

AVANTES
enlightening spectroscopy

DLL Manual



2016



Microsoft, Visual C++, Visual Basic, Visual C# , Windows, Windows XP and Microsoft Office are registered trademarks of the Microsoft Corporation. Windows Vista, Windows 7, Windows 8 and Windows 10 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Delphi and C++Builder are trademarks of CodeGear, a subsidiary of Embarcadero Technologies.

LabVIEW is a trademark of the National Instruments Corporation.

MATLAB is a registered trademark of The MathWorks, Inc.

Copyright © 2016, Avantes bv

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from Avantes bv.

This manual is sold as part of an order and subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without the prior consent of Avantes bv in any form of binding or cover other than that in which it is published.

Every effort has been made to make this manual as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. Avantes bv shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this manual.



Software License

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

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

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

Table of Content

Software License	2
Table of Content	3
1. Installation	6
1.1 Installation Dialogs	6
2. Version History	9
2.1 New in version 9.3.0.0	9
2.2 New in version 9.2.2.0	9
2.3 New in version 9.2.0.0	9
2.4 New in version 9.1.1.0	9
2.5 New in version 8.3.0.0	9
2.6 New in version 8.2.0.0	10
2.7 New in version 8.1.0.0	10
2.8 New in version 2.1.0.0	10
2.9 New in version 1.9.0.0	10
2.10 New in version 1.8.0.0	10
2.11 New in version 1.7.0.0	11
2.12 New in version 1.6.0.0	11
2.13 New in version 1.5.0.0	11
2.14 New in version 1.4.0.0	12
2.15 New in version 1.3.0.0	12
2.16 New in version 1.2.0.0	13
2.17 New in version 1.1.0.0	13
2.18 New in version 1.0.0.0	13
2.18.1 Data acquisition	14
2.18.2 Synchronization in Multichannel systems	15
2.18.3 Laser control and integration time delay, e.g. for LIBS	15
2.18.4 USB2 platform specific functions	15
3. AvaSpec-DLL description	17
3.1 Interface overview	17
3.2 Usage AvaSpec-DLL	17
3.3 Exported functions	18
3.3.1 AVS_Init	18
3.3.2 AVS_Done	18
3.3.3 AVS_GetNrOfDevices	18
3.3.4 AVS_UpdateUSBDevices	18
3.3.5 AVS_UpdateETHDevices	19
3.3.6 AVS_GetList	19
3.3.7 AVS_Activate	19

3.3.8	AVS_Deactivate	20
3.3.9	AVS_Register	20
3.3.10	AVS_PrepareMeasure.....	20
3.3.11	AVS_Measure	21
3.3.12	AVS_MeasureCallback.....	21
3.3.13	AVS_GetLambda	22
3.3.14	AVS_GetNumPixels	22
3.3.15	AVS_GetParameter	22
3.3.16	AVS_PollScan	23
3.3.17	AVS_GetScopeData	23
3.3.18	AVS_GetSaturatedPixels.....	24
3.3.19	AVS_GetAnalogIn	25
3.3.20	AVS_GetDigIn.....	26
3.3.21	AVS_GetVersionInfo	27
3.3.22	AVS_SetParameter	27
3.3.23	AVS_SetAnalogOut.....	28
3.3.24	AVS_SetDigOut	29
3.3.25	AVS_SetPwmOut	30
3.3.26	AVS_SetSyncMode	30
3.3.27	AVS_StopMeasure.....	31
3.3.28	AVS_SetPrescanMode	31
3.3.29	AVS_UseHighResAdc.....	32
3.3.30	AVS_SetSensitivityMode	33
3.3.31	AVS_GetIpConfig.....	34
3.3.32	AVS_SuppressStrayLight	34
3.4	Data Elements.....	35
3.4.1	Return value constants	43
3.4.2	Windows messages	44
4.	Example source code.....	45
4.1	Initialization and Activation of a spectrometer	46
4.2	Starting a measurement.....	47
4.2.1	Measurement structure: Start- and Stoppixel	47
4.2.2	Measurement structure: Integration Time	47
4.2.3	Measurement structure: Integration Delay	48
4.2.4	Measurement structure: Number of Averages	48
4.2.5	Measurement structure: Dynamic Dark Correction	49
4.2.6	Measurement structure: Smoothing.....	49
4.2.7	Measurement structure: Saturation Detection	50
4.2.8	Measurement structure: Trigger Type	51

4.2.9	Measurement structure: Control Settings	53
4.3	Measurement result	55
4.4	Digital IO	55
4.5	Analog IO	56
4.6	EEProm	57
4.6.1	EEProm structure: Detector Parameters	58
4.6.2	EEProm structure: Standalone Parameters	62
4.6.3	EEProm structure: Irradiance, Reflectance Calibration and Spectrum Correction	63
4.6.4	EEProm structure: Stray Light Correction.....	64
4.6.5	EEProm structure: Temperature Sensors.....	64
4.6.6	EEProm structure: Tec Control	65
4.6.7	EEProm structure: ProcessControl.....	66
4.6.8	EEProm structure: Ethernet Settings	66
Appendix A:	USB driver installation.....	67
Appendix B:	Using the Ethernet spectrometer.....	69

1. Installation

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

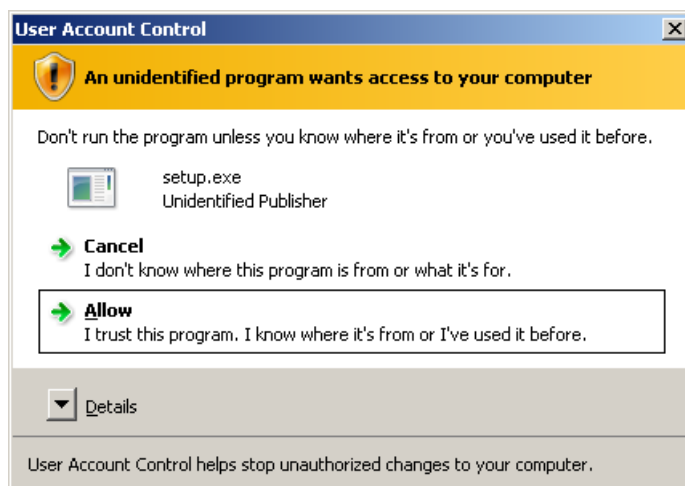
- 32 bit XP/Vista/Windows7/Windows 8/Windows 10
- 64 bit XP/Vista/Windows7/Windows 8/Windows 10

The installation program can be started by running the file “setup32.exe” from the CD-ROM.

The CD-ROM also includes an installation file for the AvaSpecX64.dll package (setup64.exe), which can be used if a 64bit programming environment is used. Note that the 32bit versions of the programming environments run perfectly well on 64bit versions of Windows. The file “32 versus 64 bit development.pdf” in the root of the CD includes more detailed information which installation file to select.

This manual describes the installation, functions and sample programs of the 32bit AvaSpec.dll.

1.1 Installation Dialogs



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

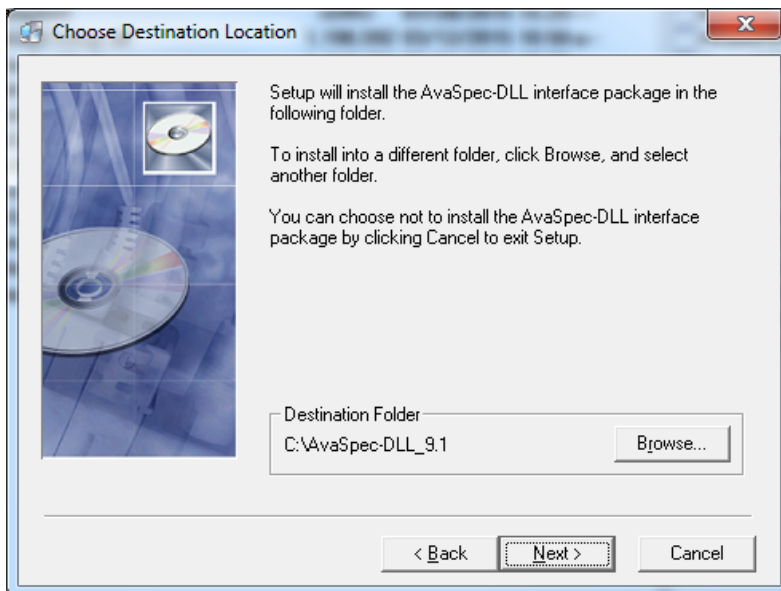
This installation is password protected. Enter the following password to proceed with the installation:



Password: Avantes6961LL4a

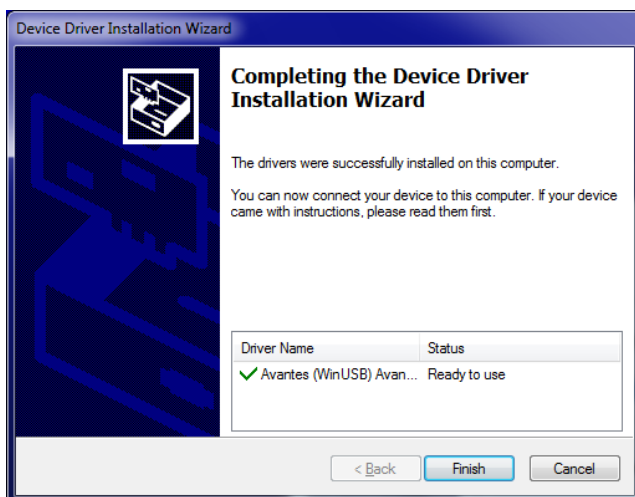
The setup program will check the system configuration of the computer. If no problems are detected, the first dialog is the “Welcome” dialog with some general information.

In the next dialog, the destination directory for the AvaSpec-DLL software can be selected. The default destination directory is C:\AvaSpec-DLL_9.3. If you want to install the software to a different directory, click the Browse button, select a new directory and click OK.



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

Next, the WinUSB device driver will be installed for the AvaSpec boards. The Device Driver Installation Wizard will be launched automatically. The last dialog in the Device Driver Installation Wizard displays whether the WinUSB driver has been installed correctly. If you experience problems here, please refer to Appendix A, which describes some special cases.

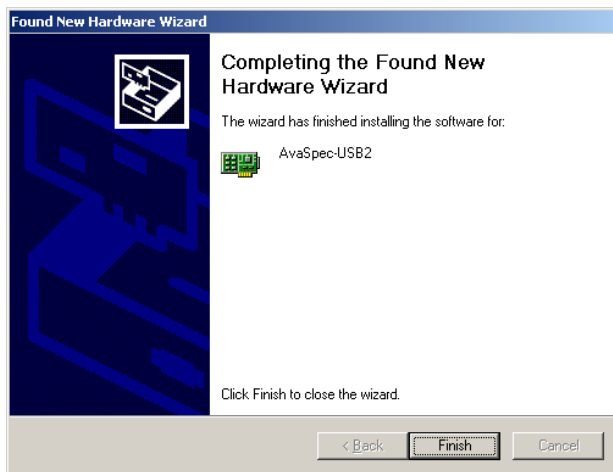


After all files have been installed, the “Installation Complete” dialog shows up. Click Finish.

Connecting the hardware

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

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:



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, under Windows XP, 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.

