance spectrum. Section 3.6.3 in this manual decribes more in detail how this can be done.

1.4 New in version 1.1.0.0

The as5216.dll version 1.1 includes one new function (AVS_SetPrescanMode) which can be used for the AvaSpec-3648 spectrometer. Furthermore, a lower and upper limit has been added to the nonlinearity polynomial to avoid incorrect correction for very low and/or high counts. Finally, example programs with source in LabView (7.1), Visual C++ (6.0), Visual Basic (6.0) and Visual Basic .NET 2005 (2.0) have been added to the already existing examples in Delphi (6.0) and Borland

C++ (5.0). The AVS_SetPrescanMode function for the AvaSpec-3648 is described in section 2.3.35, and detailed information about how to apply the nonlinearity correction can be found in section 3.6.1

1.5 New in version 1.0.0.0: as5216.dll versus as161.dll

Although there is no previous version for as5216.dll v1.0, a comparison can be made for programmers who have used the as161.dll to write application software for the USB1 platform AvaSpec spectrometers.

A number of improvements have been implemented in the as5216.dll when comparing the functions to the as161.dll. These improvements can be grouped into the following categories:

- Data acquisition
- Synchronization in multichannel systems
- Laser control and integration time delay, e.g. for Laser Induced Breakdown Spectroscopy
- USB2 platform spectrometer

These categories will be described in sections 1.5.1 to 1.5.4

1.5.1 Data acquisition

Just like with the as161.dll, 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. The following improvements have been realized in as5216.dll for the USB2 platform spectrometers:

1. **Starting a measurement**. In continuous mode, the USB1 platform spectrometers always run continuously, also if no measurement requests are posted. A call to AVS_Measure in as161.dll results in returning the first available scan (see section 2.2 of the as161.dll manual). The USB2 platform spectrometers are in idle mode if not scanning, and will start a scan if a measurement request (AVS_Measure) from the as5216.dll is received.
2. **Number of measurements**. The AVS_Measure function in as161.dll always results in one single scan. A spectrum is only sent to the PC if a measurement request is received. The AVS_Measure function in as5216.dll includes a “nrms” argument which specifies the number of measurements the spectrometer should perform after one measurement request.
3. **Stopping a measurement**. A measurement in as161.dll cannot be interrupted. After a measurement request, the application must wait for the response before changing measurement parameters or sending other commands to the spectrometer. The as5216.dll includes a function AVS_StopMeasure that can be called to interrupt a measurement request.
4. **Spectrometer not blocked while measurement is pending**. With the USB1 platform spectrometers, the spectrometer is blocked for receiving commands as long as a measurement is pending. A measurement is pending between the call to AVS_Measure and the DATA_READY message from the as161.dll to the application. The USB2 platform spectrometers are not blocked from receiving commands while a measurement is pending. This means that you can e.g. control the digital and analog IO ports while a measurement is pending.
5. **Measurement parameters**. There are a lot of parameters involved that determine the result of a scan such as integration time, number of averages, smoothing, pixelselection, dark correction etc... In the as161.dll, a lot of different functions are used to set these parameters: integration time and averaging are set in AVS_Measure, smoothing in AVS_SetSmoothing, Pixelselection in AVS_SetPixelSelection, etc… The as5216.dll uses a measurement structure which includes all measurement parameters and uses only one function (AVS_PrepareMeasure) to send these parameters to the spectrometer.

6. **External Trigger**. The only setting in the as161.dll for external trigger functionality is to switch this mode on or off by calling the function AVS_SetExternalTrigger. In external trigger mode, the USB1 platform spectrometer will start one single scan on the rising edge of the TTL pulse that is sent to the external trigger input port of the spectrometer. In the as5216.dll, the USB2 platform spectrometer can be set into external trigger mode by setting the trigger mode parameter into hardware trigger mode. If the trigger type parameter is set to "EDGE", the number of measurements to perform after receiving one TTL pulse can be specified by setting the nrms parameter in the AVS_Measure function. If the trigger type parameter is set to "LEVEL", the spectrometer will keep scanning as long as the TTL input signal is HIGH, and when the signal becomes LOW, it will return the average scan over all scans that were performed during the HIGH time period.
7. **Integration time**. The integration time in the as161.dll can be set with a 1 ms resolution. In the as5216.dll, a 0.01 ms (10 µs) resolution is used for the integration time.
8. **Timestamp.** The AVS_GetScopeData function in as5216.dll includes a timestamp in 10 µs resolution ticks generated by the microcontroller, which can be used to measure the time between two consecutive (and processed) scans very accurately.

1.5.2 Synchronization in Multichannel systems

There is a major difference between the USB1 and USB2 platform multichannel systems. The USB1 platform multichannel systems always needs the same detectortype for each channel. Also, the integration time and number of averages in a measurement request is equal for all channels.
With the multiple usb support in the as161.dll (v.1.5 and later), spectrometers with different detectors and at different integration time or average can run simultaneously, but in that case there is no synchronization between these spectrometers.
With the USB2 platform multichannel systems, the advantage of the multiple USB implementation (up to 127 spectrometers, possibility of using different detectors, integration time and averaging per channel) has been combined with the advantage of the as161 multichannel systems (synchronization). All USB2 platform spectrometers can be connected by a SYNC cable. In syncmode, one spectrometer is configured as "Master", all other ("slave") spectrometers are set into "Trigger by SYNC" mode. After a measurement request for the slave spectrometer(s), these spectrometers will wait until they receive the trigger signal on the SYNC cable. This SYNC signal will be started if a measurement request is posted for the Master spectrometer.

1.5.3 Laser control and integration time delay, e.g. for LIBS

If the AvaSpec-2048FT (USB1 platform) is set in external hardware trigger mode, an external trigger pulse results in an output signal (pulse width 15 µs) at pin2 of the DB15 connector (DO2), about 1.3 µs after the trigger pulse was received. The pulse at DO2 can be used to fire a laser in a LIBS application. The function AVS_SetIntegrationDelay can be used to specify an integration time delay, which is related to DO2.
With the AvaSpec-2048-USB2 and AvaSpec-3648-USB2, this feature has been improved at the following points:
1. **Not limited to external trigger mode**. The output signal and integration delay can be generated in external trigger mode, but also in "normal" (software trigger) mode.
2. **Multiple measurements**. The number of measurements can be set by the nrms parameter, in the AVS_Measure function.
3. **Pulse width**. The "laserpulse" width at pin 23 of the DB26 connector can be set by the user, between 0 and 1 ms (21 nanosec steps).

4. **Laser Delay**. The 1.3 µs period between receiving a trigger (measurement request in software trigger mode or TTL pulse in hardware trigger mode) can be delayed from 1.3µs to 89 sec (21ns steps).

1.5.4 USB2 platform specific functions

The functions that have been added to as5216.dll to support the new hardware features in the USB2 platform spectrometers, can be grouped into the following categories:

- Analog IO
- Digital IO and Pulse Width Modulation
- SDCard support
- Eeprom IO

1. **Analog IO**. The USB2 platform 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. Moreover, a number of onboard analog signals can be retrieved with the AVS_GetAnalogIn function. One of these onboard signals is an NTC thermistor which can be used for onboard temperature measurements.
2. **Digital IO and Pulse Width Modulation**. The USB2 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 ouput 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
3. **SDCard support**. If the spectrometer was ordered with an SDxxx card, the function AVS_SaveSpectraToSDCard can be used to save spectra at the SDCard. To access the files that are saved at the SDCard, the functions AVS_GetFileSize, AVS_GetFile, AVS_GetFirstFile, AVS_GetNextFile and AVS_DeleteFile can be used.
4. **Eeprom IO**. With the USB1 platform spectrometers, it is also possible to read/write a number of parameters from/to Eeprom with the as161.dll, such as start- and stoppixel (AVS_GetStartStopPixel and AVS_SetStartStopPixel), wavelength calibration coefficients (AVS_GetWLCoef and AVS_SetWLCoef), gain (AVS_GetGain and AVS_SetGain) and offset (AVS_GetOffset and AVS_SetOffset). The Eeprom for the USB2 spectrometers has a lot more memory available to store all kind of parameters. These parameters have been defined in the DeviceConfigType structure (see section 2.4). The functions AVS_GetParameter and AVS_SetParameter in as5216.dll can be used to read/write the DeviceConfigType structure from/to Eeprom.

