AvaSpec-Library

Interface Package for Linux Applications

Version 2.0.0.0

USER'S MANUAL
February 2016

Avantes B.V.

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

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

Microsoft, Visual C++ and Windows 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.
Qt is a trademark of Digia Plc. in Finland and/or other countries worldwide.

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

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.

1 Installation

The AvaSpec Library is a dynamically linked shared library. It was originally developed for the AS5216 board on Ubuntu Linux, version 11.04 and tested on diverse hardware, including desktop PCs and embedded boards. The present version also supports the later Avantes boards, including the Mini board and the AS7010 board.

At present, the library is available in binary form only. Binary versions are available for a number of Linux versions. We can recompile the library on request for a specific Linux version. A fee may be required for this service.

A sample C++ program using the library is available. It is written for Qt4 and includes source code.

Please refer to the documentation of the Windows DLL package as well, for a detailed description of some advanced parameters used in the sample programs.

Connecting the hardware

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

The AS7010 can also be connected to your network through an Ethernet cable. Depending on the presence of a DHCP server in your network, you may have to assign a fixed IP address to the board. It is recommended that you do this through the USB interface, using the IP settings utility.

2 Version History

This section will be used to describe the new features in the libavs.so.x.y.z, compared to the previous versions.

2.1 New in version 0.2.0

- Name change to AvaSpec Library.
- Support for the AvaSpec Mini and the AS7010 was added. The AS7010 adds USB 3.0 and Ethernet interfaces.
- The AVS_UpdateETHDevices function can be used for Ethernet connection management.
- The AVS_SetSensitivityMode function and the AVS_GetIpConfig functions were added.
- Detector support was expanded. It now includes support for CMOS detectors (Hamamatsu 11638 and 11639).

2.2 New in version 0.1.0

Although there is no previous version for libavs.so.0.1.0, a comparison can be made for programmers who have used the windows as5216.dll or avaspec.dll to write application software.

An effort was made to make the library compatible with the Windows version. Some functions were omitted, however, notably AVS_Register (which is Windows specific) and all functions that deal with the SD card on the AS5216 board.

The error messages that are returned by the different functions may not be identical to the ones from the Windows library.

3 Data acquisition

Just like with the Windows library, 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.

A big difference with the Windows DLL version is the fact that in Windows, AVS_Measure uses the Windows PostMessage function to signal that a new measurement is available. Linux has no message system, therefore in this library AVS_Measure uses a callback function for this purpose.

4 AvaSpec-Library description

4.1 Interface overview

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

4.2 Usage of the AvaSpec-Library

The Library uses a single pair of open and close functions (AVS_Init() and AVS_Done()) that have to be called by an application. As long as the open function is not yet 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 Library can be divided in four functional groups:

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

After the application has been initialized it should select the spectrometer(s) it wants to use.
For a USB connected device, the following steps have to be taken:

1. Call AVS_GetNrOfDevices c.q. AVS_UpdateUSBDevices to determine the number of attached devices
2. Allocate buffer to store the 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

4.3 Exported functions

4.3.1 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 AS5216 and Mini: Use USB port or preselected AvaGigE spectrometers with static IP address
AS7010: Use USB port

1..255 not supported in this version

256 AS5216 and Mini: Use USB port or preselected AvaGigE spectrometers with dynamic IP address
AS7010: Use Ethernet port

Return: On success, number of connected devices
On error, ERR_DEVICE_NOT_FOUND

4.3.2 AVS_Done

Function: **int AVS_Done**
(
Void
)

Group: Blocking control function

Description: Closes the communication and releases internal storage.

Parameters: None

Return: SUCCESS

4.3.3 AVS_GetNrOfDevices

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

4.3.4 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 in the device list. This number includes ETH devices.	
Parameters:	None	
Return:	> 0:	number of devices in the list
	0:	no devices found

4.3.5 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. This number includes 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

4.3.6 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

4.3.7 AVS_Activate

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

4.3.8 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

4.3.9 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

4.3.10 AVS_Measure OR AVS_MeasureCallBack

In the windows DLL version of this library the measure function with callback is available by calling AVS_MeasureCallBack. In the Linux library both functions do the same thing.