Launching the software

This AvaSpec-DLL manual can be opened from the Windows Start Menu. The source code of the example programs can be found in the Examples folder (default C:\AvaSpec-DLL_9.3\examples\). If you are using the Ethernet interface of the AS7010 board and experience problems opening a connection, please refer to Appendix B, which describes some common pitfalls.

2. Version History

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

2.1 New in version 9.3.0.0

- New function [AVS_SuppressStraylight](#).
- Bugfix offset level AvaSpec-2048x64TEC.
- Addition of new sample programs for [QT4](#) framework

2.2 New in version 9.2.2.0

- Removed AvaGigE references from the documentation and distribution. Please use version 8.x of the AS5216 DLL for AvaGigE support.
- A problem with the removal detection of USB spectrometers was solved.

2.3 New in version 9.2.0.0

- Needed for AS7010 devices with firmware 1.3.x.x and up. The Ethernet connection management was changed, to allow for selective ETH connections.
- Addition of the AVS_UpdateETHDevices function, that is used for the Ethernet connection management.
- The AVS_Init call was changed. With the AS7010, an a_Port value of 256 could be used for both USB and ETH ports in the previous version. This value is now only used for ETH ports with the AS7010.
- Addition of support for CMOS detectors (Hamamatsu 11638 and 11639).
- Addition of support for G9208_512 detector in AvaSpec-NIR512-2.5-HSC.

2.4 New in version 9.1.1.0

- Name change from AS5216.DLL to AvaSpec.DLL.
- Support has been added for the AS7010, which includes USB 3.0 and Ethernet support. Ethernet settings were added to the device configuration structure. The AVS_GetIpConfig function was added to support reading static IP settings on the AS7010. Operation of the spectrometer over Ethernet is almost identical to operation over USB, the main difference is in the parameter used in the AVS_Init call.
- The support functions for an SD card have been removed from the DLL, the device configuration structure remains compatible with previous versions.

2.5 New in version 8.3.0.0

- Support has been added for the AvaSpec Mini. The main differences are located in the digital I/O functions and in the analog input function that measures the board temperature. The AvaSpec Mini supports 6 bidirectional digital IO ports, which means that each pin can be used for both input and output.
- LabVIEW support was improved, by adding more options to signal the arrival of new data from the spectrometer to LabVIEW. This includes the addition to the DLL of a function that generates a custom user event, and the addition of two separate intermediate DLLs that will allow the use of Windows messaging and a custom user event, also simplify the calls when interfacing with LabVIEW. Please refer to the document 'LabVIEW support.pdf'. The LabVIEW samples are now supplied as llb files. This will prevent LabVIEW from mixing up subvis from different projects that have the same filename.
- MATLAB support was extended with a multichannel MEX file. Please refer to the document 'MATLAB support.pdf'. The files are located in the multichannel.zip archive.
- The simplified samples no longer use the 'invalid handle', as the Ethernet version of the DLL does not support using this handle. You always need to call AVS_GetList and AVS_Activate now to get a valid spectrometer handle.

- The vb2008 sample now imports DLL functions by name. For historical reasons, this was done by number in previous versions.

2.6 New in version 8.2.0.0

- A new function has been added for the NIR detectors (all AvaSpec-NIR models). The AvaSpec-NIR models can be operated in “Low Noise” or “High Sensitivity” mode. The new function AVS_SetSensitivityMode can be used to switch between these modes. See also section AVS_SetSensitivityMode.
- The sample programs for Visual Basic 6 are not included anymore. The Visual Basic 6.0 IDE is no longer supported by Microsoft as of April 8, 2008.

2.7 New in version 8.1.0.0

- This version adds support for the AvaGigE device, that allows you to control the spectrometer over an Ethernet connection. Operation over Ethernet is almost identical to operation over USB, the main difference is in the parameter used in the AVS_Init call.
- Some features were not implemented in this version. You cannot use an RS-232 or Bluetooth connection to the spectrometer. You also cannot use the SD card option of the AS-5216 board.

2.8 New in version 2.1.0.0

- Added support for the new detector in the AvaSpec-2048XL spectrometers, the Hamamatsu S11155 .
- The AvaSpec-HS1024x58 and 1024x122 (High Sensitivity) series with Hamamatsu S7031 detector were already supported in as5216.dll v 1.9, but detector specific data such as minimum integration time, smoothing and triggering characteristics were missing in the AS5216-DLL v1.9 manual.

2.9 New in version 1.9.0.0

- In June 2011, the installation program for the as5216-dll version 1.9 has been updated with winusb driver support. Reason for this was that for some spectrometer types at some modern PC's with 32bit Windows7 or Vista O/S communication errors occurred when using the Avantes AVSUSB2.sys usb driver. After updating the usb driver to winusb.sys, as recommended during the installation of the as5216-dll package, these communication problems were solved.
- The Sub vi “Byte_to_DeviceConfigType.vi”, used by the LabView sample program AVS_Main.vi in the folder LabViewSingleChan has been updated. In previous versions, the value “m_Irradiance.m_IntensityCalib.m_CallInttime” was not read properly from eeprom.
- Added dynamic dark support for the new detectors in the AvaSpec 2048x16 and the AvaSpec 2048x64 spectrometers. It is strongly recommended to keep dynamic dark correction enabled (default state). See also section Measurement structure: Dynamic Dark Correction.
- Minimum integration time for the AvaSpec-2048x16 changed from 0.91ms to 1.82 ms (see also section 4.2.2 Measurement structure: Integration Time).
- Minimum integration time for the AvaSpec-2048x64 changed from 1.75ms to 2.40 ms (see also section Measurement structure: Integration Time).
- Minimum integration time setting for the AvaSpec-NIR256 changed from 0.52ms to 0.01ms (see also section Measurement structure: Integration Time).
- Some changes to the VC# sample, to bring it in line with the VC# sample for as5216x64.dll.
- Support for the new NIR detectors (SENS_SU256LSB and SENS_SU512LDB) in the AvaSpec-NIR256-1.7-TEC, NIR512-1.7-TEC, NIR256-2.2-TEC and NIR512-2.2-TEC

2.10 New in version 1.8.0.0

- New support for MATLAB, for 32 bit MATLAB 7.1, R14 SP3.

- Added support for the new detectors in the AvaSpec 2048x16 and the AvaSpec 2048x64 spectrometers
- The VB .net, VC# and VC++ samples are now provided in VS2008 format.
- The marshalling code of the VB .net sample was extensively changed, which allows it to now directly pass structures to the DLL, without translation to a linear array of bytes.
- The large LabVIEW sample (\examples\LabView\ LabViewSingleChan\ AVS_Main.vi) was clarified by using an event structure for the different command buttons, the Measurement Configuration structure is now also translated to a linear array of bytes, before being transferred to the DLL.
- The Visual C++ version 6 sample was removed.

2.11 New in version 1.7.0.0

- Support of Dynamic Dark Correction for AvaSpec-NIR-2.0/2.2/2.5. The offset level for the cooled NIR detectors strongly depends on the ambient temperature. By using Dynamic Dark Correction, the offset level is measured with each new scan at a few blocked data pixels, and the measured signal is subtracted from all other data pixels. See also section 4.2.5.
- Floating Point Exceptions handling. The program environment in which the as5216.dll is written (C++Builder) uses by default another way of handling floating point exceptions than the Microsoft (Visual Studio) programming environment in which the application software can be developed. As long as no floating point exceptions are created by the application (e.g. because of division by zero), no problems occurred in previous as5216.dll versions. However if floating point exceptions did occur in the application, or were thrown by other third party libraries, this may have resulted in a fatal crash of the application. In as5216.dll version 1.7.0.0, this rare problem has been solved by setting the FPU Control Word to the Microsoft default. This solved the fatal crash in the few occasions that were reported in the past 5 years.
- Addition of new Delphi and C++Builder sample programs. The “old” Delphi 6.0 and C++Builder 5.0 sample programs are not fully compatible with the most recent Delphi and C++Builder versions. Since the Codegear 2009 versions, the character (char type) size is 2 bytes (Unicode), whereas in previous versions this type was only 1 byte. If 1 byte characters need to be used, the AnsiChar type should be used in the Codegear 2009 environment.
- Addition of a simple Visual Basic 6.0 sample program, which uses AVS_PollScan instead of Windows Messaging.
- Addition of a simple LabView sample program to illustrate how the StoreToRAM functionality can be implemented in combination with AVS_PollScan.
- Support of the new AvaSpec-(ULS)350F-USB2, AvaSpec-(ULS)950F-USB2, AvaSpec-(ULS)1350F-USB2 and AvaSpec-(ULS)1650F-USB2 spectrometers. The “F” in the name refers to Fast, because of the Fast minimum integration time that can be used for these spectrometers, see also section Measurement structure: Integration Time. For example, with the AvaSpec-(ULS)350F-USB2, 5000 full spectra (350 pixels) can be saved into onboard RAM in exactly one second (0.20 ms integration time).

2.12 New in version 1.6.0.0

Addition of support for Windows Vista x64. The DLL now detects whether it is running on a 64 bit version of Windows, and will then use the WinUSB device driver, instead of the 32 bit Avsusb2.sys kernel mode device driver. WinUSB is Microsoft’s own USB driver, that is distributed with Vista. The install package for the as5216.dll will configure WinUSB to support the AS5216 hardware. The DLL and examples are all still 32 bit programs, but they will now work on Vista x64 (in the so-called WoW64 mode).

2.13 New in version 1.5.0.0

- Addition of support for the AvaSpec-2048L-USB2, with Sony ILX-511 detector.
- Minimum integration time for the AvaSpec-2048x14-USB2 is changed from 2.24 msec to 2.17 msec.

- An additional delay of 500 msec is added when a device is activated, this proved to be necessary on recent PC's when using Windows Vista.

2.14 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 msec per scan for the AvaSpec-2048-USB2, and 0.1 msec 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
 - o The Visual C++ 2005 sample program has been created in the Express version.
 - o 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.
 - o 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 EEPROM structure: Detector Parameters under "Using the nonlinearity correction polynomial in combination with the 16bit ADC Counts range" and section 4.6.3 under "How to convert ScopeData (A/D Counts) to a power distribution [$\mu\text{Watt}/(\text{cm}^2.\text{nm})$]".

2.15 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 EEPROM structure: ProcessControl)
- 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 AVS_UseHighResAdc.
- 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.

2.16 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 4.6.6.

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 4.6.3 in this manual describes in more detail how this can be done.

2.17 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 3.3.27, and detailed information about how to apply the nonlinearity correction can be found in section 4.6.1

2.18 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 the following sections, 2.18.1 to 2.18.4

2.18.1 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 0.001 ms (1 μ s) resolution is used for the integration time.

8. **Timestamp.** The AVS_GetScopeData function in as5216.dll includes a timestamp in 10 μ s resolution ticks generated by the microcontroller, which can be used to measure the time between two consecutive (and processed) scans very accurately.

2.18.2 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.

2.18.3 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 μ s) at pin2 of the DB15 connector (DO2), about 1.3 μ s 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:

- **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.
- **Multiple measurements.** The number of measurements can be set by the nrms parameter, in the AVS_Measure function.
- **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).
- **Laser Delay.** The 1.3 μ s period (for the AvaSpec-2048-USB2) between receiving a trigger (measurement request in software trigger mode or TTL pulse in hardware trigger mode) can be delayed from 1.3 μ s to 89 sec (21ns steps).