2 AS-5216-DLL description

2.1 Interface overview

The interface from the PC to the DLL 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 AS-5216-DLL

The DLL uses a single pair of open and close functions (AVS_Init() and AVS_Done()) that have to be called by an application. As long as the open function is not yet called or not 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.
The close function (AVS_Done()) closes the communication port(s) and releases all internal data storage.

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

- internal data read functions, which read device configuration data from the internal DLL storage.
- blocking control functions which send a request to the device and wait till 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 initialised it should select the spectrometer(s) it wants to use. Therefore, the following steps have to be taken:

1. Call AVS_GetNrOfDevices to determine the number of attached devices
2. Allocate buffer to store identity info (RequiredSize = NrDevices * sizeof(AvsIdentityType))
3. Call AVS_GetList with the RequiredSize and obtain the list of connected spectrometers
4. Select the spectrometers you want to use with AVS_Activate
5. Register a notification window handle with AVS_Register to detect device attachment/removal

2.3 Exported functions

2.3.1 AVS_Init

Function:	**int AVS_Init**
	(
	short a_Port
	)
Group:	Blocking control function
Description:	Opens the communication with the spectrometer and initialises internal data structures
Parameters:	a_Port: id. of port to be used:
	-1: use auto-detect of USB or COM port
	0: use USB port
	1: use COM1 port
	2: use COM2 port
	3: use COM3 port
	4: use COM4 port
	etc….
Return:	On success, number of connected device
	On error, ERR_DEVICE_NOT_FOUND

2.3.2 AVS_Done

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

2.3.3 AVS_GetNrOfDevices

Function:	**int AVS_GetNrOfDevices**	
	(	
	void	
	)	
Group:	Blocking control function	
Description:	Internally checks the list of connected devices and returns the number of devices currently attached.	
Parameters:	None	
Return:	> 0:	number of devices in the list
	0:	no devices found

2.3.4 AVS_GetList

Function:	**int AVS_GetList**	
	(	
	unsigned int	a_ListSize,
	unsigned int*	a_pRequiredSize,
	AvsIdentityType*	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
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

2.3.5 AVS_Activate

Function:	**AvsHandle AVS_Activate**	
	(	
	AvsIdentityType*	a_pDeviceId
	)	
Group:	Blocking control function	
Description:	Activates selected spectrometer for communication and reads device configuration data from Eeprom.	
Parameters:	On success:	AvsHandle, handle to be used in subsequent function calls
Return:	On error:	INVALID_AVS_HANDLE_VALUE

2.3.6 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
Return:	true:	device successfully closed
	false:	device identifier not found

2.3.7 AVS_Register

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	
Parameters:	a_hWnd:	Application window handle
Return:	true:	Registration successful
	false:	registration failed or function not supported on OS

2.3.8 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
	a_pMeasConfig:	pointer to structure containing measurement configuration
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND
		ERR_INVALID_DEVICE_ID
		ERR_INVALID_PARAMETER
		ERR_INVALID_PIXEL_RANGE
		ERR_INVALID_CONFIGURATION (invalid fpga type)
		ERR_TIMEOUT

2.3.9 AVS_Measure

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

	a_hWnd	window handle to notify application measurement result data is available. The DLL sends a message to the window with command WM_MEAS_READY, with SUCCESS or INVALID_MEAS_DATA as WPARM value and a_hDevice as LPARM value.
	a_Nmsr	number of measurements to do after one single call to AVS_Measure (-1 is infinite)
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER ERR_INVALID_STATE (measurement already pending)

2.3.10 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 DLL during the AVS_Activate() procedure. The DLL does not test if a_pWaveLength is correctly allocated by the caller!	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate
	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

2.3.11 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 DLL during the AVS_Activate() procedure.	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate
	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

2.3.12 AVS_GetParameter

Function:	**int AVS_GetParameter**	
	(	
	AvsHandle	a_hDevice,
	unsigned int	a_Size,
	unsigned int*	a_pRequiredSize,
	DeviceConfigType*	a_pData
	)	
Group:	Internal data read function.	
Description:	Returns the device information of the spectrometer. This information is stored in the DLL during the AVS_Activate() procedure.	
Parameters:	a_hDevice,	device identifier returned by AVS_Activate
	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)

2.3.13 AVS_PollScan

Function:	**int AVS_PollScan**	
	(	
	AvsHandle	a_hDevice
	)	
Group:	Internal data read function	
Description:	Determines if new measurement results are available	
Parameters:	a_hDevice::	device identifier returned by AVS_Activate
Return:	On success:	0: no data available
		1: data available
		>1: When using the StoreToRam function, the return value signals the number of scans available for retrieval.
	On error:	ERR_DEVICE_NOT_FOUND
		ERR_INVALID_DEVICE_ID

2.3.14 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 DLL does not check the allocated buffer size!	

Parameters:	a_hDevice,	device identifier returned by AVS_Activate
	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)

2.3.15 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
	a_pSaturated	array of chars (each char indicates if saturation occurred for corresponding pixel), 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)
		ERR_OPERATION_NOT_SUPPORTED
		ERR_OPERATION_NOT_ENABLED

2.3.16 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
	a_AnalogInId	identifier of analog input
		0 = thermistor on optical bench (NIR2.2)
		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

	a_pAnalogIn:	pointer to float for analog input value [Volts]
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)

2.3.17 AVS_GetDigIn

Function: **int AVS_GetDigIn**

```
(
AvsHandle          a_hDevice,
unsigned char      a_DigInId,
unsigned char*     a_pDigIn
)
```

Group:	Blocking control function.	
Description:	Returns the status of the specified digital input	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate
	a_DigInId:	identifier of digital input (1 – 3) 0 = DI1 = Pin 24 at 26-pins connector 1 = DI2 = Pin 7 at 26-pins connector 2 = DI3 = Pin 16 at 26-pins connector
	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)

2.3.18 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. DLL does not check the size of the buffers allocated by the caller.	
Parameters:	a_hDevice,	device identifier returned by AVS_Activate
	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 DLL 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)

2.3.19 AVS_GetFileSize

Function: **int AVS_GetFileSize**

```
(
AvsHandle          a_hDevice,
unsigned char*     a_pName,
unsigned int*      a_pSize
)
```

Group:	Blocking read function	
Description:	Returns the file size in bytes if the file can be read from the SD card	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate
	a_pName:	file name (14 characters including terminating zero)
	a_pSize:	pointer to buffer to store length
Return:	On success:	ERR_SUCCESS, buffer contains valid value
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER, no SD card present or file not found

2.3.20 AVS_GetFile

Function: **int AVS_GetFile**

```
(
AvsHandle          a_hDevice,
unsigned char*     a_pName,
unsigned char      a_pDest,
unsigned int       a_pSize
)
```

Group:	Blocking read function	
Description:	Returns the contents of a binary file from the SD card	
Parameters:	a_hDevice	device identifier returned by AVS_Activate
	a_pName	file name (14 characters including terminating zero)
	a_pDest	pointer to buffer to store binary file data
	a_pSize	length of buffer (expected file size, as determined with AVS_GetFileSize, max. length is 64kB)
Return:	On success:	ERR_SUCCESS, buffer contains valid value
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER, no SD card present or file not found

2.3.21 AVS_GetFirstFile

Function:	**int AVS_GetFirstFile**	
	(	
	AvsHandle	a_hDevice,
	unsigned char*	a_pName
	)	
Group:	Blocking read function	
Description:	Returns the name of the first file in the root directory of the SD card	
Parameters:	a_hDevice,	device identifier returned by AVS_Activate
	a_pName	file name (14 characters including terminating zero)
Return:	On success:	ERR_SUCCESS, buffer contains valid value
	On error:	ERR_DEVICE_NOT_FOUND
		ERR_INVALID_DEVICE_ID
		ERR_TIMEOUT (error in communication)
		ERR_INVALID_PARAMETER, no SD card present or file not found

2.3.22 AVS_GetNextFile

Function:	**int AVS_GetNextFile**	
	(	
	AvsHandle	a_hDevice,
	unsigned char*	a_pPrevName,
	unsigned char*	a_pNextName
	)	
Group:	Blocking read function	
Description:	Returns the name of the next file in root directory after a_pPrevName	
Parameters:	a_hDevice	device identifier returned by AVS_Activate
	a_pPrevName	file name (14 characters including terminating zero), this is the name returned by AVS_GetFirstFile() or by the previous call to AVS_GetNextFile()
	a_pNextName	file name (14 characters including terminating zero)
Return:	On success:	ERR_SUCCESS, buffer contains valid value
	On error:	ERR_DEVICE_NOT_FOUND
		ERR_INVALID_DEVICE_ID
		ERR_TIMEOUT (error in communication)
		ERR_INVALID_PARAMETER, no SD card present or no more files on the SD card