Function:	**int AVS_Measure**	
	(	
	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 Library will call the given function to notify a message is ready, the int* parameter will be set to the value SUCCESS new scan available, the number of scans that were saved in RAM (if StoreToRAM parameter > 0), or INVALID_MEAS_DATA. Set this value to NULL is 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

4.3.11 AVS_GetLambda

Function:	**int AVS_GetLambda**	
	(	
	AvsHandle	a_hDevice,
	double*	a_pWavelength
	)	
Group:	Internal data read function	
Description:	Returns the wavelength values corresponding to the pixels if available. This information is stored in the Library during the AVS_Activate() procedure. The Library does not test if a_pWaveLength is correctly allocated by the caller!	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate
	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

4.3.12 AVS_GetNumPixels

Function:	int AVS_GetNumPixels	
	(	
	AvsHandle	a_hDevice,
	unsigned short*	a_pNumPixels
	)	
Group:	Internal data read function	
Description:	Returns the number of pixels of a spectrometer. This information is stored in the Library during the AVS_Activate() procedure.	
Parameters:	a_hDevice:	device identifier returned by AVS_Activate
	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

4.3.13 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 Library 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)

4.3.14 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 the callback in which case the library will call the given function as soon as a measurement is ready to be imported into the application software.
But if the programming environment used does not support callback functions, 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 the situation should be avoided where AVS_PollScan is called continuously without any delay. This can cause such a heavy load on the CPU that the application software will freeze after a while. Adding a 1 millisecond delay to the polling loop (so polling every ms) will already solve 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

4.3.15 AVS_GetScopeData

Function: **int AVS_GetScopeData**
(
AvsHandle a_hDevice,
unsigned int* a_pTimeLabel,
double* a_pSpectrum
)

Group: Internal data read function,

Description: Returns the pixel values of the last performed measurement. Should be called by the application after the notification on AVS_Measure is triggered.
The Library does not check the allocated buffer size!

Parameters:	a_hDevice,	device identifier returned by AVS_Activate
	a_pTimeLabel,	ticks count last pixel of spectrum is received by microcontroller ticks in 10 µS units since spectrometer started
	a_pSpectrum	array of doubles, size equal to the selected pixelrange
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_MEAS_DATA (no measurement data received)

4.3.16 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

4.3.17 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
	a_pAnalogIn:	pointer to float for analog input value [Volts or degrees Celsius]
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_INVALID_PARAMETER (invalid analog input id.) ERR_TIMEOUT (error in communication)

4.3.18 AVS_GetDigIn

Function: **int AVS_GetDigIn**
(
AvsHandle a_hDevice,
unsigned char a_DigInId,
unsigned char* a_pDigIn
)

Group: Blocking control function.

Description: Returns the status of the specified digital input

Parameters:
a_hDevice: device identifier returned by AVS_Activate
a_DigInId: identifier of digital input

AS5216:
0 = DI1 = Pin 24 at 26-pins connector
1 = DI2 = Pin 7 at 26-pins connector
2 = DI3 = Pin 16 at 26-pins connector

Mini:
0 = DI1 = Pin 7 on Micro HDMI = Pin 5 on HDMI terminal
1 = DI2 = Pin 5 on Micro HDMI = Pin 3 on HDMI Terminal
2 = DI3 = Pin 3 on Micro HDMI = Pin 1 on HDMI Terminal
3 = DI4 = Pin 1 on Micro HDMI = Pin 19 on HDMI Terminal
4 = DI5 = Pin 4 on Micro HDMI = Pin 2 on HDMI Terminal
5 = DI6 = Pin 2 on Micro HDMI = Pin 14 on HDMI Terminal

AS7010:
0 = DI1 = Pin 24 at 26-pins connector
1 = DI2 = Pin 7 at 26-pins connector
2 = DI3 = Pin 16 at 26-pins

a_pDigIn: pointer to digital input status (0 – 1)

Return:
On success: ERR_SUCCESS, a_pDigIn contains valid value
On error: ERR_DEVICE_NOT_FOUND
ERR_INVALID_DEVICE_ID
ERR_INVALID_PARAMETER (invalid digital input id.)
ERR_TIMEOUT (error in communication)

4.3.19 AVS_GetVersionInfo

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

4.3.20 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)

4.3.21 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
	a_Value	DAC value to be set in Volts (internally an 8-bits DAC is used) with range 0 – 5.0V
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER

4.3.22 AVS_SetDigOut

Function: **int AVS_SetDigOut**
(
AvsHandle a_hDevice
unsigned char a_PortId,
unsigned char a_Value
)

Group: Blocking data send function.