2.18.4 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.** 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.
- **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 output ports can be configured for pulse

width modulation. With the AVS_SetPwmOut function, a frequency and duty cycle can be programmed for these 6 digital output ports

- **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 Exported functions). The functions AVS_GetParameter and AVS_SetParameter in as5216.dll can be used to read/write the DeviceConfigType structure from/to Eeprom.

3. AvaSpec-DLL description

3.1 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.

3.2 Usage AvaSpec-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 initialized it should select the spectrometer(s) it wants to use. Therefore, the following steps have to be taken:

1. Call AVS_UpdateUSBDevices (was AVS_GetNrOfDevices in earlier versions) 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 (USB only)

3.3 Exported functions

3.3.1 AVS_Init

Function:	int AVS_Init (short a_Port)
Group:	Blocking control function
Description:	Opens the communication with the spectrometer and initializes internal data structures. Only devices connected to the default Network Interface Controller of your host computer are initialized by AVS_Init.
Parameters:	a_Port: ID of port to be used -1 AS7010: Use both Ethernet and USB ports 0 Use USB port 1..255 not supported in this version 256 AS7010: Use Ethernet port
Return:	On success, number of connected devices On error, ERR_DEVICE_NOT_FOUND

3.3.2 AVS_Done

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

3.3.3 AVS_GetNrOfDevices

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

3.3.4 AVS_UpdateUSBDevices

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

3.3.5 AVS_UpdateETHDevices

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

3.3.6 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

3.3.7 AVS_Activate

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

3.3.8 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

3.3.9 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. This is only supported on USB connections. Note that when connecting an AS7010 to USB, the Windows Arrival message will be sent before the board receives USB power. Therefore it is recommended to implement a delay of 5 seconds after receiving the DEVICEARRIVAL message, before the application calls AVS_GetNrOfDevices and rebuilds the list (AVS_GetList)
Parameters:	a_hWnd: Application window handle
Return:	true: Registration successful false: registration failed or function not supported on OS

3.3.10 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_OPERATION_PENDING ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER ERR_INVALID_PIXEL_RANGE ERR_INVALID_CONFIGURATION (invalid fpga type) ERR_TIMEOUT ERR_INVALID_MEASPARAM_DYNDARK

3.3.11 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, the number of scans that were saved in RAM (if StoreToRAM parameter > 0), or INVALID_MEAS_DATA as WPARAM value and a_hDevice as LPARAM 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_OPERATION_PENDING ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER ERR_INVALID_STATE

3.3.12 AVS_MeasureCallback

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

3.3.13 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

3.3.14 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

3.3.15 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)

3.3.16 AVS_PollScan

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

3.3.17 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 μ S 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)

3.3.18 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

3.3.19 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 AS5216: 0 = thermistor on optical bench (NIR 2.0 / NIR2.2 / NIR 2.5 / TEC) 1 = 1V2 2 = 5VIO 3 = 5VUSB 4 = AI2 = pin 18 at 26-pins connector 5 = AI1 = pin 9 at 26-pins connector 6 = NTC1 onboard thermistor 7 = Not used Mini: 0 = NTC1 onboard thermistor 1 = Not used 2 = Not used 3 = Not used 4 = AI2 = pin 13 on micro HDMI = pin 11 on HDMI Terminal 5 = AI1 = pin 16 on micro HDMI = pin 17 on HDMI Terminal 6 = Not used 7 = Not used AS7010: 0 = thermistor on optical bench (NIR 2.0 / NIR2.2 / NIR 2.5 / TEC) 1 = Not used 2 = Not used 3 = Not used 4 = AI2 = pin 18 at 26-pins connector 5 = AI1 = pin 9 at 26-pins connector 6 = digital temperature sensor, returns degrees Celsius, not Volts 7 = Not used
Return:	a_pAnalogIn: pointer to float for analog input value [Volts or degrees Celsius] 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)

3.3.20 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:	<p>a_hDevice: device identifier returned by AVS_Activate</p> <p>a_DigInId: identifier of digital input</p> <p>AS5216:</p> <p>0 = DI1 = Pin 24 at 26-pins connector</p> <p>1 = DI2 = Pin 7 at 26-pins connector</p> <p>2 = DI3 = Pin 16 at 26-pins connector</p> <p>Mini:</p> <p>0 = DI1 = Pin 7 on Micro HDMI = Pin 5 on HDMI terminal</p> <p>1 = DI2 = Pin 5 on Micro HDMI = Pin 3 on HDMI Terminal</p> <p>2 = DI3 = Pin 3 on Micro HDMI = Pin 1 on HDMI Terminal</p> <p>3 = DI4 = Pin 1 on Micro HDMI = Pin 19 on HDMI Terminal</p> <p>4 = DI5 = Pin 4 on Micro HDMI = Pin 2 on HDMI Terminal</p> <p>5 = DI6 = Pin 2 on Micro HDMI = Pin 14 on HDMI Terminal</p> <p>AS7010:</p> <p>0 = DI1 = Pin 24 at 26-pins connector</p> <p>1 = DI2 = Pin 7 at 26-pins connector</p> <p>2 = DI3 = Pin 16 at 26-pins</p>
	a_pDigIn: pointer to digital input status (0 - 1)
Return:	<p>On success: ERR_SUCCESS, a_pDigIn contains valid value</p> <p>On error: ERR_DEVICE_NOT_FOUND</p> <p> ERR_INVALID_DEVICE_ID</p> <p> ERR_INVALID_PARAMETER (invalid digital input id.)</p> <p> ERR_TIMEOUT (error in communication)</p>

3.3.21 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)

3.3.22 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)

3.3.23 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: AS5216: 0 = AO1 = pin 17 at 26-pins connector 1 = AO2 = pin 26 at 26-pins connector Mini: 0 = AO1 = Pin 12 on Micro HDMI = Pin 10 on HDMI terminal 1 = AO2 = Pin 14 on Micro HDMI = Pin 12 on HDMI terminal AS7010: 0 = AO1 = pin 17 at 26-pins connector 1 = AO2 = pin 26 at 26-pins connector DAC value to be set in Volts (internally an 8-bits DAC is used) with range 0 - 5.0V a_Value
Return:	On success: ERR_SUCCESS On error: ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER

3.3.24 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:	<div> a_hDevice device identifier returned by AVS_Activate </div> <div> a_PortId: identifier for one of the 10 output signals: </div> <div> AS5216: </div> <div> 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 </div> <div> Mini: </div> <div> 0 = DO1 = Pin 7 on Micro HDMI = Pin 5 on HDMI terminal 1 = DO2 = Pin 5 on Micro HDMI = Pin 3 on HDMI Terminal 2 = DO3 = Pin 3 on Micro HDMI = Pin 1 on HDMI Terminal 3 = DO4 = Pin 1 on Micro HDMI = Pin 19 on HDMI Terminal 4 = DO5 = Pin 4 on Micro HDMI = Pin 2 on HDMI Terminal 5 = DO6 = Pin 2 on Micro HDMI = Pin 14 on HDMI Terminal 6 = Not used 7 = Not used 8 = Not used 9 = Not used </div> <div> AS7010: </div> <div> 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 </div>
Return:	<div> a_Value: value to be set (0-1) </div> <div> On success: ERR_SUCCESS </div> <div> On error: ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER </div>

3.3.25 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] For the AS5216, the frequency of outputs 0, 1 and 2 is the same (the last specified frequency is used) and also the frequency of outputs 4, 5 and 6 is the same.
	a_DutyCycle	For the AS7010, you can define six different frequencies. percentage high time in one cycle (0 - 100) For the AS5216, channels 0, 1 and 2 have a synchronized rising edge, the same holds for channels 4, 5 and 6. For the AS7010, rising edges are unsynchronized.
Return:	On success: ERR_SUCCESS On error: ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER	
Remark:	The PWM functionality is not supported on the Mini	

3.3.26 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.	
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	

3.3.27 AVS_StopMeasure

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

3.3.28 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 (ghost spectrum) is included in the signal. This mode cannot be used if the integration time cycle needs to start within microseconds after the spectrometer is externally triggered, but since the prescan duration is exactly known at each integration time, accurate timing (21 nanoseconds precision in external trigger mode) is very well possible in prescan mode.

- The Clear-Buffer mode.

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

3.3.29 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 software using the High Resolution mode, you need to apply the additional correction with ADCFactor (= 4.0), as explained in detail in section 4.6.1 and 4.6.3
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

3.3.30 AVS_SetSensitivityMode

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

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

3.3.31 AVS_GetIpConfig

Function:	int AVS_GetIpConfig (AvsHandle a_hDevice EthernetSettingsType* a_Data)
Group:	Blocking data send function
Description:	
Parameters:	a_hDevice: device identifier returned by AVS_Activate EthernetSettingsType: pointer to buffer that will be filled with the Ethernet settings data
Return:	On success: ERR_SUCCESS On error: ERR_DEVICE_NOT_FOUND
Remark:	Use this function to read the Ethernet settings of the spectrometer, without having to read the complete device configuration structure. Setting the values can be done with one of the full demos (like the Delphi or Qt4 demo)

3.3.32 AVS_SuppressStrayLight

Function:	int AVS_SuppressStrayLight (AvsHandle a_hDevice float a_MultiFactor, double* a_pSrcSpectrum, double* a_pDestSpectrum)
Group:	Internal data read function,
Description:	Returns the stray light corrected pixel values of a dark corrected measurement. Can be called by the application if a new spectrum is received (AVS_GetScopeData) and a dark spectrum has been subtracted. The DLL does not check the allocated buffer size! Please refer to the section under EEPROM structure for more details.
Parameters:	a_hDevice, device identifier returned by AVS_Activate a_Multifactor, multiplication factor in stray light algorithm a_pSrcSpectrum array of doubles (scope minus dark), with array size equal to maximum number of detector pixels a_pDestSpectrum array of doubles stray light suppressed, with array size equal to maximum number of detector pixels
Return:	On success: ERR_SUCCESS On error: ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_MEAS_DATA (no measurement data received) ERR_SL_CALIBRATION_NOT_AVAILABLE ERR_SL_STARTPIXEL_NOT_IN_RANGE ERR_SL_ENDPIXEL_NOT_IN_RANGE ERR_SL_STARTPIX_GT_ENDPIX ERR_SL_MFACTOR_OUT_OF_RANGE
Remark:	In the Qt4_demo_SLS sample program, this function is called to demonstrate SLS In section 4.6.4, more detailed information about using this function can be found

3.4 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 AvaSpec 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 (Size = 75 bytes)