2.3.23 AVS_DeleteFile

Function:	**int AVS_DeleteFile**	
	(	
	AvsHandle	a_hDevice,
	unsigned char*	a_pName
	)	
Group:	Blocking read function	
Description:	Deletes a file from the SD card	
Parameters:	a_hDevice	device identifier returned by AVS_Activate
	a_pName	file name (14 characters including terminating zero)
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER, no SD card present or no more files on the SD card

2.3.24 AVS_GetFirstDirectory

Function:	**int AVS_GetFirstDirectory**	
	(	
	AvsHandle	a_hDevice,
	unsigned char*	a_pName
	)	
Group:	Blocking read function	
Description:	Returns the name of the directory in the root directory of the SD card	
Parameters:	a_hDevice	device identifier returned by AVS_Activate (-1 for first active device)
	a_pName:	directory name (14 characters including terminating zero)
Return:	On success:	ERR_SUCCESS (a_pName buffer contains valid info)
	On error:	ERR_DEVICE_NOT_FOUND (communication not open yet) ERR_INVALID_DEVICE_ID (device handle is not known in DLL) ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER, no SD card present or no directory found on the SD card

2.3.25 AVS_GetNextDirectory

Function:	**int AVS_GetNextDirectory**	
	(	
	AvsHandle	a_hDevice,
	unsigned char*	a_pPrevName,
	unsigned char*	a_pNextName
	)	
Group:	Blocking read function	
Description:	Returns the name of the next directory in the root directory after a_pPrevName	
Parameters:	a_hDevice	device identifier returned by AVS_Activate (-1 for first active device)

	a_pPrevName	directory name (14 characters including terminating zero), this is the name returned by AVS_GetFirstDirectory() or by the previous call to AVS_GetNextDirectory()
	a_pNextName	directory name (14 characters including terminating zero)
Return:	On success:	ERR_SUCCESS, a_pNextName contains valid value
	On error:	ERR_DEVICE_NOT_FOUND (communication not open yet) ERR_INVALID_DEVICE_ID (device handle is not known in DLL) ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER, no SD card present or no more directories on the SD card

2.3.26 AVS_DeleteDirectory

Function: **int AVS_DeleteDirectory**

```
(
AvsHandle        a_hDevice,
unsigned char*   a_pName
)
```

Group:	Blocking read function	
Description:	Deletes a directory from the SD card	
Parameters:	a_hDevice	device identifier returned by AVS_Activate (-1 for first active device)
	a_pName	directory name (14 characters including terminating zero)
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND (communication not open yet) ERR_INVALID_DEVICE_ID (device handle is not known in DLL) ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER, no SD card present or no more directories on the SD card

2.3.27 AVS_SetDirectory

Function: **int AVS_SetDirectory**

```
(
AvsHandle        a_hDevice,
char             a_aFileRootName[6]
)
```

Group:	Blocking data send function	
Description:	Sets current working directory. All file-functions will act on this directory.	
Parameters:	a_hDevice	device identifier returned by AVS_Activate (-1 for first active device)
	a_aFileRootName	string that sets the current working directory for all file-based functions
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND (communication not open yet) ERR_INVALID_DEVICE_ID (device handle is not known in DLL) ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER, no SD card present

2.3.28 AVS_SaveSpectraToSDCard

Function:	**int AVS_SaveSpectraToSDCard**	
	(	
	AvsHandle	a_hDevice,
	bool	a_Enable,
	unsigned char	a_SpectrumType,
	char	a_aFileRootName[6],
	TimeStampType	a_TimeStamp
	)	
Group:	Blocking data send function.	
Description:	Enables/disables writing spectra to file (if disabled the other parameters are neglected)	
Parameters:	a_hDevice	device identifier returned by AVS_Activate
	a_Enable	enable/disable storage of spectra to SD card
	a_SpectrumType	0 = Dark Spectrum
		1 = Reference Spectrum
		2 = Normal Spectrum
		The spectrumtype determines the file extension (drk, ref or roh)
	a_aFileRootName[6]	string that is used as first part of the name of the stored spectra
	a_TimeStamp	file time and date that will be used when the spectra are stored
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND
		ERR_INVALID_DEVICE_ID
		ERR_TIMEOUT (error in communication)
		ERR_INVALID_PARAMETER
		ERR_OPERATION_NOT_SUPPORTED (SD Card not present)

2.3.29 AVS_SetParameter

Function:	**int AVS_SetParameter**	
	(	
	AvsHandle	a_hDevice,
	DeviceConfigType*	a_pData
	)	
Group:	Blocking data send function.	
Description:	Overwrites the device configuration data internally and in the spectrometer. The data is not checked.	
Parameters:	a_hDevice,	device identifier returned by AVS_Activate
	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)

2.3.30 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
	a_PortId,	identifier for one of the two output signals: 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

2.3.31 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
	a_PortId:	identifier for one of the 10 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 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

2.3.32 AVS_SetPwmOut

Function:	**int AVS_SetPwmOut**	
	(	
	AvsHandle	a_hDevice,
	unsigned char	a_PortId,
	unsigned long	a_Frequency,
	unsigned char	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
	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], the frequency of outputs 0, 1 and 2 is the same (the last specified frequency is used), also the frequency of outputs 4, 5 and 6 is the same
	a_DutyCycle	percentage high time in one cycle (0 – 100), channels 0, 1 and 2 have a synchronised rising edge, the same holds for channels 4, 5 and 6
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER

2.3.33 AVS_SetSyncMode

Function:	**int AVS_SetSyncMode**	
	(	
	AvsHandle	a_hDevice,
	unsigned char	a_Enable
	)	
Group:	Internal DLL write function	
Description	Disables/enables support for synchronous measurement. DLL takes care of dividing Nmsr request into Nmsr number of single measurement requests. **Note: this function should only be called for the master spectrometer!**	
Parameters	a_hDevice	master device identifier returned by AVS_Activate
	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

2.3.34 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 integrationtime and/or high number of averages	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND
		ERR_INVALID_DEVICE_ID
		ERR_TIMEOUT (error in communication)
		ERR_INVALID_PARAMETER

2.3.35 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
	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 2nd 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 (ghostspectrum) is included in the signal. This mode cannot be used for fast external trigger and accurate timing, since the start of the scan is always delayed with the integration time (min. 3.7 ms).
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.

2.3.36 AVS_UseHighResAdc

Function:	**int AVS_UseHighResAdc**	
	(	
	AvsHandle	a_hDevice
	bool	a_Enable
	)	
Group:	Internal DLL 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

Parameters:	a_hDevice:	device identifier returned by AVS_Activate
	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

2.4 Data Elements

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

Note: To match the structures that are used in the AS5216 firmware the structures mentioned here have to be compiled with *byte alignment*.

Table 1 API data elements

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.

AvsIdentity Type	struct
	{
	char m_aSerialId[10],
	char m_aUserFriendlyId[64],
	DeviceStatus m_Status
	}
			
			serial identification number
			user friendly name to be defined by application
			device status

Type	Format	Value/Range	Description

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,
	TempSensorType m_Temperature[3],
	TecControlType m_TecControl
	ProcessControlType m_ProcessControl
	unsigned char m_aReserved[13832]
	}
		
		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)
			calibration parameters of three temperature sensors
			TecControl parameters
			ProcessControl parameters
			makes structure size equal to 63484 bytes
			(Size = 63484)

DeviceStatus	enum
	{
	UNKNOWN,
	AVAILABLE,
	IN_USE_BY_APPLICATION,
	IN_USE_BY_OTHER
	}
		
		0
		1
		2
		3
			
			initial state
			device is connected to PC and not in use
			device is connected to PC and in use by caller
			device is connected to PC and in use by other application

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 – 5.7
		
		-0.350 - +0.350
		
		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 nonlinearity correction
			higher counts limit for nonlinearity 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)

IrradianceType	struct
	{
	SpectrumCalibrationType m_IntensityCalib,
	unsigned char m_CalibrationType,
	unsigned int m_FiberDiameter,
	}
			
			Setting during intensity calibration
			Bare fiber, diffusor, integrating sphere, ….
			Fiber diameter during intensity calibration
			(Size = 16391+1+4 = 16396 bytes)