Description: Sets the digital output value for the specified digital output

Parameters:
a_hDevice device identifier returned by AVS_Activate
a_PortId: identifier for one of the 10 output signals:

AS516:
0 = DO1 = pin 11 at 26-pins connector
1 = DO2 = pin 2 at 26-pins connector
2 = DO3 = pin 20 at 26-pins connector
3 = DO4 = pin 12 at 26-pins connector
4 = DO5 = pin 3 at 26-pins connector
5 = DO6 = pin 21 at 26-pins connector
6 = DO7 = pin 13 at 26-pins connector
7 = DO8 = pin 4 at 26-pins connector
8 = DO9 = pin 22 at 26-pins connector
9 = DO10 = pin 25 at 26-pins connector

Mini:
0 = DO1 = Pin 7 on Micro HDMI = Pin 5 on HDMI terminal
1 = DO2 = Pin 5 on Micro HDMI = Pin 3 on HDMI Terminal
2 = DO3 = Pin 3 on Micro HDMI = Pin 1 on HDMI Terminal
3 = DO4 = Pin 1 on Micro HDMI = Pin 19 on HDMI Terminal
4 = DO5 = Pin 4 on Micro HDMI = Pin 2 on HDMI Terminal
5 = DO6 = Pin 2 on Micro HDMI = Pin 14 on HDMI Terminal
6 = Not used
7 = Not used
8 = Not used
9 = Not used

AS7010:
0 = DO1 =pin 11 at 26-pins connector
1 = DO2 = pin 2 at 26-pins connector
2 = DO3 = pin 20 at 26-pins connector
3 = DO4 = pin 12 at 26-pins connector
4 = DO5 = pin 3 at 26-pins connector
5 = DO6 = pin 21 at 26-pins connector
6 = DO7 = pin 13 at 26-pins connector
7 = DO8 = pin 4 at 26-pins connector
8 = DO9 = pin 22 at 26-pins connector
9 = DO10 = pin 25 at 26-pins connector

a_Value: value to be set (0-1)

Return: On success: ERR_SUCCESS

	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER

4.3.23 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 The PWM functionality is not supported on the Mini
	a_Frequency	desired PWM frequency (500 – 300000) [Hz] For the AS5216, the frequency of outputs 0, 1 and 2 is the same (the last specified frequency is used) and also the frequency of outputs 4, 5 and 6 is the same. For the AS7010, you can define six different frequencies.
	a_DutyCycle	percentage high time in one cycle (0 – 100) For the AS5216, channels 0, 1 and 2 have a synchronized rising edge, the same holds for channels 4, 5 and 6. For the AS7010, rising edges are unsynchronized.
Return:	On success:	ERR_SUCCESS
	On error:	ERR_DEVICE_NOT_FOUND ERR_INVALID_DEVICE_ID ERR_TIMEOUT (error in communication) ERR_INVALID_PARAMETER

4.3.24 AVS_SetSyncMode

Function:	**int AVS_SetSyncMode**	
	(	
	AvsHandle	a_hDevice,
	unsigned char	a_Enable
	)	
Group:	Internal Library write function	
Description	Disables/enables support for synchronous measurement. Library takes care of dividing Nmsr request into Nmsr number of single measurement requests.	
Parameters	a_hDevice	master device identifier returned by AVS_Activate
	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

4.3.25 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

4.3.26 AVS_SetPrescanMode

Function: **int AVS_SetPrescanMode**
(
AvsHandle a_hDevice
bool a_Prescan
)

Group: Blocking data send function

Description: If a_Prescan is set, the first measurement result will be skipped. This function is only useful for the AvaSpec-3648 because this detector can be operated in prescan mode, or clearbuffer mode (see below)

Parameters: a_hDevice: device identifier returned by AVS_Activate
a_Prescan: If true, the first measurement result will be skipped (prescan mode), else the detector will be cleared before each new scan (clearbuffer mode)

Return: On success: ERR_SUCCESS
On error: ERR_DEVICE_NOT_FOUND
ERR_INVALID_DEVICE_ID
ERR_TIMEOUT (error in communication)

The Toshiba detector in the AvaSpec-3648, can be used in 2 different control modes:
The Prescan mode (default mode).
In this mode the Toshiba detector will automatically generate an additional prescan for every request from the PC, the first scan contains non-linear data and will be rejected, the 2nd scan contains linear data and will be sent to the PC. This prescan mode is default and should be used in most applications, like with averaging (only one prescan is generated for a nr of averages), with the use of an AvaLight-XE (one or more flashes per scan) and with multichannel spectrometers. The advantage of this mode is a very stable and linear spectrum. The disadvantage of this mode is that a minor (<5%) image of the previous scan (ghostspectrum) is included in the signal. This mode cannot be used 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.