Type	Format	Value/Range	Description
BroadcastAnswer Type	struct { unsigned char InterfaceType, unsigned char serial[AVS_SERIAL_LEN], unsigned short port, unsigned char status, unsigned int RemoteHostIp, unsigned int LocalIp, unsigned char reserved[4] }		Shows type of device that is answering Serial number of device TCP port used in communications DeviceStatus IP address of computer connected to spectrometer IP address of spectrometer reserved for future expansion (Size = 26 bytes)
ControlSettings Type	struct { unsigned short m_StrobeControl, unsigned int m_LaserDelay, unsigned int m_LaserWidth, float m_LaserWaveLength unsigned short m_StoreToRam, } 	0 - 0xFFFF 0 - 0xFFFFFFFF 0 - 0xFFFF 0 - 0xFFFF	number of strobe pulses during integration period (high time of pulse is 1 ms), (0 = no strobe pulses) laser delay since trigger, unit is internal FPGA clock cycle laser pulse width , unit is internal FPGA clock cycle (0 = no laser pulse) Peak wavelength of laser (nm), used for Raman Spectroscopy 0 = no storage to RAM > 0 = number of spectra to be stored (Size = 16 bytes)
DarkCorrection Type	struct { unsigned char m_Enable, unsigned char m_ForgetPercentage } 	0 - 1 0 - 100	disable - enable dynamic dark correction (sensor dependent) percentage of the new dark value pixels that has to be used. e.g., a percentage of 100 means only new dark values are used. A percentage of 10 means that 10 percent of the new dark values is used and 90 percent of the old values is used for drift correction (Size = 2 bytes)

Type	Format	Value/Range	Description
DeviceConfig Type	struct { unsigned short m_Len, unsigned short m_ConfigVersion, char m_aUserFriendlyId[64], DetectorType m_Detector, IrradianceType m_Irradiance, SpectrumCalibrationType m_Reflectance, SpectrumCorrectionType m_SpectrumCorrect, StandaloneType m_StandAlone, DynamicStorageType m_DynamicStorage, TempSensorType m_Temperature[3], TecControlType m_TecControl, ProcessControlType m_ProcessControl, EthernetSettingsType m_EthernetSettings, unsigned char m_aReserved[13816] }	0 - 0xFFFF	Configuration data structure: size of this structure in bytes version of this structure user friendly identification string sensor/detector related parameters intensity calibration parameters reflectance calibration parameters correction parameters stand-alone related parameters (e.g. measure mode, control) dynamic storage parameters calibration parameters of three temperature sensors TecControl parameters ProcessControl parameters EthernetSettings parameters makes structure size equal to 63484 bytes (Size = 63484)
DeviceStatus	enum { UNKNOWN, USB_AVAILABLE, USB_IN_USE_BY_APPLICATION, USB_IN_USE_BY_OTHER, ETH_AVAILABLE, ETH_IN_USE_BY_APPLICATION, ETH_IN_USE_BY_OTHER, ETH_ALREADY_IN_USE_USB }	0 1 2 3 4 5 6 7	initial state device connected by USB and not in use device connected by USB and in use by caller device connected by USB and in use by other application device connected by ETH and not in use device connected by ETH and in use by caller device connected by ETH and in use by other application device is already in use, connected by USB

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 float float unsigned short }	m_SensorType, m_NrPixels, m_aFit[5], m_NLEnable, m_aNLCorrect[8], m_aLowNLCounts, m_aHighNLCounts, m_Gain[2], m_Reserved, m_Offset[2], m_ExtOffset, m_DefectivePixels[30],	0 - 4096 -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 non-linearity correction higher counts limit for non-linearity correction gain correction for spectrometer ADC (range is divided in 64 steps) not used offset correction for spectrometer ADC in Volt (range is divided in 512 steps) offset to match the detector output range with the ADC range defective pixel numbers (Size = 188 bytes)
Dynamic StorageType	{ int32 m_Nmsr, uint8 m_Reserved[8] }	m_Nmsr, m_Reserved[8]		Number of measurements (future use) For future use and backwards compatibility (Size = 12 bytes)
Ethernet SettingsType	struct { unsigned int unsigned int unsigned int unsigned char unsigned short unsigned char } { unsigned int m_IpAddr; unsigned int m_NetMask; unsigned int m_Gateway; unsigned char m_DhcpEnabled; unsigned short m_TcpPort; unsigned char m_LinkStatus;	m_IpAddr; m_NetMask; m_Gateway; m_DhcpEnabled; m_TcpPort; m_LinkStatus;	0 - 0xFFFFFFFFF	Static IP Address (when not using a DHCP server) Net Mask value (e.g. 255.255.255.0) Default gateway value (e.g. 192.168.1.254) 0=Static IP Address used, 1=DHCP enabled Default values is 4500, used to connect to spectrometer Reserved (Size = 16 bytes)

Type	Format	Value/Range	Description
InterfaceType	enum { RS232, USB5216, USBMINI, USB7010, ETH7010 }	0 1 2 3 4	Used to tell the different AvaSpec models apart, e.g. in the Broadcast answer
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)
MeasConfig Type	struct { unsigned short m_StartPixel, unsigned short m_StopPixel, float m_IntegrationTime, unsigned int m_IntegrationDelay, unsigned int m_NrAverages, DarkCorrectionType m_CorDynDark, SmoothingType m_Smoothing, unsigned char m_SaturationDetection, TriggerType m_Trigger, ControlSettingsType m_Control, }	0-4095 0-4095 0.002 - 600000 0 - 0xFFFFFFFF 1 - 0xFFFFFFFF 0 - 2	first pixel to be sent to PC last pixel to be sent to PC integration time in ms integration delay, unit is internal FPGA clock cycle (0 = one unit before laser start) number of averages in a single measurement dynamic dark correction parameters smoothing parameters 0 = disabled, 1 = enabled, determines during each measurement if pixels are saturated (ADC value = $2^{16} - 1$) 2 = enabled, and also corrects inverted pixels (only ILX554) trigger parameters control parameters (Size = 41 bytes)

Type	Format	Value/Range	Description
ProcessControl Type	struct { float m_AnalogLow[2] float m_AnalogHigh[2] float m_DigitalLow[10] float m_DigitalHigh[10] }		Settings that can be used for the 2 analog and 10 digital output signals at the DB26 connector. The analog settings can be used to define a function output range that should correspond to the 0-5V range of the analog output signals. The digital output settings can be used as lower- and upper thresholds. (Size = 96 bytes)
SensorType	unsigned char	0 - 0x18	0x00 = Reserved 0x01 = Hams8378-256 0x02 = Hams8378-1024 0x03 = ILX554 0x04 = Hams9201 0x05 = Toshiba TCD1304 0x06 = TSL1301 0x07 = TSL1401 0x08 = Hams8378-512 0x09 = Hams9840 0x0A = ILX511 0x0B = Hams10420-2048x64 0x0C = Hams11071-2048x64 0x0D = Hams7031-1024x122 0x0E = Hams7031-1024x58 0x0F = Hams11071-2048x16 0x10 = Hams11155 0x11 = SU256LSB 0x12 = SU512LDB 0x13 = reserved 0x14 = reserved 0x15 = HAMS11638 0x16 = HAMS11639 0x17 = HAMS12443 0x18 = HAMG9208_512

Type	Format	Value/Range	Description
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] } 	0.002 - 600000	smoothing parameter during calibration integration time during calibration (ms) Conversion table from Scopedata to calibrated data (Size = 16391 bytes)
Spectrum Correction Type	struct { float m_aSpectrumCorrect[4096] } 		Correct pixel values, e.g. for PRNU (Size = 16384 bytes)
Standalone Type	struct { bool m_Enable, MeasConfigType m_Meas, signed short m_Nmsr } 		 (Size = 44 bytes)
TecControl Type	struct { bool m_Enable, float m_Setpoint, float m_aFit[2] } 		Tec Control parameters Set to True if device supports TE Cooling SetPoint for detector temperature in degr. Celsius DAC polynomial (Size = 13 bytes)
TempSensor Type	struct { float m_aFit[5] } 		Calibration coefficients temperature sensor (Size = 20 bytes)



Type	Format	Value/Range	Description
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) (Size = 4 bytes)
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) Level triggering is only supported on the AS5216 board. (Size = 3 bytes)

3.4.1 Return value constants

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

Return code	Value	Description
ERR_SUCCESS	0	Operation succeeded
ERR_INVALID_PARAMETER	-1	Function called with invalid parameter value.
ERR_OPERATION_NOT_SUPPORTED	-2	e.g. Function called to use 16bit ADC mode, with 14bit ADC hardware
ERR_DEVICE_NOT_FOUND	-3	Opening communication failed or time-out during communication occurred.
ERR_INVALID_DEVICE_ID	-4	AvsHandle is unknown in the DLL
ERR_OPERATION_PENDING	-5	Function is called while result of previous call to AVS_Measure is not received yet.
ERR_TIMEOUT	-6	No answer received from device
Reserved	-7	
ERR_INVALID_MEAS_DATA	-8	No measurement data is received at the point AVS_GetScopeData is called
ERR_INVALID_SIZE	-9	Allocated buffer size too small
ERR_INVALID_PIXEL_RANGE	-10	Measurement preparation failed because pixel range is invalid
ERR_INVALID_INT_TIME	-11	Measurement preparation failed because integration time is invalid (for selected sensor)
ERR_INVALID_COMBINATION	-12	Measurement preparation failed because of an invalid combination of parameters, e.g. integration time of (600000) and (Navg > 5000)
Reserved	-13	
ERR_NO_MEAS_BUFFER_AVAIL	-14	Measurement preparation failed because no measurement buffers available
ERR_UNKNOWN	-15	Unknown error reason received from spectrometer
ERR_COMMUNICATION	-16	Error in communication occurred
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 could 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 AvaSpec is in wrong state (e.g AVS_Measure without calling AVS_PrepareMeasurement first)
ERR_INVALID_PARAMETER_NR_PIXEL	-100	NrOfPixel in Device data incorrect
ERR_INVALID_PARAMETER_ADC_GAIN	-101	Gain Setting Out of Range
ERR_INVALID_PARAMETER_ADC_OFFSET	-102	Offset Setting Out of Range
ERR_INVALID_MEASPARAM_AVG_SAT2	-110	Use of Saturation Detection Level 2 is not compatible with the Averaging function
ERR_INVALID_MEASPARAM_AVG_RAM	-111	Use of Averaging is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_SYNC_RAM	-112	Use of the Synchronize setting is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_LEVEL_RAM	-113	Use of Level Triggering is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_SAT2_RAM	-114	Use of Saturation Detection Level 2

Return code	Value	Description
		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.
ERR_INVALID_MEASPARAM_DYNDARK	-116	Dynamic Dark Correction not supported
ERR_NOT_SUPPORTED_BY_SENSOR_TYPE	-120	Use of AVS_SetSensitivityMode not supported by detector type
ERR_NOT_SUPPORTED_BY_FW_VER	-121	Use of AVS_SetSensitivityMode not supported by firmware version
ERR_NOT_SUPPORTED_BY_FPGA_VER	-122	Use of AVS_SetSensitivityMode not supported by FPGA version
ERR_SL_CALIBRATION_NOT_AVAILABLE	-140	Spectrometer was not calibrated for stray light correction
ERR_SL_STARTPIXEL_NOT_IN_RANGE	-141	Incorrect start pixel found in EEPROM
ERR_SL_ENDPIXEL_NOT_IN_RANGE	-142	Incorrect end pixel found in EEPROM
ERR_SL_STARTPIX_GT_ENDPIX	-143	Incorrect start or end pixel found in EEPROM
ERR_SL_MFACTOR_OUT_OF_RANGE	-144	Factor should be in range 0.0 - 4.0

3.4.2 Windows messages

The following table gives an overview of window messages.

Windows message identifier	WPARAM	LPARAM	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.