Type	Format		Value/Range	Description
MeasConfig Type	struct {			
	unsigned short	m_StartPixel,	0-4095	first pixel to be sent to PC
	unsigned short	m_StopPixel,	0 – 4095	last pixel to be sent to PC
	float	m_IntegrationTime,	0.01 – 600000	integration time in ms
	unsigned int	m_IntegrationDelay,	0 – 0xFFFFFFFF	integration delay, unit is internal FPGA clock cycle (0 = one unit before laser start)
	unsigned int	m_NrAverages,	1 – 0xFFFFFFFF	number of averages in a single measurement
	DarkCorrectionType	m_CorDynDark,		dynamic dark correction parameters
	SmoothingType	m_Smoothing,		smoothing parameters
	unsigned char	m_SaturationDetection,	0 – 2	0 = disabled, 1 = enabled, determines during each measurement if pixels are saturated (ADC value = 2^14 –1) 2 = enabled, and also corrects inverted pixels (only Sony)
	TriggerType	m_Trigger,		trigger parameters
	ControlSettingsType	m_Control,		control parameters
	}			(Size = 41 bytes)

ProcessControl Type	struct
	{
	float
	float
	float
	float
	}
		
		m_AnalogLow[2]
		m_AnalogHigh[2]
		m_DigitalLow[10]
		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)

SDCardType	Struct
	{
	bool
	unsigned char
	char
	TimeStampType
	}
		
		m_Enable,
		m_SpectrumType,
		m_aFileRootName[6],
		m_TimeStamp
				Settings for SD Card, needed in stand-alone operation
				(Size = 12 bytes)

Type	Format	Value/Range	Description
SensorType	unsigned char	0 – 9	0x00 = Reserved 0x01 = Hams8378-256 0x02 = Hams8378-1024 0x03 = ILX554 0x04 = Hams9201 0x05 = Toshiba TCD1304 0x06 = TSL1301 0x07 = TSL1401 0x08 = Hams8378-512 0x09 = Hams9840

Smoothing Type	struct
	{
	unsigned short m_SmoothPix,
	unsigned char m_SmoothModel
	}
		
		0 – 2048
		0
			
			number of neighbour 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)

Spectrum Calibration Type	struct
	{
	SmoothingType m_Smoothing,
	float m_CalInttime,
	float m_aCalibConvers[4096]
	}
		
		0.01 – 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)

Type	Format	Value/Range	Description

Standalone Type	struct
	{
	bool m_Enable,
	MeasConfigType m_Meas,
	signed short m_Nmsr,
	SDCardType m_SDCard
	}
			(Size = 56 bytes)

TecControl Type	struct
	{
	bool m_Enable,
	float m_Setpoint,
	float m_aFit[2]
	}
			Tec Control parameters for AvaSpec-256-NIR2.2
			
			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 m_Date,
	
	unsigned short 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)

TriggerType	struct
	{
	unsigned char m_Mode,
	unsigned char m_Source,
	unsigned char m_SourceType
	}
		0 – 1
		0 – 1
		0 – 1
			Trigger parameters
			
			mode, (0 = Software, 1 = Hardware)
			trigger source, (0 = external trigger, 1 = sync input)
			source type, (0 = edge trigger, 1 = level trigger)
			(Size = 3 bytes)

2.4.1 Return value constants

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

Return code	**Value**	**Description**
SUCCESS	0	Operation succeeded
ERR_INVALID_PARAMETER	-1	Function called with invalid parameter value.
ERR_OPERATION_NOT_SUPPORTED	-2	Fixed strobe not supported for ILX in fast trigger mode
ERR_DEVICE_NOT_FOUND	-3	Opening communication failed or time-out during communication occurred.
ERR_INVALID_DEVICE_ID	-4	AvsHandle is unknown in the DLL
ERR_OPERATION_PENDING	-5	Function is called while result of previous function 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 occured
ERR_NO_SPECTRA_IN_RAM	-17	No more spectra available in RAM, all read or measurement not started yet.
ERR_INVALID_DLL_VERSION	-18	DLL version information can not be retrieved
ERR_NO_MEMORY	-19	Memory allocation error in the DLL
ERR_DLL_INITIALISATION	-20	Function called before AVS_Init() is called
ERR_INVALID_STATE	-21	Function failed because AS5216 is in wrong state (e.g AVS_StartMeasurement while measurement is pending)
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

Return code	Value	Description
		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.

2.4.2 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 DLL sends this message to the application. The command value used is WM_MEAS_READY and is defined as (WM_USER + 1)
WM_DEVICECHANGE	DBT_DEVNODES_CHANGED(7)	0	After device attachment/removal Windows sends this message to the application.

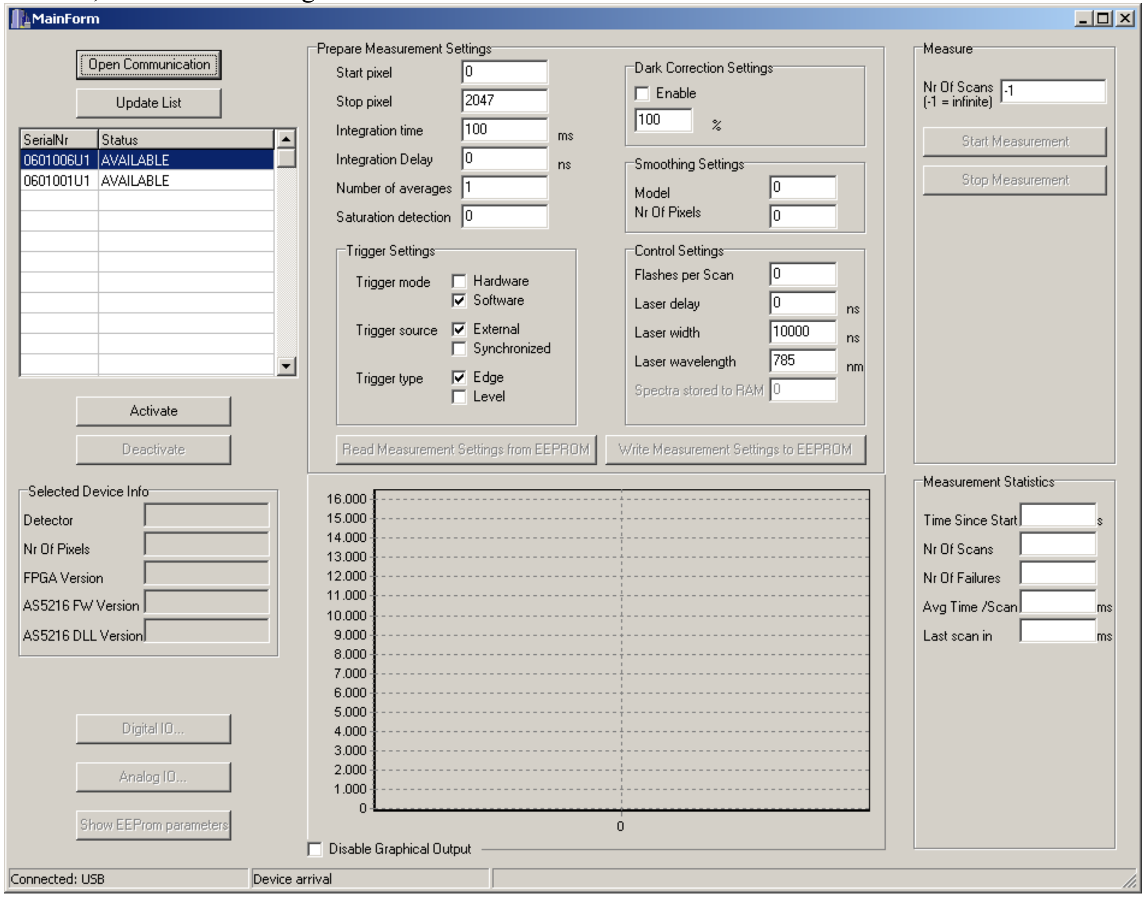
3 Example source code

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

- Borland Delphi (v. 6.0)
- Borland C++ Builder (v.5.0)
- Microsoft Visual C++ (Visual Studio v.6.0)
- Microsoft Visual C++ (2005)
- Microsoft Visual Basic (Visual Studio v.6.0)
- Microsoft Visual Basic .NET 2005 (v. 2.0)
- Microsoft C# .NET 2005 (v. 2.0)
- National Instruments LabView (v. 8.2, older versions on request)

3.1 Initialization and Activation of a spectrometer

After starting one of the Borland example programs (Borland C++ or Delphi), 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_GetNrOfDevices and AVS_GetList). The result is displayed in the list at the top left of the window, as shown in the figure below.