4.3.27 AVS_UseHighResAdc

Function: **int AVS_UseHighResAdc**
(
AvsHandle a_hDevice
bool a_Enable
)

Group: Internal Library write function

Description: With the as5216 electronic board revision 1D and later, a 16bit resolution AD Converter is used instead of a 14bit in earlier hardware versions. As a result, the ADC Counts scale can be set to the full 16 bit (0..65535) Counts. For compatibility reasons with previous hardware revisions, the default range is set to 14 bit (0..16383.75) ADC Counts.

Remark: When using the 16 bit ADC in full High Resolution mode (0..65535), please note that the irradiance intensity calibration, as well as the nonlinearity calibration are based on the 14bit ADC range. Therefore, if using the nonlinearity correction or irradiance calibration in your own software using the High Resolution mode, you need to apply the additional correction with ADCFactor (= 4.0).

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

4.3.28 AVS_SetSensitivityMode

Function: **int AVS_SetSensitivityMode**
(
AvsHandle a_hDevice
unsigned int a_SensitivityMode
)

Group: Blocking data send function

Description: The AvaSpec-NIR models can be operated in LowNoise (a_SensitivityMode = 0) or High Sensitivity Mode (a_SensitivityMode > 0).

Parameters:
a_hDevice: device identifier returned by AVS_Activate
a_SensitivityMode: 0 = LowNoise, >0 = High Sensitivity

Return:
On success: ERR_SUCCESS
On error: ERR_DEVICE_NOT_FOUND
ERR_INVALID_DEVICE_ID
ERR_TIMEOUT (error in communication)
ERR_NOT_SUPPORTED_BY_SENSOR_TYPE
ERR_NOT_SUPPORTED_BY_FW_VER
ERR_NOT_SUPPORTED_BY_FPGA_VER

Remark: AVS_SetSensitivityMode is supported by the following detector types: HAMS9201, SU256LSB and SU512LDB. Calling this function for another detectortype will result in a return value of -120 (ERR_NOT_SUPPORTED_BY_SENSOR_TYPE)
This function requires a firmware function x.30.x.x or later. Calling this function for a spectrometer for which an older firmware version is loaded will result in a return value of -121 (ERR_NOT_SUPPORTED_BY_FW_VER).
The detector specific FPGA needs to support the sensitivity selection feature as well. The table below shows the minimum required version for the 3 detector types. Calling AVS_SetSensitivityMode for a spectrometer for which an older FPGA version is loaded will result in a return value of -122 (ERR_NOT_SUPPORTED_BY_FPGA_VER).
The table below also lists the Default Mode for each detector type. This is the mode in which the detector operates if the function AVS_SetSensitivityMode is not called. The default mode is also the mode that is used in models with older firmware and FPGA versions. Note that irradiance calibrated systems are calibrated in the default mode. Changing the sensitivity mode for an irradiance and/or nonlinearity calibrated system requires a recalibration of the system.

Spectrometer	**Detector Type**	**FPGA version**	**Default Mode**
AvaSpec-NIR256-1.7, AvaSpec-NIR256-2.0TEC, AvaSpec-NIR256-2.5TEC	SENS_HAMS9201	x.13.x.x	Low Noise
AvaSpec-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

4.3.29 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)