4. 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 C++ Builder 5.0 (native code)
- Borland Delphi 6.0 (native code)
- Embarcadero C++Builder 2009 (native code)
- Embarcadero Delphi 2009 (native code)
- LabVIEW 8.2, older versions on request (native code)
- MATLAB 7.1 R14 SP3 (native code)
- Microsoft Visual C++ 2008 combined with the Qt4 framework (native code)
- Microsoft Visual Basic 2008 (managed code, for .net version 3.5)
- Microsoft C# 2008 (managed code, for .net version 3.5)
- Microsoft Visual C++ 2008 (managed code, for .net version 3.5)

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

The Qt4 samples include a comprehensive sample, a simple sample and a sample that demonstrates the stray light suppression function. Note the use of the AVS_MeasureCallback function in the Qt4 samples. The reason for this is the fact that the WM_USER+1 Windows message that the 32 bit AvaSpec DLL uses to signal new data is also used by Qt4. This means that the regular AVS_Measure function does not work correctly. An alternative would be to use the AVS_PollScan function.

For LabVIEW, the following sample programs are available:

- A comprehensive program for a single channel AvaSpec-USB2, which also includes subvi's for all functions in the AvaSpec.dll (LabViewSingleChan.llb in the LabViewSingleChan folder)
- A simple sample program that uses AVS_PollScan instead of Windows Messaging (polling.llb in the polling folder)
- A multichannel example program which illustrates how to run multiple spectrometer channels (fixed to 2 channels in the example program) in SYNC mode, as well as ASYNC mode (polling_mc.llb in the polling folder)
- A simple sample program that illustrates how the StoreToRam functionality can be implemented in combination with AVS_PollScan (polling_StoreToRAM.llb in the polling folder)
- Simple sample programs that demonstrate the use of the AVS_MeasureLV function, which will generate a custom user event to signal arrival of new data. Demos for one, two and four channels are available (eventdemo3.llb, mc_eventdemo.llb and mc4_eventdemo.llb in the events folder)
- Simple sample programs that demonstrate the use of an intermediate DLL that will use Windows messaging to signal the arrival of new data. Demos for one and two channels are available. (messaging2.llb and messaging_mc.llb in the messaging folder)
- Simple sample programs that demonstrate the use of an intermediate DLL that will use a custom user event to signal arrival of new data. Demos for one and two channels are available. (intermediate.llb and intermediate_mc.llb in the intermediate folder)

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

The sample program in MATLAB is described in a separate manual (MATLAB support.pdf) that can be found in the main folder where the AvaSpec.dll package has been installed. A version of the MEX file for multiple spectrometers can be found in the multichannel.zip file in the MATLAB subdirectory.

4.1 Initialization and Activation of a spectrometer

After starting one of the full feature example programs (C++Builder, Delphi or Qt4_demo_full folder), the main window will be displayed. By clicking the “Open Communication” button, the AVS_Init function is called and if successful, the serial number and status for the connected spectrometer(s) is collected (AVS_UpdateUSBDevices c.q. AVS_GetNrOfDevices and AVS_GetList). The result is displayed in the list at the top left of the window, as shown in the figure below.

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

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

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

4.2 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:

The parameters in the measurement structure have been briefly described in section 3.4. In this section a more detailed description will be given.

4.2.1 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-2048XL/x14/x16/x64) and the detectortype is different from “HAMS9201, HAMS9208_512, SU256LSB or SU512LDB” (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.

4.2.2 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 0.001 ms steps. The minimum integration time is detector dependent. The table below shows the values for the different detector types:

Spectrometer	Detector Type	Min. Integration time [ms]
AvaSpec-256-USB2	SENS_HAMS8378_256	0.56
AvaSpec-1024-USB2	SENS_HAMS8378_1024	2.20
AvaSpec-2048-USB2	SENS_ILX554	1.05
AvaSpec-2048L-USB2	SENS_ILX511	1.05

AvaSpec-NIR256-1.7, AvaSpec-NIR256-2.0TEC, AvaSpec-NIR256-2.5TEC, AvaSpec-NIR512-2.5-HSC	SENS_HAMS9201 SENS_HAMG9208_512	0.01* 0.01
AvaSpec-NIR256-1.7TEC, AvaSpec-NIR256-2.2TEC**	SENS_SU256LSB	0.02
AvaSpec-NIR512-1.7TEC AvaSpec-NIR512-2.2TEC	SENS_SU512LDB	0.02
AvaSpec-3648-USB2	SENS_TCD1304	0.01
AvaSpec-102-USB2	SENS_TSL1301	0.06
AvaSpec-128-USB2	SENS_TSL1401	0.07
AvaSpec-2048x14-USB2	SENS_HAMS9840	2.17
AvaSpec-350F-USB2	SENS_ILX554	0.20
AvaSpec-950F-USB2	SENS_ILX554	0.50
AvaSpec-1350F-USB2	SENS_ILX554	0.70
AvaSpec-1650F-USB2	SENS_ILX554	0.85
AvaSpec-2048x16-USB2	SENS_HAMS11071_2048X16	1.82***
AvaSpec-2048x64-USB2	SENS_HAMS11071_2048X64	2.40****
AvaSpec-2048x64TEC-USB2	SENS_HAMS10420_2048X64	9.70
AvaSpec-HS1024x58-USB2	SENS_HAMS7031_1024X58	5.22
AvaSpec-HS1024x122-USB2	SENS_HAMS7031_1024X122	6.24
AvaSpec-2048XL-USB2	SENS_HAMS11155	0.002
AvaSpec-2048CL	SENS_HAMS11639	0.03

* = 0.01ms for SENS_HAMS9201 in Firmware v. 000.025.000.000 and later, else 0.52ms

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

*** = 1.82 ms for SENS_HAMS11071_2048X16 in FPGA 006.003.000.000 or later, else 0.91ms

**** = 2.40 ms for SENS_HAMS11071_2048X64 in FPGA 006.003.000.000 or later, else 1.75ms

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

4.2.3 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 Measurement structure: Control Settings about using the integration delay in combination with the control settings: laser delay and pulse width.

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

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

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

4.2.5 Measurement structure: Dynamic Dark Correction

The pixels of the CCD detector are thermally sensitive, which causes a small dark current, even without exposure to light. To get an approximation of this dark current, the signal of some optical black pixels of the detector can be taken and subtracted from the raw scope data. This will happen if the “Correct for Dynamic Dark” option is enabled. Some detector types (AvaSpec-2048/2048L/3648) include dedicated optical black pixels. At these optical black pixels, the intensity and thermal behaviour is the same as the active data pixels, if no light falls on the detector. Enabling dynamic dark correction will therefore result in a baseline fluctuating round zero, and measurement data will be less sensitive for temperature changes than with dynamic dark correction off.

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

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

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

The Dark Correction Type structure includes an `m_enable` and `m_ForgetPercentage` field (see also section Exported functions). 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.

4.2.6 Measurement structure: Smoothing

The smoothing type structure includes a `smoothpix` and a `smoothmodel` field. In the current version of the AvaSpec.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 $15 \times 14 = 210$ 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 $((\text{slit size} / \text{pixel size}) - 1) / 2$

In the table below, the recommended smoothing values for the AvaSpec spectrometer are listed as function of the light beam that enters the spectrometer. This light beam is the fiber core diameter, or if a smaller slit has been installed in the spectrometer, the slit width. Note that this table shows

the optimal smoothing without losing resolution. If resolution is not an important issue, a higher smoothing parameter can be set to decrease noise at the price of less resolution.

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

4.2.7 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 AvaSpec dll, after a measurement result has been sent to the PC. If a measurement is the result of a number of averages, the AvaSpec 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, 2048L, 2048x14 and 3648 and AvaSpec-NIR do not show this effect, so no correction is needed. Normally, you don't need to use this third level for the AvaSpec-2048, but when measuring a peaky spectrum with some heavily saturated peaks, the autocorrect can be used. A limitation to this level is that it can be used only if no averaging is used (m_NrAverages=1). The AvaSpec-USB2 spectrometers with an AS5216 board Rev C and earlier were equipped with a 14-bit AD converter (range 0..16383). In this case the detector is saturated at 16383 counts.

4.2.8 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.

The delay between the rising edge of the TTL pulse and the start of the integration time cycle depends on the spectrometer type, as shown in the table below.

Spectrometer Type	Minimum Delay [μ s]	Maximum Delay [μ s]
AvaSpec-128-USB2	9	60
AvaSpec-256-USB2	0.80	0.84
AvaSpec-1024-USB2	0.80	0.84
AvaSpec-2048-USB2*	1.28	1.30
AvaSpec-2048L-USB2	3.28	3.30
AvaSpec-3648-USB2 (clearbuffer mode)**	0.28	0.30
AvaSpec-NIR256-1.7, AvaSpec-NIR256-2.0TEC, AvaSpec-NIR256-2.5TEC	0	600
AvaSpec-NIR512-2.5-HSC	531	533
AvaSpec-NIR256-1.7TEC, AvaSpec-NIR256-2.2TEC	4.92***	5.75***
AvaSpec-NIR512-1.7TEC, AvaSpec-NIR512-2.2TEC	4.92****	5.75****
AvaSpec-2048x14-USB2	-2170	0
AvaSpec-2048x16-USB2	-1820	0
AvaSpec-2048x64-USB2	-2400	0
AvaSpec-2048x64TEC-USB2	-9700	0
AvaSpec-HS1024x58-USB2	-5220	0
AvaSpec-HS1024x122-USB2	-6240	0
AvaSpec-2048XL-USB2	0.28	0.30
AvaSpec-2048CL-USB2	0.90	0.92

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

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

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

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

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

So if the application allows that the AvaSpec-3648-USB2 in prescan mode is triggered a couple of milliseconds before the event that needs to be measured, this event can be shifted with high precision into the integration time cycle of the spectrometer. Moreover, the integration delay parameter (section 4.2.3) can be used to add additional delay in steps of 21 nanoseconds to the min_delay calculated above.

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

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

Trigger Mode = Hardware, Level triggered

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

Running multiple spectrometers Synchronized

All USB2 platform spectrometers can be connected by a SYNC cable. In syncmode, one spectrometer is configured as "Master" by calling the AVS_SetSyncMode function for this channel with the a_Enable flag set to 1. The trigger source for the Master channel should **not** be set to Synchronized, but to External. The trigger mode for the Master can be set to Software (if a measurement should start after a measurement request in the software), or to Hardware (if a measurement should start after an external hardware trigger pulse at pin 6 of the Master DB26 connector. All other ("slave") spectrometers are set into "Synchronized" mode by setting the Trigger Source to "Synchronized" and the Trigger Mode to "Hardware".

A synchronized measurement is started by calling AVS_Measure first for all slave channels. As a result, these channels start listening to their SYNC input port. Secondly a measurement request (call to AVS_Measure) needs to be posted for the Master channel. If the trigger mode for the Master is "software", this result in nrms measurements for all channels. If the trigger mode for the Master is "hardware", the nrms measurements for all channels are started after an external trigger has been received at at pin 6 of the Master DB26 connector. The nrms parameter in the AVS_Measure function should be set to the same value for all activated channels.