SerialNr	Status
0601006U1	AVAILABLE
0601001U1	AVAILABLE

Selected Device Info
Detector
Nr Of Pixels
FPGA Version
AS5216 FW Version
AS5216 DLL Version

Prepare Measurement Settings
Start pixel 0
Stop pixel 2047
Integration time 100 ms
Integration Delay 0 ns
Number of averages 1
Saturation detection 0

Trigger Settings
Trigger mode [ ] Hardware [✓ Software]
Trigger source [✓ External] [ ] Synchronized
Trigger type [✓ Edge] [ ] Level

Dark Correction Settings
[ ] Enable
100 %

Smoothing Settings
Model 0
Nr Of Pixels 0

Control Settings
Flashes per Scan 0
Laser delay 0 ns
Laser width 10000 ns
Laser wavelength 785 nm
Spectra stored to RAM 0

Measure
Nr Of Scans (-1 = infinite) -1

Measurement Statistics
Time Since Start s
Nr Of Scans
Nr Of Failures
Avg Time /Scan ms
Last scan in ms

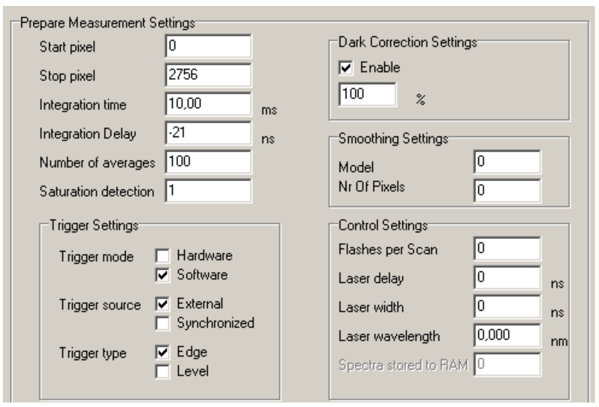
[ ] Disable Graphical Output

Connected: USB | Device arrival

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 dll and this device. After a succcessful 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 devicehandles.
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 AS5216 firmware the structures used in the as5216.dll should be compiled with *byte alignment*

3.2 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:



Prepare Measurement Settings

Start pixel	0	
Stop pixel	2756	
Integration time	10,00	ms
Integration Delay	-21	ns
Number of averages	100	
Saturation detection	1	

Trigger Settings

Trigger mode: Hardware [✓ Software]
Trigger source: [✓ External] Synchronized
Trigger type: [✓ Edge] Level