4.4 Data Elements

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

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

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

Type	Format		Value/Range	Description
DetectorType	struct			Sensor configuration structure:
	{			
	SensorType	m_SensorType,		sensor identification
	unsigned short	m_NrPixels,	0 – 4096	number of pixels of sensor
	float	m_aFit[5],		polynomial coefficients needed to determine wavelength
	bool	m_NLEnable,		enable/disable nonlinearity correction
	double	m_aNLCorrect[8],		polynomial coefficients needed for non-linearity correction
	double	m_aLowNLCounts,		lower counts limit for non-linearity correction
	double	m_aHighNLCounts,		higher counts limit for non-linearity correction
	float	m_Gain[2],	1 – 5.7	gain correction for spectrometer ADC (range is divided in 64 steps)
				not used
	float	m_Reserved,		offset correction for spectrometer ADC in Volt (range is divided in
	float	m_Offset[2],	-0.350 - +0.350	512 steps)
				offset to match the detector output range with the ADC range
	float	m_ExtOffset,	0.0 – 2.0	defective pixel numbers
	unsigned short	m_DefectivePixels[30],		
	}			(Size = 188 bytes)
Dynamic StorageType	{			
	int32	m_Nmsr,		Number of measurements (future use)
	uint8	m_Reserved[8]		For future use and backwards compatibility
	}			(Size = 12 bytes)
Ethernet SettingsType	struct			
	{			
	unsigned int	m_IpAddr;	0 – 0xFFFFFFFFFF	Static IP Address (when not using a DHCP server)
	unsigned int	m_NetMask;		Net Mask value (e.g. 255.255.255.0)
	unsigned int	m_Gateway;		Default gateway value (e.g. 192.168.1.254)
	unsigned char	m_DhcpEnabled;		0=Static IP Address used, 1=DHCP enabled
	unsigned short	m_TcpPort;		Default values is 4500, used to connect to spectrometer
	unsigned char	m_LinkStatus;		Reserved
	}			
				(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)

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

Type	Format	Value/Range	Description
SensorType	unsigned char	0 – 0x12	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

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)

Type	Format	Value/Range	Description

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)

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)

Type	Format	Value/Range	Description
TriggerType	struct		Trigger parameters
	{		
	unsigned char m_Mode,	0 – 1	mode, (0 = Software, 1 = Hardware)
	unsigned char m_Source,	0 – 1	trigger source, (0 = external trigger, 1 = sync input)
	unsigned char m_SourceType	0 – 1	source type, (0 = edge trigger, 1 = level trigger)
	}		Level triggering is only supported on the AS5216 board.
			(Size = 3 bytes)

4.4.1 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 Library
ERR_OPERATION_PENDING	-5	Function is called while result of previous call to AVS_Measure is not received yet.
ERR_TIMEOUT	-6	No answer received from device
Reserved	-7	
ERR_INVALID_MEAS_DATA	-8	No measurement data is received at the point AVS_GetScopeData is called
ERR_INVALID_SIZE	-9	Allocated buffer size too small
ERR_INVALID_PIXEL_RANGE	-10	Measurement preparation failed because pixel range is invalid
ERR_INVALID_INT_TIME	-11	Measurement preparation failed because integration time is invalid (for selected sensor)
ERR_INVALID_COMBINATION	-12	Measurement preparation failed because of an invalid combination of parameters, e.g. integration time of (600000) and (Navg > 5000)
Reserved	-13	
ERR_NO_MEAS_BUFFER_AVAIL	-14	Measurement preparation failed because no measurement buffers available
ERR_UNKNOWN	-15	Unknown error reason received from spectrometer
ERR_COMMUNICATION	-16	Error in communication occured
ERR_NO_SPECTRA_IN_RAM	-17	No more spectra available in RAM, all read or measurement not started yet.
ERR_INVALID_DLL_VERSION	-18	Library version information cannot be retrieved
ERR_NO_MEMORY	-19	Memory allocation error in the Library
ERR_DLL_INITIALISATION	-20	Function called before AVS_Init() is called
ERR_INVALID_STATE	-21	Function failed because AS5216 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

Return code	Value	Description
		compatible with the Averaging function
ERR_INVALID_MEASPARAM_AVG_RAM	-111	Use of Averaging is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_SYNC_RAM	-112	Use of the Synchronize setting is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_LEVEL_RAM	-113	Use of Level Triggering is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_SAT2_RAM	-114	Use of Saturation Detection Level 2 Parameter is not compatible with the StoreToRam function
ERR_INVALID_MEASPARAM_FWVER_RAM	-115	The StoreToRam function is only supported with firmware version 0.20.0.0 or later.
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

4.4.2 Callback function

The callback function takes two parameters, the first is the handle of the spectrometer acquired by AVS_Activate, the second is an integer value representing the value of the callback function. It is the equivalent of the WPARAM value in the Windows DLL.

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

Value	**Description**
0 (on success)	Measurement data is available.
> 0 (in StoreToRAM mode)	Value is the number of scans actually stored in RAM (which can be smaller than the amount requested)
< 0	The measurement failed. See table 4.4.1 for a description of the error message.