Source code for the sample programs that support synchronization of multichannel systems can be found in the following folders:

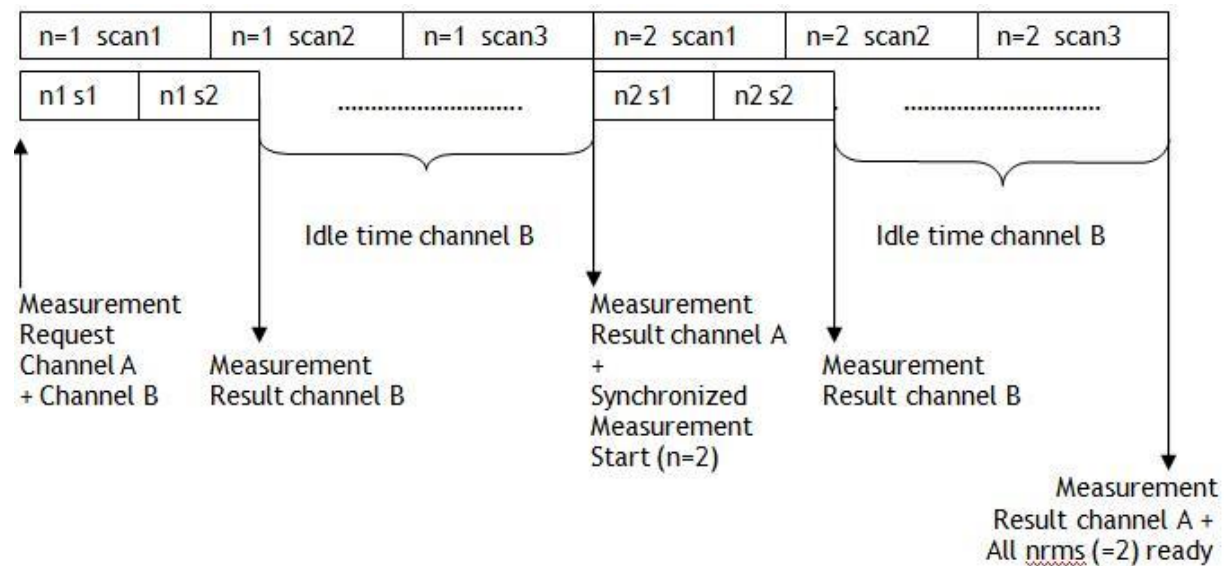
```
..\examples\ Borland Delphi 6\ multichannel\
..\examples\ Codegear Delphi 2009\ multichannel\
..\examples\LabView\polling_mc\
```

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

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

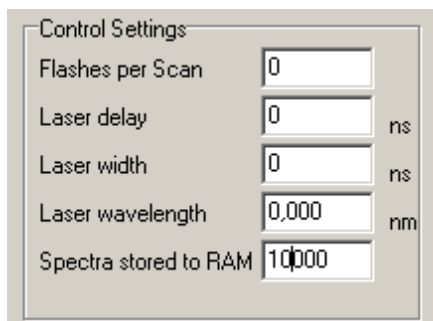
Channel B: nrms=2, integration time 65ms, average 2,

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



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

4.2.9 Measurement structure: Control Settings



The Control Settings include parameters to control

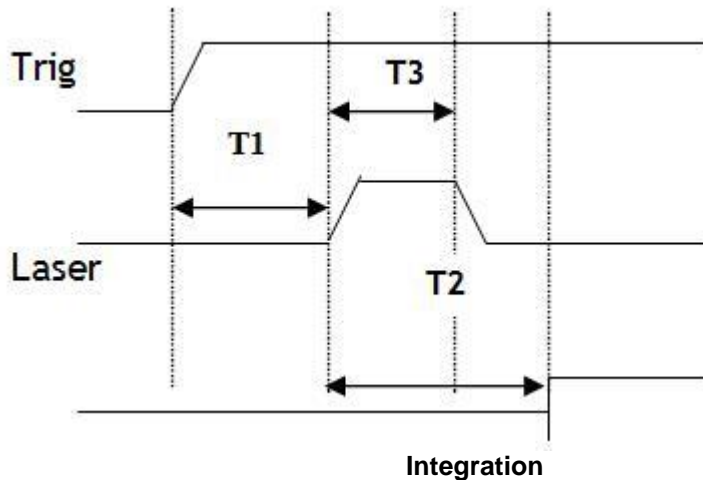
- A pulsed lightsource (m_StrobeControl)
- A laser pulse (m_LaserDelay and m_LaserWidth)
- The Number of Spectra that will be stored to onboard RAM (m_StoreToRam)

Pulsed lightsource control

A pulsed light source like the AvaLight-XE needs to be synchronized with the integration time cycle. The `m_StrobeControl` parameter determines the number of pulses the spectrometer sends out at pin 5 at the DB26 connector during one integration time cycle. The maximum frequency at which the AvaLight-XE operates is 100 Hz. This means that the minimum integration time for 1 pulse per scan is 10 ms. When setting the number of pulses e.g. to 3, the minimum integration time should be 30 ms. The AvaSpec 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 AvaSpec dll.

Laser pulse control

For the fast trigger detectors ILX554, ILX511, S11155 and S11639 in the AvaSpec-2048, 2048L, 2048XL and 2048CL, pin 23 at the DB26 connector can be used to send out a TTL signal which is related to the start of the integration time cycle. 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 Measurement structure: Trigger Type. 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 Measurement structure: Integration Delay) 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 1 μ s (ILX554 detector in AvaSpec-2048-USB2, see also section Measurement structure: Integration Delay).

Laser wavelength

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

StoreToRam

As of AS5216 firmware version 0.20.0.0 the StoreToRam function has been implemented. To use this function, you must set the requested number of scans in the `m_StoreToRam` control setting, and start measuring with a call to `AVS_Measure` using 1 as the number of measurements (`a_Nmsr`).

For the AS5216, there is an amount of 4MB available for scans, corresponding with 1013 scans of 2048 pixels. The AvaSpec Mini and the AS7010 have much more memory, allowing for 7783 and 16383 scans of 2048 pixels respectively. Scanning less pixels will in each case yield a larger capacity in scans. The AVS_Measure message signaling the arrival of data will have a WParam value equal to the number of scans stored in RAM. In regular measurements, this value only signals success (with value ERR_SUCCESS) or failure (with a negative error message).

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

The scans can subsequently be read with a corresponding number of calls to AVS_GetScopeData. If you request more scans than will fit in memory, scanning will continue until the memory is fully used, therefore you should always request the number of scans that is returned in WParam (when using Windows Messaging).

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

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

4.3 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 3.4.1 Return value constants)

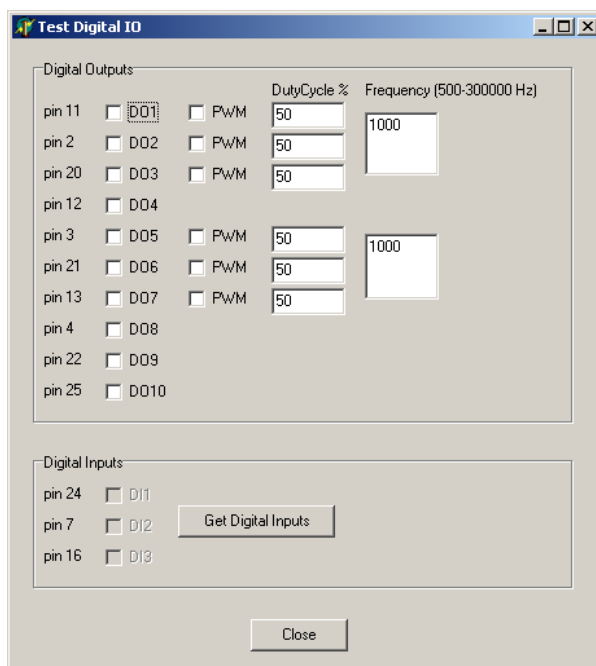
The LParam value of the message contains the devicehandle for the spectrometer for which the data is ready.

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

4.4 Digital IO

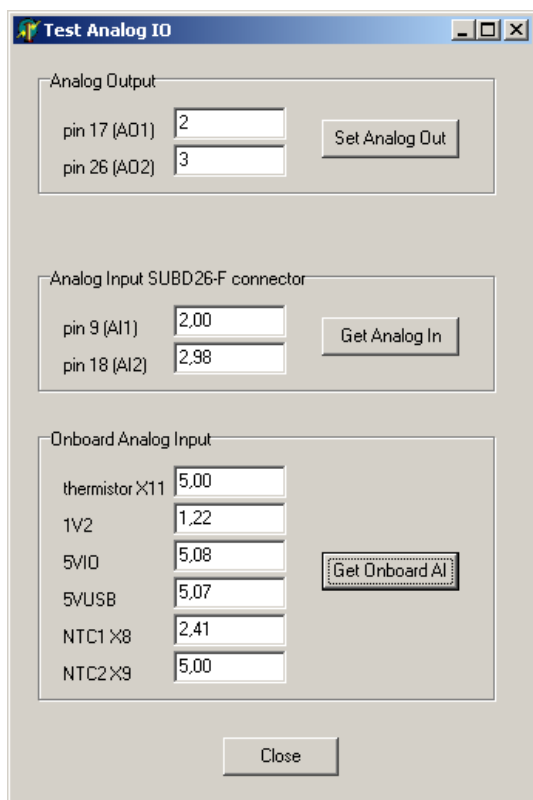
The AS5216 and AS7010 platform spectrometers have 10 programmable digital output pins and 3 programmable input pins available at the DB26 connector. The function AVS_SetDigOut and AVS_GetDigIn can be used to control these ports. Moreover, 6 out of the 10 programmable output ports can be configured for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

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



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

4.5 Analog IO



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

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

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

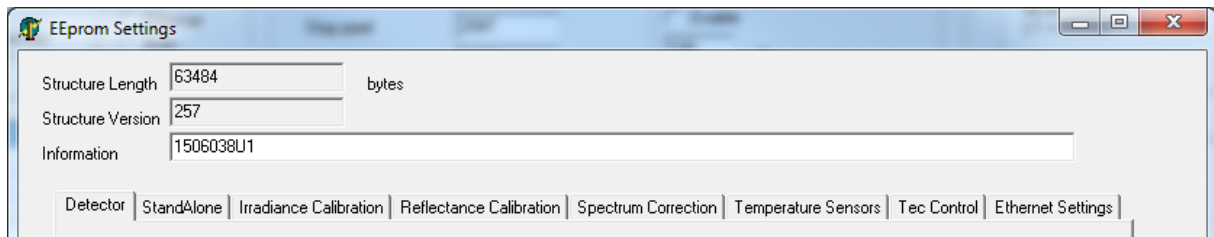
The AvaSpec Mini also has two programmable analog output pins and two analog input pins, , available at the micro HDMI connector. A HDMI terminal adapter is available, which has a different pin numbering scheme. The NTC1 thermistor is read with identifier value 0, in the position of the Thermistor X1.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

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

4.6 EEPROM

The EEPROM parameters in the DeviceConfigType structure have been briefly described in section 3.4. In this section a more detailed description will be given. The C++Builder and Delphi sample programs display most of the parameters in the structure. The Structure Length (m_Len), Structure Version (m_ConfigVersion) and InfoString (m_aUserFriendlyId[64]) are shown on top of the tabs that correspond to the structures that are used to group the parameters into the following categories:

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

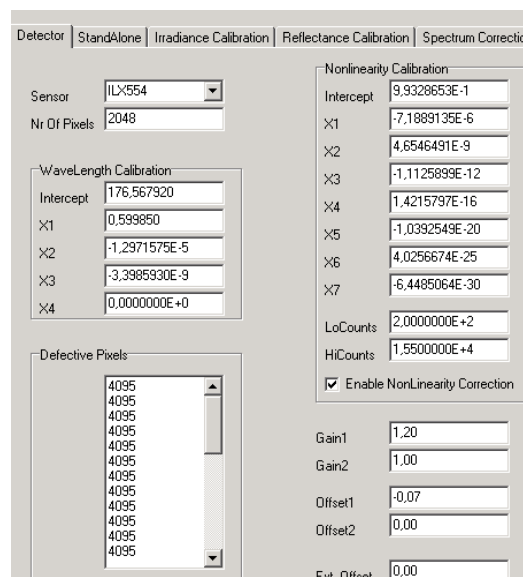


The structure version is used internally to maintain compatible between different versions of the dll and firmware. The Information character string can be used e.g. to write a user friendly name for the spectrometer.