Dark Correction Settings

[✓ Enable]
100 %

Smoothing Settings

Model	0	
Nr Of Pixels	0	

Control Settings

Flashes per Scan	0	
Laser delay	0	ns
Laser width	0	ns
Laser wavelength	0,000	nm
Spectra stored to RAM	0	

```
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 2.4. In this section a more detailed description will be given.

3.2.1 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 detectortype used in the spectrometer (see also AVS_GetNumPixels). If the wavelength range of a spectrometer exceeds 1100nm (1160nm for the AvaSpec-2048x14) and the detectortype is different from “HAMS9201” (AvaSpec-NIR), the stoppixel can be set to the pixelnumber 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.

3.2.2 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 0.01 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	1.10
AvaSpec-NIR256-USB2	SENS_HAMS9201	0.52
AvaSpec-3648-USB2	SENS_TCD1304	0.01
AvaSpec-102-USB2	SENS_TSL1301	0.06
AvaSpec-128-USB2	SENS_TSL1401	0.06
AvaSpec-2048x14-USB2	SENS_HAMS9840	2.24

The longest integration time is 10 minutes (600000 ms).

3.2.3 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 3.2.9 about using the integration delay in combination with the control settings: laser delay and pulse width.

3.2.4 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 as5216 board, therefore, no time is lost by sending the individual scans from the spectrometer to the PC.

3.2.5 Measurement structure: Dynamic Dark Correction

The pixels of the CCD detector (AvaSpec-2048/3648/2048x14) are thermally sensitive, which causes a small dark current, even without light exposure. To get an approximation of this dark current, the signal of the first 14 optical black pixels of the CCD-detector can be taken and subtracted from the raw scope data. This will happen if the correct for dynamic dark option is enabled. As these 14 pixels have the same thermal behaviour as the active pixels, the correction is dynamic.
The Dark Correction Type structure includes an m_enable and m_ForgetPercentage field (see also section 2.4). 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.

3.2.6 Measurement structure: Smoothing

The smoothing type structure includes a smoothpix and a smoothmodel field. In the current version of the as5216.dll 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.

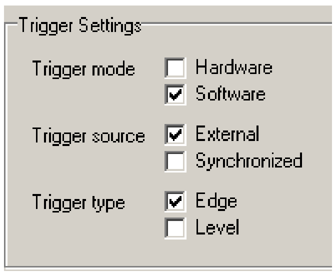
Slit or Fiber	**AvaSpec-102** **Pixel 77 µm**	**AvaSpec-128** **Pixel 63.5 µm**	**AvaSpec-256** **AvaSpec-1024** **Pixel 25 µm**	**AvaSpec-2048** **AvaSpec-2048x14** **Pixel 14 µm**	**AvaSpec-3648** **Pixel 8 µm**	**AvaSpec-NIR256** **Pixel 50 µm**
10µm	n.a.	n.a.	n.a.	0	0	n.a.
25µm	n.a.	n.a.	0	0-1	1	n.a.
50µm	0	0	0-1	1-2	2-3	0
100µm	0-1	0-1	1-2	3	5-6	0-1
200µm	1	1	3-4	6-7	12	1-2
400µm	2	2-3	7-8	13-14	24-25	3-4
500µm	3	3-4	9-10	17	31	4-5
600µm	3-4	4	11-12	21	37	5-6

3.2.7 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 as5216.dll, after a measurement result has been sent to the PC. If a measurement is the result of a number of averages, the as5216.dll 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, 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.

3.2.8 Measurement structure: Trigger Type


Trigger Settings
Trigger mode [ ] Hardware [✓ Software]
Trigger source [✓ External] [ ] Synchronized
Trigger type [✓ Edge] [ ] Level

The trigger type structure includes settings for Trigger Mode (Hardware, Software), Trigger Source (External, Synchronized) and Trigger type (Edge, Level).

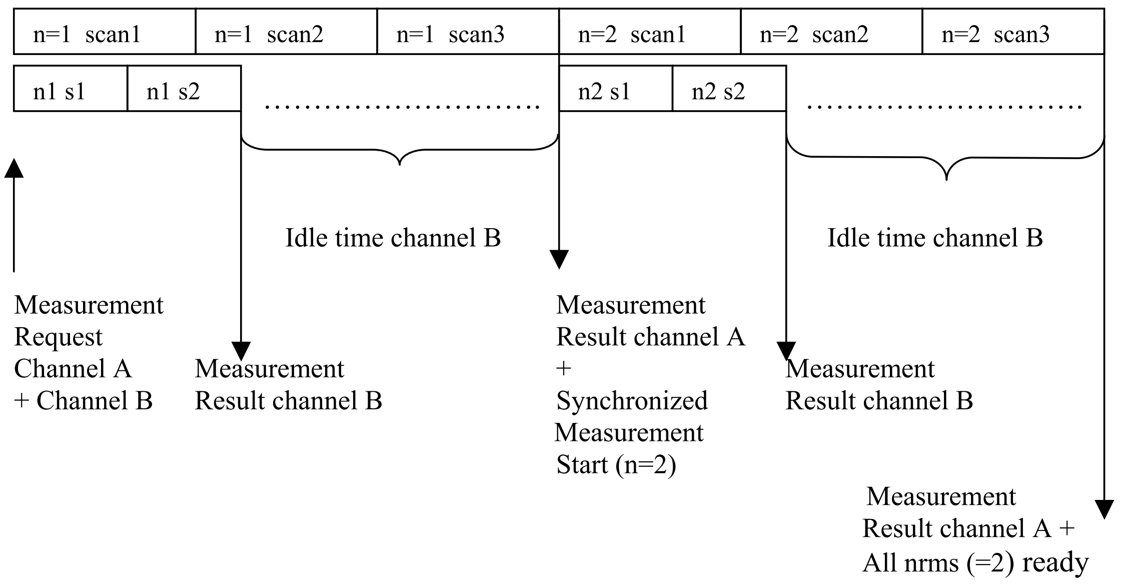
Setting the Trigger Source to Synchronized is relevant if multiple spectrometers need to run synchronised (all spectrometers start a measurement at the same time). This option will be described below under "Running multiple spectrometers Synchronized".
Single channel spectrometers, or multiple spectrometers 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.

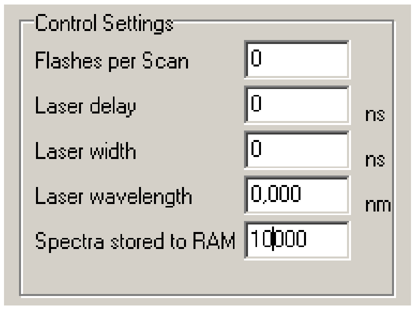
Trigger Mode = Hardware, Level triggered
This trigger setting is used when scans should be performed as long as the external trigger at pin 6 of the DB26 connector is HIGH. The spectrometer will start to accumulate data (take scans at the selected integration time) at the rising edge of the TTL pulse 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 specially useful for conveying belt applications, when a product needs to be scanned, independent of the transport speed.

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. 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". Do not call the AVS_SetSyncMode for the Slave spectrometers. 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 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. 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:


n=1 scan1
n=1 scan2
n=1 scan3
n=2 scan1
n=2 scan2
n=2 scan3
n1 s1
n1 s2
n2 s1
n2 s2
Idle time channel B
Idle time channel B
Measurement Request Channel A + Channel B
Measurement Result channel B
Measurement Result channel A + Synchronized Measurement Start (n=2)
Measurement Result channel B
Measurement Result channel A + All nrms (=2) ready

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.

3.2.9 Measurement structure: Control Settings



Control Settings

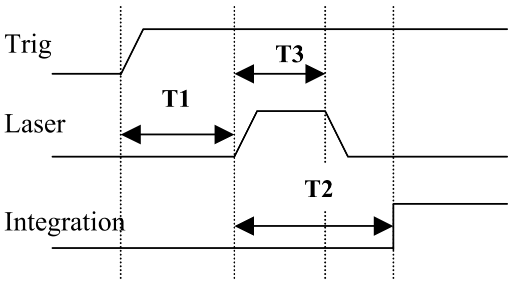
Flashes per Scan	0	
Laser delay	0	ns
Laser width	0	ns
Laser wavelength	0,000	nm
Spectra stored to RAM	10000	

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 lightsource control
A pulsed lightsource 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 as5216.dll does not check for this limitation because other light sources may operate at higher frequencies, and should also be controllable by the AvaSpec and as5216.dll.

Laser pulse control
Pin 23 at the DB26 connector can be used to send out a TTL signal which is related to the start of the integration time cyle. In the figure below, a measurement is started at the rising edge of the Trig signal. This can be a hardware or software trigger, see also section 3.2.8. The TTL signal at pin 23 (Laser) is set after the laserdelay (T1) expires. The pulsewidth for the laser pulse (T3) is set by the m_LaserWidth parameter. The integration time cycle starts after the integration delay parameter (see section 3.2.3) expires.



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.

Laser wavelength
The Laser wavelength (m_LaserWaveLength) control setting is not used in the current version of as5216. A value can be entered, but the as5216 firmware does not use this information.

StoreToRam
As of 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).
There is an amount of 4MB available for scans, corresponding with 1013 scans of 2048 pixels. Scanning less pixels will 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).
Alternatively, when using polling instead of a message driven interface, the AVS_PollScan function will also return the number of available scans. In regular measurements, it will return 1 when data are available, and 0 when they are not.
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 or AVS_PollScan.
The StoreToRam functionality has been implemented in most sample programs that come with the as5216-dll interface package.

3.3 Measurement result

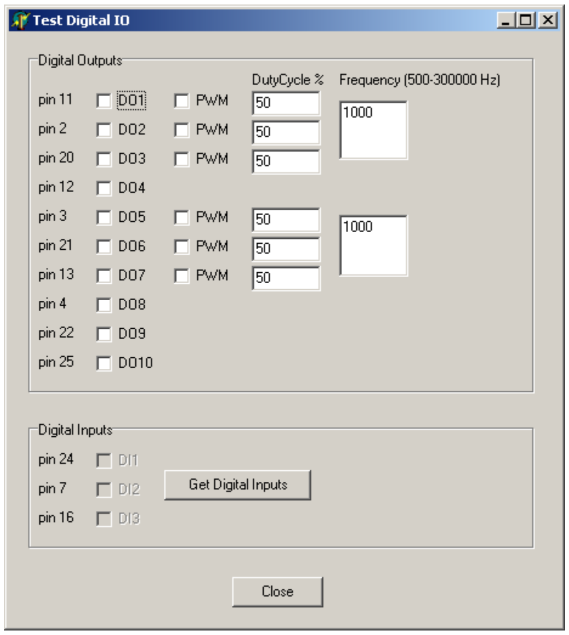
If a measurement is ready, the windows message WM_MEAS_READY is sent to the application. The Wparam value of the message should be:

- 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 section 2.4.1).

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 dll.
By calling the function AVS_GetScopeData, the spectral data is stored in the application for further processing.

3.4 Digital IO

The USB2 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 ouput 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 (dutycycle) of an AvaLight-LED lightsource, which receives input from DO1 (pin 11 of the DB26 connector).


Test Digital IO

Digital Outputs

			DutyCycle %	Frequency (500-300000 Hz)
pin 11	☐ DO1	☐ PWM	50	1000
pin 2	☐ DO2	☐ PWM	50	
pin 20	☐ DO3	☐ PWM	50	
pin 12	☐ DO4			
pin 3	☐ DO5	☐ PWM	50	1000
pin 21	☐ DO6	☐ PWM	50	
pin 13	☐ DO7	☐ PWM	50	
pin 4	☐ DO8			
pin 22	☐ DO9			
pin 25	☐ DO10			

Digital Inputs

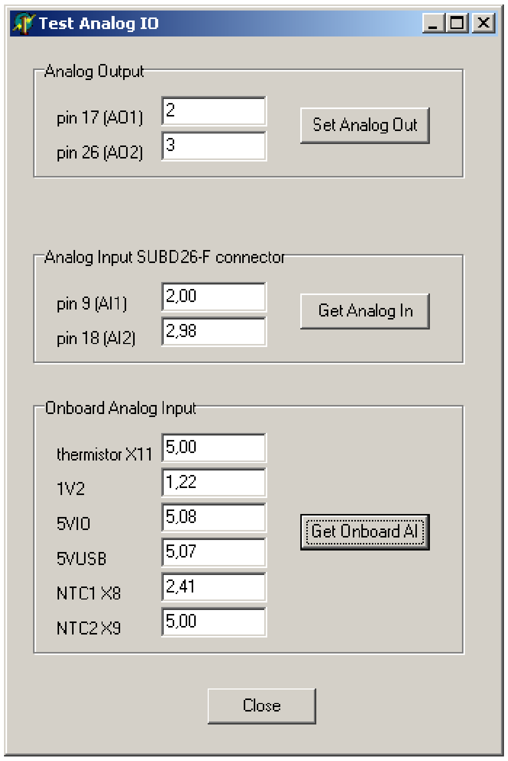
pin 24	☐ DI1
pin 7	☐ DI2
pin 16	☐ DI3

3.5 Analog IO

The USB2 platform 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:

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

The thermistor X11 is the signal received from the cooled NIR 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


Test Analog IO

Analog Output

pin 17 (AO1)	2
pin 26 (AO2)	3

Analog Input SUBD26-F connector

pin 9 (AI1)	2,00
pin 18 (AI2)	2,98

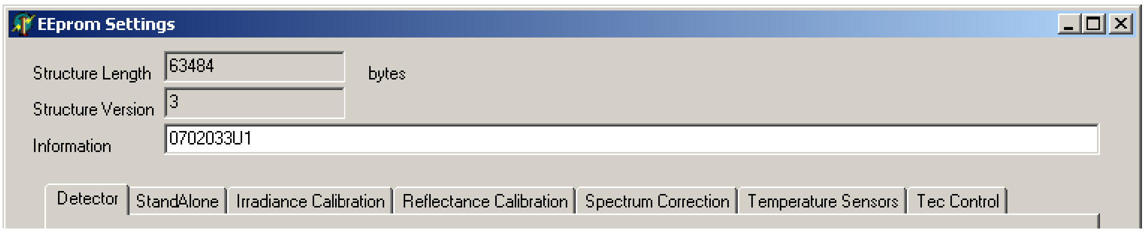
Onboard Analog Input

thermistor X11	5,00
1V2	1,22
5VIO	5,08
5VUSB	5,07
NTC1 X8	2,41
NTC2 X9	5,00

3.6 EEProm

The EEProm parameters in the DeviceConfigType structure have been briefly described in section 2.4. In this section a more detailed description will be given. The Borland C++ 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)



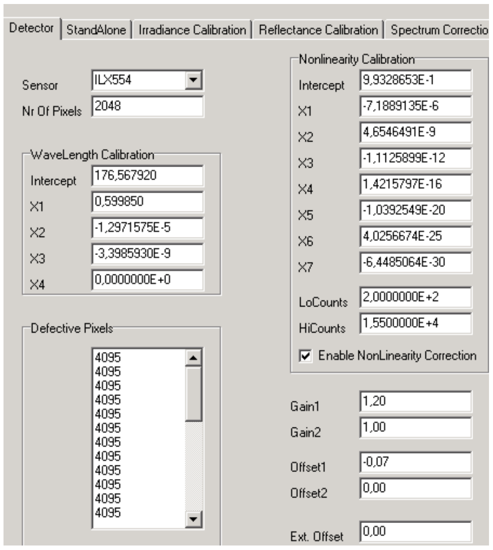
Structure Length: 63484 bytes
Structure Version: 3
Information: 0702033U1

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

3.6.1 EEProm structure: Detector Parameters

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]
```