4.6.1 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]



SensorType and Number of Pixels

The boards support many different detectors which are used in the AvaSpec spectrometers as shown in the tables below:

AS5216:

Spectrometer	DetectorType	Number of Pixels
AvaSpec-102-USB2	SENS_TSL1301	102
AvaSpec-128-USB2	SENS_TSL1401	128
AvaSpec-256-USB2	SENS_HAMS8378_256	256
AvaSpec-1024-USB2	SENS_HAMS8378_1024	1024
AvaSpec-2048x14-USB2	SENS_HAMS9840	2048
AvaSpec-2048x16-USB2	SENS_HAMS11071_2048X16	2048
AvaSpec-2048x64-USB2	SENS_HAMS11071_2048X64	2048
AvaSpec-2048x64TEC-USB2	SENS_HAMS10420_2048X64	2048
AvaSpec-NIR256-1.7, AvaSpec-NIR256-2.0TEC, AvaSpec-NIR256-2.5TEC	SENS_HAMS9201	256
AvaSpec-NIR512-2.5-HSC	SENS_HAMG9208_512	512
AvaSpec-NIR256-1.7TEC, AvaSpec-NIR256-2.2TEC*	SENS_SU256LSB	256
AvaSpec-NIR512-1.7TEC, AvaSpec-NIR512-2.2TEC	SENS_SU512LDB	512
AvaSpec-2048-USB2	SENS_ILX554	2048
AvaSpec-350F-USB2	SENS_ILX554	350
AvaSpec-950F-USB2	SENS_ILX554	950
AvaSpec-1350F-USB2	SENS_ILX554	1350
AvaSpec-1650F-USB2	SENS_ILX554	1650
AvaSpec-2048L-USB2	SENS_ILX511	2048
AvaSpec-3648-USB2	SENS_TCD1304	3648
AvaSpec-HS1024x58-USB2	SENS_HAMS7031_1024X58	1024
AvaSpec-HS1024x122-USB2	SENS_HAMS7031_1024X122	1024
AvaSpec-2048XL-USB2	SENS_HAMS11155	2068

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

Mini:

Spectrometer	DetectorType	Number of Pixels
AvaSpec-Mini-2048	SENS_ILX554	2048
AvaSpec-Mini-2048L	SENS_ILX511	2048
AvaSpec-Mini-3648	SENS_TCD1304	3648

AS7010:

Spectrometer	DetectorType	Number of Pixels
AvaSpec-ULS2048L-EVO	SENS_ILX511	2048
AvaSpec-ULS2048CL-EVO	SENS_HAMS11639	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.

Also for the Fast Series (350F, 950F, 1350F, 1650F), the number of pixels is fixed and should not be changed.

Wavelength Calibration

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

$$\lambda = m_aFit[0] + m_aFit[1] * pixnr + m_aFit[2] * pixnr^2 + m_aFit[3] * pixnr^3 + m_aFit[4] * pixnr^4$$

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

Nonlinearity Calibration and Correction

A polynomial can be used to correct for nonlinear behavior of the detector. The polynomial coefficients can be stored in the EEPROM and used by the application software to correct the raw AD Counts.

The nonlinearity calibration service (determination of the polynomial coefficients) is included in the IRRAD-CAL irradiance calibration service, but can also be ordered separately (NL-Calibration).

The m_aLowNLCounts and m_aHighNLCounts parameters have been added since AvSpec dll version 1.1, to be able to limit the range (in counts) for which the correction polynomial should be applied.

The correction that needs to be implemented in the application software can be illustrated by using an example:

Suppose the following nonlinearity polynomial has been calculated:

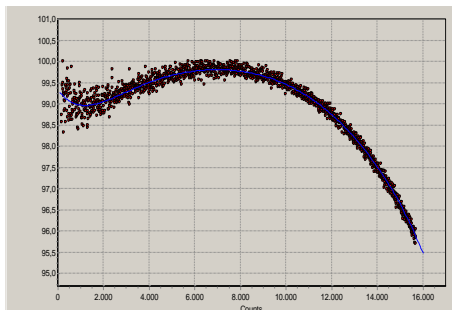
```

m_aNLCorrect[0] = 9.93286529334744E-001
m_aNLCorrect[1] = -7.18891352982627E-006
m_aNLCorrect[2] = 4.65464905353804E-009
m_aNLCorrect[3] = -1.11258994803382E-012
m_aNLCorrect[4] = 1.42157972847117E-016
m_aNLCorrect[5] = -1.03925487491128E-020
m_aNLCorrect[6] = 4.02566735990250E-025
m_aNLCorrect[7] = -6.44850644473040E-030
m_aLowNLCounts = 200.0
m_aHighNLCounts = 15500.0

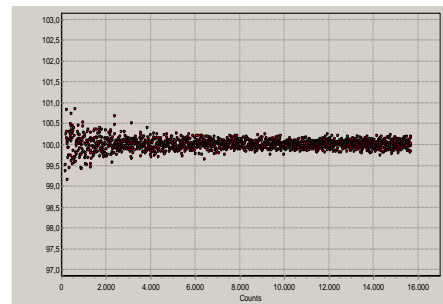
```

The polynomial is calculated by measuring the AD Counts for a number of pixels (10) over different integration times to get the pixel data over a wide range from (in this example) 200 to 15500 counts. The measured AD Counts are corrected for the offset value by subtracting the dark

spectrum. For each of the 10 pixels in the measurement the counts per second is calculated and normalized to its maximum value, which is set to 100%. In the left figure below, the normalized counts per second are displayed against the measured AD Counts (corrected for dark). The polynomial is the best fit through these measured points. The right figure below has been created by applying the polynomial to the measured points, and recalculating the normalized counts per second. It is important to realize that the polynomial should be applied to the AD Counts that have been corrected for the dark counts.



Before linearization



After linearization

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

$$\begin{aligned} \text{NCPS} = & \text{m_aNLCorrect}[0] + \\ & \text{m_aNLCorrect}[1] * 13700 + \\ & \text{m_aNLCorrect}[2] * 13700^2 + \\ & \text{m_aNLCorrect}[3] * 13700^3 + \\ & \text{m_aNLCorrect}[4] * 13700^4 + \\ & \text{m_aNLCorrect}[5] * 13700^5 + \\ & \text{m_aNLCorrect}[6] * 13700^6 + \\ & \text{m_aNLCorrect}[7] * 13700^7 = 0.97741 \end{aligned}$$

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

The `m_aLowNLCounts` and `m_aHighNLCounts` parameters can be used to limit the range for the correction (in counts) for which the polynomial should be applied. The use of polynomials beyond the range of measured data points can give erratic corrections. In AvaSoft, Avantes uses the same correction factor (NCPS) for measured counts (corrected for dark) that are lower than `m_aLowNLCounts` as is used for `m_aLowNLCounts`, and for counts higher than `m_aHighNLCounts` the same NCPS as is used for `m_aHighNLCounts`. In the example above, `NCPS[200] = 0.99203` and all counts ≤ 200 will be corrected in AvaSoft by dividing through 0.99203. Likewise `NCPS[15500] = 0.96099` and all counts ≥ 15500 will be corrected in AvaSoft by dividing through 0.96099. All counts: $200 < \text{counts} < 15500$ 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 `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 $4 \times 300 = 1200$ Counts for darkdata and $4 \times 14000 = 56000$ 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:

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

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 $\text{ADCFactor} * (\text{measured counts (corrected for Dark)}) < 200$, use $\text{NCPS}[200] = 0.99203$

else if $\text{ADCFactor} * (\text{measured counts (corrected for Dark)}) > 15500$, use $\text{NCPS}[15500] = 0.96099$

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): $800 < \text{counts} < 62000$ will be corrected by the NCPS calculated by the polynomial.

Gain and Offset

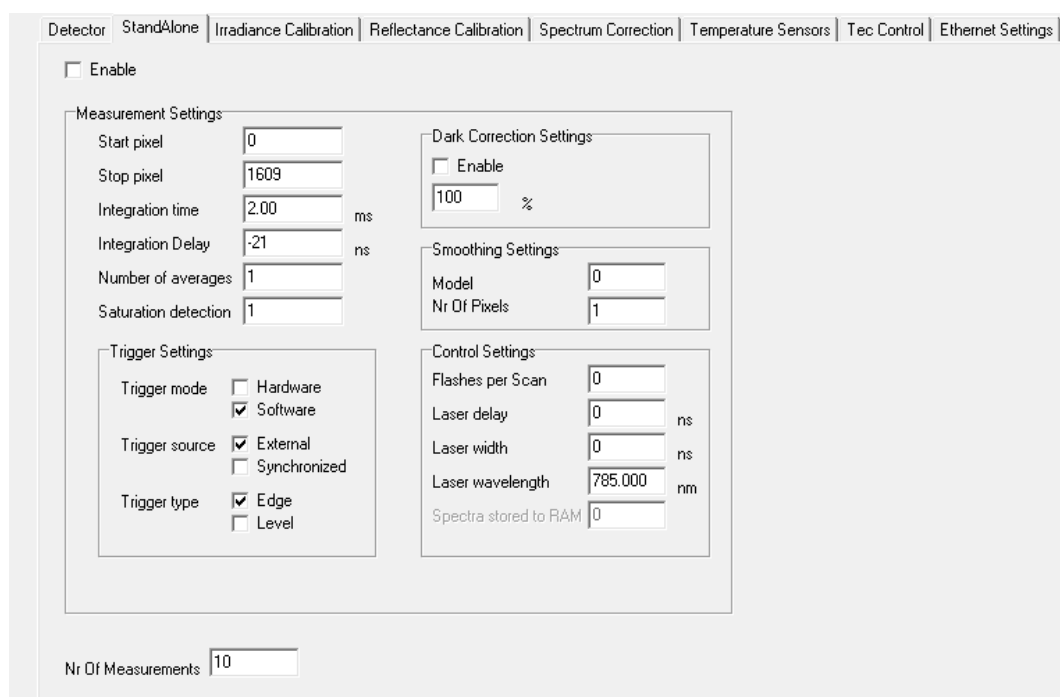
These parameters have been optimized by Avantes, and there should be no need to change these values. The m_Gain and m_Offset parameters are used to optimize the Gain and Offset of the AD Converter. Most detector types use only the m_Gain[0] and m_Offset[0]. The parameters m_Gain[1] and m_Offset[1] are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

Defective Pixels

The m_DefectivePixels[30] array can be used to store the pixelnumbers that should be eliminated from the data transfer. The AvaSpec dll will calculate the data for a defective pixel by interpolating the data of the neighbor pixels. A defective pixel can be specified in the range from 0 to "NrOfPixels-1", where NrOfPixels specifies the total pixels available for the detectortype used in the spectrometer (see also AVS_GetNumPixels).