Sensor: ILX554
Nr Of Pixels: 2048

WaveLength Calibration
Intercept: 176,567920
X1: 0,599850
X2: -1,2971575E-5
X3: -3,3985930E-9
X4: 0,0000000E+0

Defective Pixels
4095
4095
4095
4095
4095
4095
4095
4095
4095
4095
4095
4095

Nonlinearity Calibration
Intercept: 9,9328653E-1
X1: -7,1889135E-6
X2: 4,6546491E-9
X3: -1,1125899E-12
X4: 1,4215797E-16
X5: -1,0392549E-20
X6: 4,0256674E-25
X7: -6,4485064E-30
LoCounts: 2,0000000E+2
HiCounts: 1,5500000E+4
[✓] Enable NonLinearity Correction

Gain1: 1,20
Gain2: 1,00
Offset1: -0,07
Offset2: 0,00
Ext. Offset: 0,00

SensorType and Number of Pixels
The as5216 board supports many different detectors which are used in the AvaSpec spectrometers as shown in the table below:

Spectrometer	DetectorType	Number of Pixels
AvaSpec-256-USB2	SENS_HAMS8378_256	256
AvaSpec-1024-USB2	SENS_HAMS8378_1024	1024
AvaSpec-2048-USB2	SENS_ILX554	2048
AvaSpec-NIR256-USB2	SENS_HAMS9201	256
AvaSpec-3648-USB2	SENS_TCD1304	3648
AvaSpec-102-USB2	SENS_TSL1301	102
AvaSpec-128-USB2	SENS_TSL1401	128
AvaSpec-2048x14-USB2	SENS_HAMS9840	2048

For each detector, different FPGA firmware is needed. The SensorType parameter should therefore not be changed unless new FPGA firmware for another detectortype has been loaded.

The number of pixels is determined by the detectortype and should therefore not be changed, unless another detectortype has been connected and the right FPGA code has been loaded.

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:

$\lambda = \text{m\_aFit}[0] + \text{m\_aFit}[1] \cdot \text{*pixnr} + \text{m\_aFit}[2] \text{*pixnr}^2 + \text{m\_aFit}[3] \text{*pixnr}^3 + \text{m\_aFit}[4] \text{*pixnr}^4$

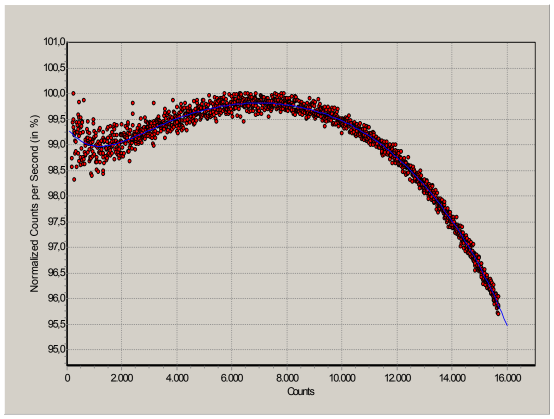
In the function AVS_GetLambda, the m_aFit coefficients are used internally to store the wavelength numbers into an array.

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).
New in version 1.1 of the as5216.dll are the m_aLowNLCounts and m_aHighNLCounts parameters. These have been added 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:

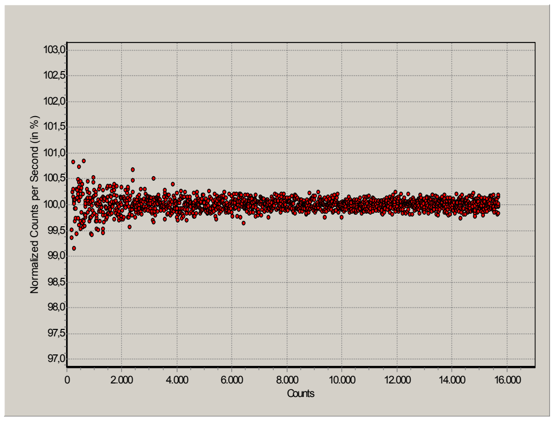
Suppose the following nonlinearity polynomial has been calculated:

```
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.

Note that the AvaSpec-2048, -2048x14 and -3648 include a "Correct for Dynamic Dark" option (see section 3.2.5). If this correction is applied, the measured dark AD Counts value (without subtracting measured dark counts) is already fluctuating around zero. The polynomial can therefore be applied directly to the measured 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 16bit ADC Counts range (see also section 2.3.36, function AVS_UseHighResAdc) does require a small modification in your application software, since the polynomial was recorded in 14bit mode, and therefore should be applied to a 14bit 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 14bit ADC mode.

In 16bit ADC mode, the measured counts will be a factor 4 higher than in 14bit mode, or with a 14 bit ADC. Therefore, the same pixel of the same spectrometer in this example returns 4*300 = 1200 Counts for darkdata 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:

$$
\begin{aligned}
NCPS = {} & \texttt{m\_aNLCorrect}\,[0] + \\
& \texttt{m\_aNLCorrect}\,[1] * (ADCFactor * 54800) + \\
& \texttt{m\_aNLCorrect}\,[2] * (ADCFactor * 54800)^2 + \\
& \texttt{m\_aNLCorrect}\,[3] * (ADCFactor * 54800)^3 + \\
& \texttt{m\_aNLCorrect}\,[4] * (ADCFactor * 54800)^4 + \\
& \texttt{m\_aNLCorrect}\,[5] * (ADCFactor * 54800)^5 + \\
& \texttt{m\_aNLCorrect}\,[6] * (ADCFactor * 54800)^6 + \\
& \texttt{m\_aNLCorrect}\,[7] * (ADCFactor * 54800)^7 = 0.97741
\end{aligned}
$$

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.

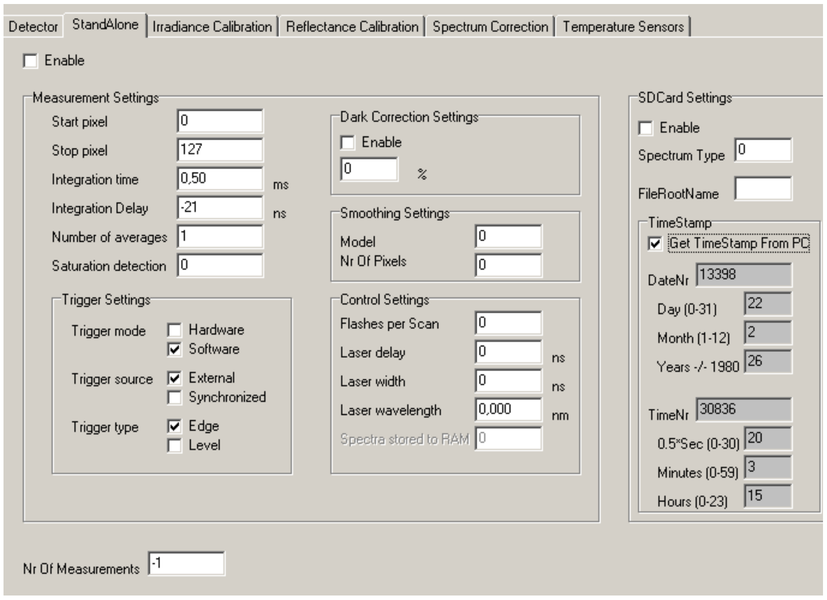
In the example above, all counts (corrected for dark) <= 800 will be corrected in AvaSoft by dividing through 0.99203. Likewise, all counts (corrected for dark) >= 62000 will be corrected in AvaSoft by dividing through 0.96099. All counts (corrected for dark): 800<counts<62000 will be corrected by the 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 spectrometers use only the m_Gain[0] and m_Offset[0]. Only the NIR2.2 with 2stage TE Cooling uses the m_Gain[1] and m_Offset[1].
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 pixelnumbers that should be eliminated from the data transfer. The as5216.dll will calculate the data for a defective pixel by interpolating the data of the neigbor 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 as5216.dll 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.

3.6.2 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). Finally, the SDCardType structure has been added, to have storage space available to store dark and reference and scopemode spectra.



☐ Enable

Measurement Settings
Start pixel: 0
Stop pixel: 127
Integration time: 0,50 ms
Integration Delay: -21 ns
Number of averages: 1
Saturation detection: 0

Dark Correction Settings
☐ Enable
0 %

Smoothing Settings
Model: 0
Nr Of Pixels: 0

Trigger Settings
Trigger mode: ☐ Hardware ☑ Software
Trigger source: ☑ External ☐ Synchronized
Trigger type: ☑ Edge ☐ Level

Control Settings
Flashes per Scan: 0
Laser delay: 0 ns
Laser width: 0 ns
Laser wavelength: 0,000 nm
Spectra stored to RAM: 0

SDCard Settings
☐ Enable
Spectrum Type: 0
FileRootName:

TimeStamp
☑ Get TimeStamp From PC
DateNr: 13398
Day (0-31): 22
Month (1-12): 2
Years -/- 1980: 26
TimeNr: 30836
0.5*Sec (0-30): 20
Minutes (0-59): 3
Hours (0-23): 15

Nr Of Measurements: -1

The Measurement parameter structure (**MeasConfigType**) has been described in detail in section 3.2, as well as the **Number of Measurements** parameter (m_Nmsr).
The **SDCardType** structure includes the following parameters:

bool	m_Enable
unsigned char	m_SpectrumType
char	m_aFileRootName[6]
TimeStampType	m_TimeStamp

These are the same parameters that have been defined in the function AVS_SaveSpectraToSDCard.
The boolean **m_Enable** is not used but has been added for possible future standalone functionality to start saving spectra to the SDCard if m_Enable becomes true;
The **m_SpectrumType** can be set to 0, 1 or 2 to indicate that a dark (*.drk), reference (*.ref) or scope (*.roh) spectrum should be saved. The **m_aFileRootName[6]** parameter is a character string that is used as first part of the name of the stored spectra. A sequence number (00 to 99 if the rootname is six characters long, 000 to 999 if the rootname is five characters long etc…) and the file extension (*.drk), reference (*.ref) or scope (*.roh) completes the filename on SDCard.
The **m_TimeStamp** parameter has been added to be able to add a date/time to the files saved on SDCard.

3.6.3 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, fiberdiameter). By reading these data from EEProm, it will be possible to convert a spectrum with raw scopedata 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:

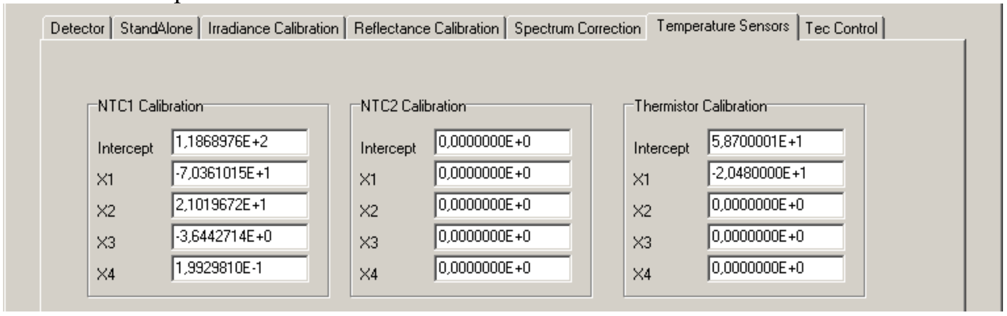
Inttimefactor = (CalInttime/CurInttime)
Irradiance Intensity = Inttimefactor* ((ScopeData(i) -DarkData(i))/IntensityCal(i))

If Scopedata(i) and Darkdata(i) are taken with the 16bit ADC Counts range (see also section 2.3.36, 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 16bit 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 scopedata into a Reflectance or Absorbance spectrum. The Spectrum Correction data can be used to correct the spectral data, e.g. for Pixel Response Non Uniformity (PRNU), or for temperature effects. The Reflectance and Correction arrays are not yet used for calibration purposes by Avantes.

3.6.4 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 NIR2.2 detector. The voltage level of the thermistors can be retrieved by calling the AVS_GetAnalogIn function (see also section 2.3.16 and 3.5).
The structure TempSensorType can hold the coefficients for a polynomial that converts the voltage level into a temperature.



NTC1 Calibration

Intercept	1,1868976E+2
X1	-7,0361015E+1
X2	2,1019672E+1
X3	-3,6442714E+0
X4	1,9929810E-1

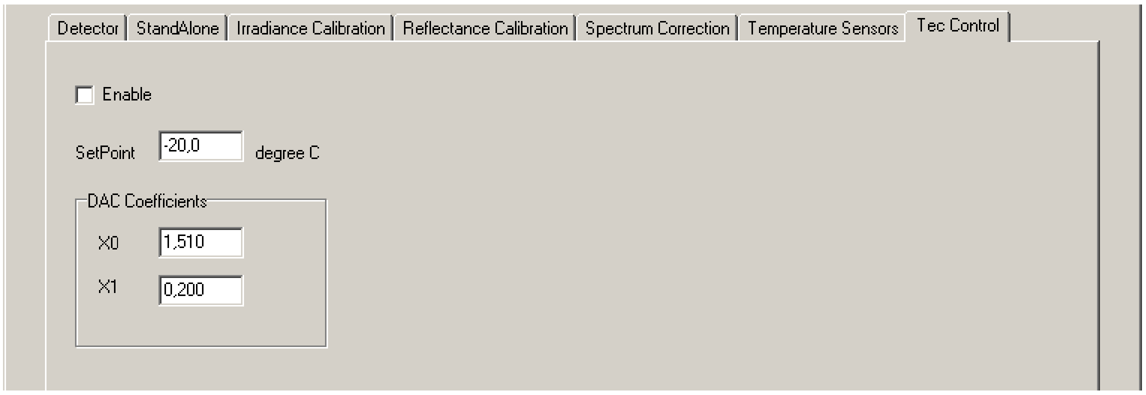
NTC2 Calibration

Intercept	0,0000000E+0
X1	0,0000000E+0
X2	0,0000000E+0
X3	0,0000000E+0
X4	0,0000000E+0

Thermistor Calibration

Intercept	5,8700001E+1
X1	-2,0480000E+1
X2	0,0000000E+0
X3	0,0000000E+0
X4	0,0000000E+0

3.6.5 EEProm structure: Tec Control



☐ Enable

SetPoint -20,0 degree C

DAC Coefficients

X0	1,510
X1	0,200

The TecControl parameters are used to control the cooling of the detector in the AvaSpec-256-NIR2.2 For this spectrometer, the m_Enable flag will always be set to true. The default setpoint in degrees Celsius is –20 °C, but it can be changed if needed. It is not recommended to change the DAC polynomial (m_aFit) which has been optimized for the AvaSpec-256-NIR2.2.

To monitor the detector temperature of the AvaSpec-256-NIR2.2, use the AVS_GetAnalogIn function, with a_AnalogInId set to 0 (see also section 2.3.16 and 3.5).

The polynomial for converting the voltage (U) to degrees Celsius for the thermistor is:

Temp [^{o}C] = 58.70 - 20.48*U

The thermistor coefficients are stored in m_Temperature[2] (see section 3.6.4).

3.6.6 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.