The AvaSpec 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.

4.6.2 EEPROM structure: Standalone Parameters



Detector StandAlone Irradiance Calibration Reflectance Calibration Spectrum Correction Temperature Sensors Tec Control Ethernet Settings

☐ Enable

Measurement Settings

Start pixel: 0

Stop pixel: 1609

Integration time: 2.00 ms

Integration Delay: -21 ns

Number of averages: 1

Saturation detection: 1

Dark Correction Settings

☐ Enable

100 %

Smoothing Settings

Model: 0

Nr Of Pixels: 1

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: 785.000 nm

Spectra stored to RAM: 0

Nr Of Measurements: 10

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

The Measurement parameter structure (**MeasConfigType**) has been described in detail in section 4.2, as well as the **Number of Measurements** parameter (**m_Nmsr**).

4.6.3 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 [$\mu\text{Watt}/(\text{cm}^2 \cdot \text{nm})$]

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 3.3.28, 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. It is not yet used for calibration purposes by Avantes.
 The Spectrum Correction structure is used by Avantes for stray light correction purposes starting with DLL version 9.3.

4.6.4 EEPROM structure: Stray Light Correction

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

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

In the Qt4_demo_SLS sample program, `AVS_SupressStrayLight` is called to demonstrate SLS

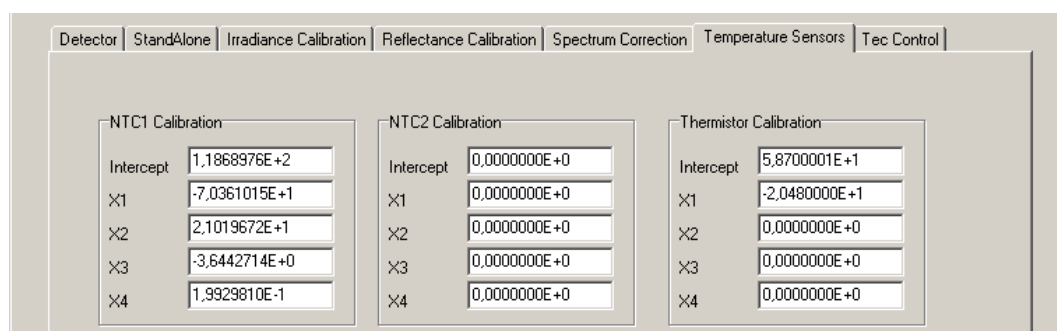
4.6.5 EEPROM structure: Temperature Sensors

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

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

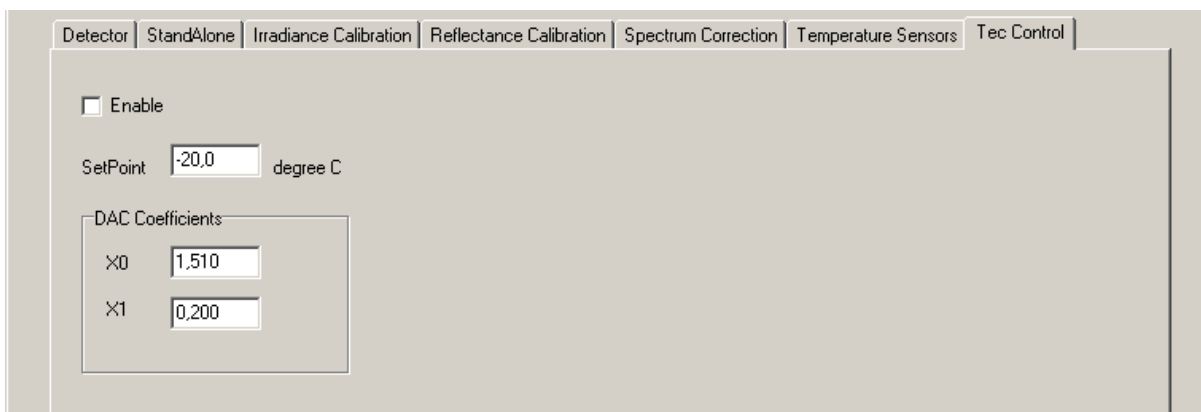
The AS7010 has a digital temperature sensor that already transmits degrees Celsius. Therefore the NTC1 calibration polynomial will be (0,1,0,0,0) for the AS7010.

The detector thermistor calibration of the AS7010 is identical to the AS5216 one.



NTC1 Calibration		NTC2 Calibration		Thermistor Calibration	
Intercept	1.1868976E+2	Intercept	0.0000000E+0	Intercept	5.8700001E+1
X1	-7.0361015E+1	X1	0.0000000E+0	X1	-2.0480000E+1
X2	2.1019672E+1	X2	0.0000000E+0	X2	0.0000000E+0
X3	-3.6442714E+0	X3	0.0000000E+0	X3	0.0000000E+0
X4	1.9929810E-1	X4	0.0000000E+0	X4	0.0000000E+0

4.6.6 EEPROM structure: Tec Control



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

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

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

Spectrometer	DetectorType	<code>m_aTemperature[2].m_aFit[0]</code>	<code>m_aTemperature[2].m_aFit[1]</code>
AvaSpec-NIR-2.0/2.5TEC	SENS_HAMS9201	58.70	-20.48
AvaSpec-NIR512-2.5-HSC	SENS_HAMG9208_512	58.70	-20.48
AvaSpec-NIR256-1.7/2.2TEC	SENS_SU256LSB	56.60	-18.58
AvaSpec-NIR512-1.7/2.2TEC	SENS_SU512LDB	56.60	-18.58
AvaSpec-2048TEC	SENS_ILX554	51.4	-16.38
AvaSpec-3648TEC	SENS_TCD1304	51.4	-16.38
AvaSpec-2048x64TEC	SENS_HAMS10420	51.4	-16.38
AvaSpec-HS1024x58	SENS_HAMS7031	82.15	-22.43
AvaSpec-HS1024x122	SENS_HAMS7031	82.15	-22.43

These coefficients are stored in the `TempSensorType` structure in the eeprom as described in section 4.6.5.

4.6.7 EEPROM structure: ProcessControl

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

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

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

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

4.6.8 EEPROM structure: Ethernet Settings

Ethernet Settings	
IP Address	192.168.11.4
Net Mask	255.255.255.0
Gateway	0.0.0.0
TCP Port	4500
DHCP Enabled	<input type="checkbox"/>
Link Status	<input type="checkbox"/>

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

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

The Link Status value is not used.

Appendix A: USB driver installation

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

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

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

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

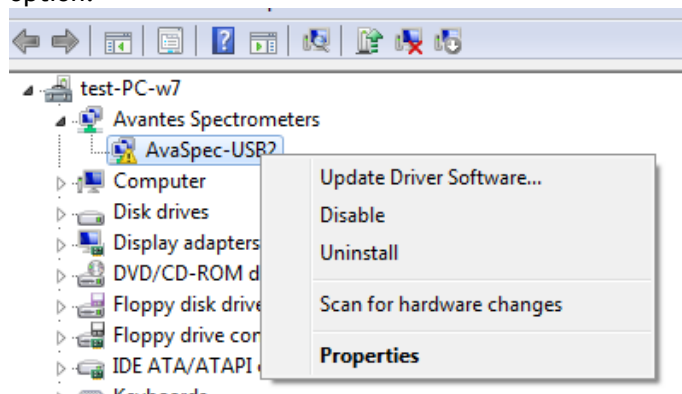
In this Appendix, some compatibility issues will be described that may occur after upgrading to WinUSB:

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

A1: Spectrometer cannot be found after update to WinUSB driver

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

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

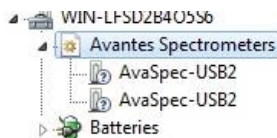


The 'Avantes Spectrometers' entry will show a yellow triangle with an exclamation mark. Right-click the AvaSpec-USB2 line and select the "Update Driver Software" option, as shown in the figure at the left. In the next dialog, select "Search automatically for updated driver software". Windows should now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



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

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



If the WinUSB driver has been installed properly, the Device Manager will display the connected devices without the yellow triangle with exclamation mark. The sample programs that are shipped with this version can be executed and everything runs well. However, after installing some application software, it is possible that the spectrometer cannot be

detected anymore. The sample programs cannot detect the AvaSpec-USB2 spectrometers either. One reason can be that the application software uses an old as5216.dll version 1.7 or earlier, as will be described below under A3.

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

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

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

Appendix B: Using the Ethernet spectrometer

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

B1: DHCP vs. using static IP addresses

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

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

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

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

B2: Cables

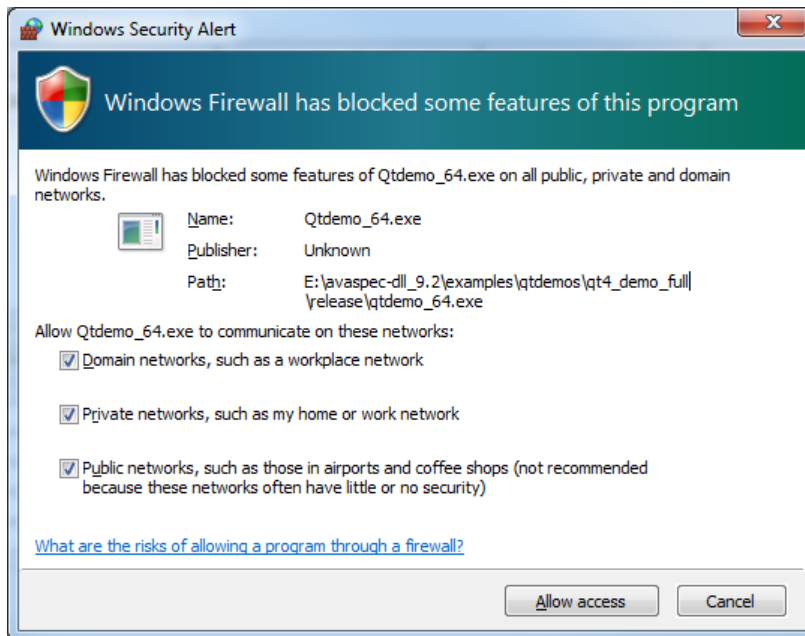
You can use standard Ethernet patch cables to connect the spectrometer to your network. If you use a direct connection to your PC, make sure you have a GigE network connection, if you want to use a standard Ethernet patch cable. If your PC has an older Fast Ethernet connection (100 Mbps), then you will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

B3: Routers

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

B4: Firewall

The Windows Firewall must allow both incoming and outgoing connections to the spectrometer. If the Windows Firewall is enabled, then a dialog will appear if you first use a program that will access the network:



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

B5: DLL / firmware version conflicts

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

B6: Connecting both interfaces at the same time

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